Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

S3C2410X 32-Bit RISC Microprocessor
User's Manual, Revision 1.2
Publication Number: 21.2-S3-C2410X-052003
© 2003  Samsung Electronics

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electric or mechanical, by photocopying, recording, or otherwise, without the prior written consent of Samsung Electronics.

Samsung Electronics’ microcontroller business has been awarded full ISO-9001 certification (BSI Certificate No. FM24653). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Ri, Giheung- Eup
Yongin-City, Gyeonggi-Do, Korea
C.P.O. Box #37, Suwon 449-900

TEL: (82)-(031)-209-1934
FAX: (82)-(031)-209-1899

Home Page: http://www.samsungsemi.com

Printed in the Republic of Korea
# Table of Contents

**Chapter 1  Product Overview**

- Introduction .................................................................................................................. 1-1  
- Features ......................................................................................................................... 1-2  
- Block Diagram ............................................................................................................. 1-5  
- Pin Assignments ........................................................................................................... 1-6  
  - Signal Descriptions .................................................................................................... 1-20  
  - S3C2410X Special Registers ..................................................................................... 1-25  

**Chapter 2  Programmer’s Model**

- Overview ....................................................................................................................... 2-1  
  - Processor Operating States ...................................................................................... 2-1  
  - Switching State ........................................................................................................ 2-1  
  - Memory Formats ........................................................................................................ 2-1  
  - Big-Endian Format ..................................................................................................... 2-2  
  - Little-Endian Format .................................................................................................. 2-2  
  - Instruction Length ...................................................................................................... 2-2  
  - Operating Modes ......................................................................................................... 2-3  
  - Registers .................................................................................................................... 2-3  
  - The Program Status Registers .................................................................................. 2-7  
  - Exceptions .................................................................................................................. 2-10  
  - Interrupt Latencies ................................................................................................... 2-15  
  - Reset ............................................................................................................................ 2-15
Chapter 3 ARM Instruction Set

Instruction Set Summary ................................................................. 3-1
Format Summary ........................................................................... 3-1
Instruction Summary ................................................................. 3-2
The Condition Field ........................................................................ 3-4
Branch and Exchange (BX) .......................................................... 3-5
Instruction Cycle Times .................................................................. 3-5
Assembler Syntax ........................................................................... 3-5
Using R15 As An Operand ............................................................ 3-5
Branch and Branch With Link (B, BL) ........................................... 3-7
The Link Bit .................................................................................. 3-7
Instruction Cycle Times .................................................................. 3-7
Assembler Syntax ........................................................................... 3-8
Data Processing ........................................................................... 3-9
Cpsr Flags .................................................................................... 3-11
Shifts ............................................................................................ 3-12
Immediate Operando Rotates .......................................................... 3-16
Writing to R15 .............................................................................. 3-16
Using R15 As an Operando ............................................................ 3-16
TEQ, TST, CMP and CMN Opcodes .............................................. 3-16
Instruction Cycle Times .................................................................. 3-16
Assembler Syntax ........................................................................... 3-17
Examples ....................................................................................... 3-17
PSR Transfer (MRS, MSR) .......................................................... 3-18
Operand Restrictions ...................................................................... 3-18
Reserved Bits .................................................................................. 3-20
Examples ....................................................................................... 3-20
Instruction Cycle Times .................................................................. 3-20
Assembly Syntax ............................................................................ 3-20
Examples ....................................................................................... 3-21
Multiply and Multiply-Accumulate (MUL, MLA) ......................... 3-22
CPSR Flags .................................................................................... 3-22
Instruction Cycle Times .................................................................. 3-22
Assembler Syntax ........................................................................... 3-24
Examples ....................................................................................... 3-24
Multiply Long and Multiply-Accumulate Long (MULL, MLAL) .... 3-25
Operand Restrictions ...................................................................... 3-26
Cpsr Flags .................................................................................... 3-26
Instruction Cycle Times .................................................................. 3-26
Assembler Syntax ........................................................................... 3-27
Examples ....................................................................................... 3-27
# Table of Contents (Continued)

## Chapter 3 ARM Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Data Transfer (LDR, STR)</td>
<td>3-28</td>
</tr>
<tr>
<td>Offsets and Auto-Indexing</td>
<td>3-29</td>
</tr>
<tr>
<td>Shifted Register Offset</td>
<td>3-29</td>
</tr>
<tr>
<td>Bytes and Words</td>
<td>3-29</td>
</tr>
<tr>
<td>Use Of R15</td>
<td>3-31</td>
</tr>
<tr>
<td>Example</td>
<td>3-31</td>
</tr>
<tr>
<td>Data Aborts</td>
<td>3-31</td>
</tr>
<tr>
<td>Instruction Cycle Times</td>
<td>3-31</td>
</tr>
<tr>
<td>Assembler Syntax</td>
<td>3-32</td>
</tr>
<tr>
<td>Examples</td>
<td>3-33</td>
</tr>
<tr>
<td>Halfword and Signed Data Transfer (LDRH/STRH/ LDRSB/LDRSH)</td>
<td>3-34</td>
</tr>
<tr>
<td>Offsets and Auto-Indexing</td>
<td>3-35</td>
</tr>
<tr>
<td>Halfword Load And Stores</td>
<td>3-36</td>
</tr>
<tr>
<td>Use of R15</td>
<td>3-37</td>
</tr>
<tr>
<td>Data Aborts</td>
<td>3-37</td>
</tr>
<tr>
<td>Instruction Cycle Times</td>
<td>3-37</td>
</tr>
<tr>
<td>Assembler Syntax</td>
<td>3-38</td>
</tr>
<tr>
<td>Examples</td>
<td>3-39</td>
</tr>
<tr>
<td>Block Data Transfer (LDM, STM)</td>
<td>3-40</td>
</tr>
<tr>
<td>The Register List</td>
<td>3-40</td>
</tr>
<tr>
<td>Addressing Modes</td>
<td>3-41</td>
</tr>
<tr>
<td>Address Alignment</td>
<td>3-41</td>
</tr>
<tr>
<td>Use of the S Bit</td>
<td>3-43</td>
</tr>
<tr>
<td>Use of R15 as the Base</td>
<td>3-43</td>
</tr>
<tr>
<td>Inclusion of the Base In the Register List</td>
<td>3-44</td>
</tr>
<tr>
<td>Data Aborts</td>
<td>3-44</td>
</tr>
<tr>
<td>Instruction Cycle Times</td>
<td>3-44</td>
</tr>
<tr>
<td>Assembler Syntax</td>
<td>3-45</td>
</tr>
<tr>
<td>Examples</td>
<td>3-46</td>
</tr>
<tr>
<td>Single Data Swap (SWP)</td>
<td>3-47</td>
</tr>
<tr>
<td>Bytes and Words</td>
<td>3-47</td>
</tr>
<tr>
<td>Use of R15</td>
<td>3-48</td>
</tr>
<tr>
<td>Data Aborts</td>
<td>3-48</td>
</tr>
<tr>
<td>Instruction Cycle Times</td>
<td>3-48</td>
</tr>
<tr>
<td>Assembler Syntax</td>
<td>3-48</td>
</tr>
<tr>
<td>Examples</td>
<td>3-49</td>
</tr>
<tr>
<td>Software Interrupt (SWI)</td>
<td>3-49</td>
</tr>
<tr>
<td>Return from the Supervisor</td>
<td>3-49</td>
</tr>
<tr>
<td>Comment Field</td>
<td>3-49</td>
</tr>
<tr>
<td>Instruction Cycle Times</td>
<td>3-49</td>
</tr>
<tr>
<td>Assembler Syntax</td>
<td>3-50</td>
</tr>
<tr>
<td>Coprocessor Data Operations (CDP)</td>
<td>3-51</td>
</tr>
<tr>
<td>Coprocessor Instructions</td>
<td>3-51</td>
</tr>
<tr>
<td>Instruction Cycle Times</td>
<td>3-52</td>
</tr>
<tr>
<td>Examples</td>
<td>3-52</td>
</tr>
</tbody>
</table>
### Table of Contents (Continued)

#### Chapter 3  ARM Instruction Set (Continued)

- Coprocessor Data Transfers (LDC, STC) ............................................................... 3-53
- The Coprocessor Fields ......................................................................................... 3-54
- Addressing Modes ................................................................................................. 3-54
- Address Alignment ................................................................................................. 3-54
- Data Aborts ........................................................................................................... 3-54
- Assembler Syntax ................................................................................................. 3-55
- Examples ............................................................................................................. 3-55
- Coprocessor Register Transfers (MRC, MCR) ................................................... 3-56
- The Coprocessor Fields ......................................................................................... 3-56
- Transfers to R15 .................................................................................................... 3-57
- Transfers from R15 ............................................................................................... 3-57
- Instruction Cycle Times ....................................................................................... 3-57
- Assembler Syntax ................................................................................................. 3-57
- Examples ............................................................................................................. 3-57
- Undefined Instruction ........................................................................................... 3-58
- Instruction Cycle Times ....................................................................................... 3-58
- Assembler Syntax ................................................................................................. 3-58
- Instruction Set Examples ..................................................................................... 3-59
- Using The Conditional Instructions .................................................................... 3-59
- Pseudo-Random Binary Sequence Generator .................................................... 3-61
- Multiplication by Constant Using the Barrel Shifter .......................................... 3-61
- Loading a Word from an Unknown Alignment ................................................... 3-63

#### Chapter 4  THUMB Instruction Set

- THUMB Instruction Set Format ........................................................................... 4-1
- Format Summary ................................................................................................... 4-2
- Opcode Summary ................................................................................................. 4-3
- Format 1: Move Shifted Register .......................................................................... 4-5
  - Operation ........................................................................................................ 4-5
  - Instruction Cycle Times ................................................................................... 4-6
  - Examples ....................................................................................................... 4-6
- Format 2: Add/Subtract ...................................................................................... 4-7
  - Operation ........................................................................................................ 4-7
  - Instruction Cycle Times ................................................................................... 4-8
  - Examples ....................................................................................................... 4-8
- Format 3: Move/Compare/Add/Subtract Immediate ......................................... 4-9
  - Operations ..................................................................................................... 4-9
  - Instruction Cycle Times ................................................................................... 4-10
  - Examples ..................................................................................................... 4-10
- Format 4: ALU Operations ............................................................................... 4-11
  - Operation ....................................................................................................... 4-11
  - Instruction Cycle Times ................................................................................... 4-12
  - Examples ..................................................................................................... 4-12
Table of Contents (Continued)

Chapter 4 THUMB Instruction Set (Continued)

Format 16: Conditional Branch ................................ ................................ ................................ ................ 4-34
  Operation ................................................................................................................................. 4-34
  Instruction Cycle Times ......................................................................................................... 4-35
  Examples ................................................................................................................................. 4-35
  Instruction Cycle Times ......................................................................................................... 4-36
  Examples ................................................................................................................................. 4-36
Format 18: Unconditional Branch ................................ ................................ ................................ ............ 4-37
  Operation ................................................................................................................................. 4-37
  Examples ................................................................................................................................. 4-37
  Instruction Cycle Times ......................................................................................................... 4-38
  Examples ................................................................................................................................. 4-39
Instruction Set Examples .............................................................................................................. 4-40
  Multiplication By a Constant Using Shifts and Adds ............................................................. 4-40
  General Purpose Signed Divide ............................................................................................. 4-41
  Division by a Constant ............................................................................................................ 4-43

Chapter 5 Memory Controller

Overview ........................................................................................................................................ 5-1
Function Description ..................................................................................................................... 5-3
  Bank0 Bus Width....................................................................................................................... 5-3
  Memory (SROM/SDRAM) Address Pin Connections ............................................................ 5-3
  SDRAM Bank Address Pin Connection .................................................................................. 5-4
  Nwait Pin Operation ................................................................................................................ 5-5
  nXBREQ/nXBACK Pin Operation ........................................................................................... 5-6
  Programmable Access Cycle .................................................................................................. 5-11
  Bus Width & Wait Control Register (BWCON) ..................................................................... 5-13
  Bank Control Register (BANKCONN: nGCS0-nGCS5) ......................................................... 5-15
  Bank Control Register (BANKCONn: nGCS6-nGCS7) ............................................................ 5-16
  Refresh Control Register ........................................................................................................ 5-17
  Banksize Register .................................................................................................................... 5-18
  SDRAM Mode Register Set Register (MRSR) ......................................................................... 5-19
Table of Contents (Continued)

Chapter 6 NAND Flash Controller

Overview ........................................................................................................................................ 6-1
Features ..................................................................................................................................... 6-1
Block Diagram ............................................................................................................................. 6-2
Operation Scheme ....................................................................................................................... 6-2
Auto Boot Mode Sequence .......................................................................................................... 6-3
Nand Flash Mode Configuration .................................................................................................. 6-3
Nand Flash Memory Timing ......................................................................................................... 6-3
Pin Configuration ......................................................................................................................... 6-4
Boot and Nand Flash Configurations ......................................................................................... 6-4
512-Byte ECC Parity Code Assignment Table .......................................................................... 6-4
Special Function Registers ........................................................................................................ 6-6
  Nand Flash Configuration (NFCONF) Register ................................................................. 6-6
  Nand Flash Command Set (NFCMD) Register ..................................................................... 6-7
  Nand Flash Address Set (NFADDR) Register ....................................................................... 6-7
  Nand Flash Data (NFDATA) Register ..................................................................................... 6-7
  Nand Flash Operation Status (NFSTAT) Register ............................................................... 6-7
  Nand Flash ECC (NFECC) Register ....................................................................................... 6-8

Chapter 7 Clock & Power Management

Overview ..................................................................................................................................... 7-1
Functional Description ............................................................................................................... 7-2
  Clock Architecture ................................................................................................................ 7-2
  Clock Source Selection .......................................................................................................... 7-2
  Phase Locked Loop (PLL) ..................................................................................................... 7-4
  Clock Control Logic ............................................................................................................. 7-6
  Power Management .............................................................................................................. 7-9
Clock Generator & Power Management Special Register .................................................. 7-19
  Lock Time Count Register (LOCKTIME) .......................................................................... 7-19
  PLL Value Selection Table ................................................................................................. 7-20
  Clock Control Register (CLKCON) .................................................................................... 7-21
  Clock Slow Control (CLKSLOW) Register .......................................................................... 7-22
  Clock Divider Control (CLKDIVN) Register ...................................................................... 7-22
# Table of Contents (Continued)

## Chapter 8 DMA

- Overview ........................................................................................................................................... 8-1  
- DMA Request Sources ....................................................................................................................... 8-2  
- DMA Operation ................................................................................................................................. 8-2  
- Examples ........................................................................................................................................... 8-6  
- DMA Special Registers ....................................................................................................................... 8-7  
  
  - DMA Initial Source (DISRC) Register .............................................................................................. 8-7  
  - DMA Initial Source Control (DISRCC) Register .............................................................................. 8-7  
  - DMA Initial Destination (DIDST) Register ..................................................................................... 8-8  
  - DMA Initial Destination Control (DIDSTC) Register ..................................................................... 8-8  
  - DMA Control (DCON) Register ...................................................................................................... 8-9  
  - DMA Status (DSTAT) Register ....................................................................................................... 8-11  
  - DMA Current Source (DCSRC) Register ......................................................................................... 8-11  
  - Current Destination (DCDST) Register ............................................................................................ 8-12  
  - DMA Mask Trigger (DMASKTRIG) Register ................................................................................. 8-13  

## Chapter 9 I/O ports

- Overview ........................................................................................................................................... 9-1  
- Port Control Descriptions .................................................................................................................. 9-7  
  
  - Port Configuration Register (GPACON-GPHCON) ...................................................................... 9-7  
  - Port Data Register (GPADAT-GPHDAT) ......................................................................................... 9-7  
  - Port Pull-Up Register (GPBUP-GPHUP) ......................................................................................... 9-7  
  - Miscellaneous Control Register ..................................................................................................... 9-7  
  - External Interrupt Control Register (EXTINTN) ........................................................................... 9-7  
  - Power Off Mode and I/O Ports ........................................................................................................ 9-7  
- I/O Port Control Register .................................................................................................................. 9-8  
  
  - Port A Control Registers (GPACON/GPADAT) ............................................................................. 9-8  
  - Port B Control Registers (GPBCON, GPBDAT, and GPBUP) ......................................................... 9-9  
  - Port C Control Registers (GPCCON, GPCDAT, and GPCUP) ....................................................... 9-10  
  - Port D Control Registers (GPDCON, GPDAT, and GPDUP) ......................................................... 9-12  
  - Port E Control Registers (GPECON, GPEDAT, and GPEUP) ......................................................... 9-14  
  - Port F Control Registers (GPFCON, GPFDAT, and GPFPU) ......................................................... 9-16  
  - Port G Control Registers (GPGCON, GPGDAT, AND GPGUP) ..................................................... 9-17  
  - Port H Control Registers (GPHCON, GPHDAT, AND GPHUP) ..................................................... 9-19  
  - Miscellaneous Control Register (MISCCR) ..................................................................................... 9-20  
  - DCLK Control Registers (DCLKCON) ............................................................................................. 9-21  
  - External Interrupt Control Register (EXTINTn) ............................................................................. 9-22  
  - External Interrupt Filter Register (EINTFLTn) ................................................................................. 9-25  
  - External Interrupt Mask Register (EINTMASK) .............................................................................. 9-26  
  - External Interrupt Pending Register (EINTPENDn) ......................................................................... 9-27  
  - General Status Register (GSTATUSn) ............................................................................................ 9-28
# Table of Contents (Continued)

## Chapter 10  PWM Timer

Overview ........................................................................................................................................... 10-1  
PWM Timer Operation ....................................................................................................................... 10-3  
   Prescaler & Divider ....................................................................................................................... 10-3  
   Auto Reload & Double Buffering ................................................................................................. 10-4  
   Timer Initialization Using Manual Update Bit and Inverter Bit ............................................... 10-5  
   Timer Operation ........................................................................................................................... 10-6  
   Pulse Width Modulation (PWM) ................................................................................................. 10-7  
   Output Level Control ................................................................................................................... 10-8  
   Dead Zone Generator ................................................................................................................ 10-9  
   Dma Request Mode .................................................................................................................... 10-10  
   PWM Timer Control Registers .................................................................................................. 10-11  
      Timer Configuration Register0 (TCFG0) ................................................................................. 10-11  
      Timer Configuration Register1 (TCFG1) ................................................................................. 10-12  
      Timer Control (TCON) Register ............................................................................................. 10-13  
      Timer 0 Count Buffer Register & Compare Buffer Register (TCNTB0/TCMPB0) ................. 10-15  
      Timer 1 Count Buffer Register & Compare Buffer Register (TCNTB1/TCMPB1) ................. 10-16  
      Timer 2 Count Buffer Register & Compare Buffer Register (TCNTB2/TCMPB2) ................. 10-17  
      Timer 3 Count Buffer Register & Compare Buffer Register (TCNTB3/TCMPB3) ................. 10-18  
      Timer 3 Count Observation Register (TCNTO3) ..................................................................... 10-18  
      Timer 4 Count Buffer Register (TCNTB4) ............................................................................. 10-19  

## Chapter 11  UART

Overview .......................................................................................................................................... 11-1  
   Features ....................................................................................................................................... 11-1  
   Block Diagram ............................................................................................................................ 11-2  
   UART Operation .......................................................................................................................... 11-3  
   UART Special Registers ............................................................................................................. 11-10  
      UART Line Control Register .................................................................................................. 11-10  
      UART Control Register ......................................................................................................... 11-11  
      UART FIFO Control Register ................................................................................................. 11-13  
      UART Modem Control Register ............................................................................................. 11-14  
      UART Tx/Rx Status Register .................................................................................................. 11-15  
      UART Error Status Register ................................................................................................. 11-16  
      UART FIFO Status Register ................................................................................................. 11-17  
      UART Modem Status Register ............................................................................................... 11-18  
      UART Transmit Buffer Register (Holding Register & FIFO Register) .................................. 11-19  
      UART Receive Buffer Register (Holding Register & FIFO Register) .................................... 11-19  
      UART Baud Rate Divisor Register ......................................................................................... 11-20
### Table of Contents (Continued)

**Chapter 12  USB Host Controller**
- Overview .................................................................................................................. 12-1
- USB Host Controller Special Registers ................................................................. 12-2

**Chapter 13  USB Device**
- Overview .................................................................................................................. 13-1
- Feature ..................................................................................................................... 13-1
- USB Device Controller Special Registers .............................................................. 13-3
  - Function Address Register (FUNC_ADDR_REG) ..................................................... 13-5
  - Power Management Register (PWR_REG) ............................................................ 13-6
  - Interrupt Register (EP_INT_REG/USB_INT_REG) ................................................... 13-7
  - Interrupt Enable Register (EP_INT_EN_REG/USB_INT_REG) .............................. 13-9
  - Frame Number Register (FPAME_NUM1_REG/FRAME_NUM2_REG) ...................... 13-10
  - Index Register (INDEX_REG) ................................................................................ 13-11
  - END Point0 Control Status Register (EP0_CSR) .................................................. 13-12
  - END Point In Control Status Register (IN_CSR1_REG/IN_CSR2_REG) ............... 13-14
  - END Point Out Control Status Register (OUT_CSR1_REG/OUT_CSR2_REG) .... 13-16
  - END Point FIFO Register (EPN_FIFO_REG) ......................................................... 13-18
  - MAX Packet Register (MAXP_REG) ...................................................................... 13-19
  - END Point Out Write Count Register (OUT_FIFO_CNT1_REG/OUT_FIFO_CNT2_REG) 13-20
  - DMA Interface Control Register (EPN_DMA_CON) .............................................. 13-21
  - DMA Unit Counter Register (EPN_DMA_UNIT) .................................................... 13-22
  - DMA FIFO Counter Register (EPN_DMA_FIFO) .................................................... 13-23
  - DMA Total Transfer Counter Register (EPN_DMA_TTC_L,M,H) ............................ 13-24
**Table of Contents (Continued)**

### Chapter 14  Interrupt Controller

- Overview .................................................................................................................. 14-1
- Interrupt Controller Operation ................................................................................ 14-2
- Interrupt Sources ...................................................................................................... 14-3
- Interrupt Priority Generating Block ........................................................................ 14-4
- Interrupt Priority ...................................................................................................... 14-5
- Interrupt Controller Special Registers ...................................................................... 14-6
  - Source Pending (Srcpnd) Register ......................................................................... 14-6
  - Interrupt Mode (INTMOD) Register .................................................................... 14-8
  - Interrupt Mask (INTMSK) Register ..................................................................... 14-10
  - Priority Register (PRIORITY) ............................................................................. 14-12
  - Interrupt Pending (INTPND) Register ................................................................ 14-14
  - Interrupt Offset (INTOFFSET) Register ............................................................ 14-16
  - SUB Source Pending (SUBSRCPND) Register ..................................................... 14-17
  - Interrupt Sub Mask (INTSUBMSK) Register ......................................................... 14-18

### Chapter 15  LCD Controller

- Overview .................................................................................................................. 15-1
- Features ..................................................................................................................... 15-1
- Common Features .................................................................................................... 15-2
- External Interface Signal .......................................................................................... 15-2
- Block Diagram .......................................................................................................... 15-3
- STN LCD Controller Operation .............................................................................. 15-4
  - Timing Generator (TIMEGEN) ........................................................................... 15-4
  - Video Operation .................................................................................................... 15-5
  - Dithering and Frame Rate Control .................................................................... 15-7
  - Memory Data Format (STN, BSWP=0) ................................................................. 15-9
- TFT LCD Controller Operation .............................................................................. 15-15
  - Video Operation .................................................................................................... 15-15
  - Memory Data Format (TFT) ................................................................................ 15-16
  - 256 Palette Usage (TFT) ..................................................................................... 15-20
  - Samsung TFT LCD Panel (3.5" Portrait / 256k Color /Reflective A-SI TFT LCD) .................................................................................................................. 15-23
  - Virtual Display (TFT/STN) ................................................................................ 15-24
  - LCD Power Enable (STN/TFT) .......................................................................... 15-25
  - LCD Controller Special Registers ....................................................................... 15-26
  - Frame Buffer Start Address 1 Register ................................................................. 15-32
Table of Contents (Continued)

Chapter 16  A/D Converter

Overview ........................................................................................................................................ 16-1
Features ......................................................................................................................................... 16-1
ADC & Touch Screen Interface Operation .................................................................................... 16-2
  Block Diagram ............................................................................................................................ 16-2
  Example for Touch Screen ........................................................................................................ 16-3
  Function Descriptions ................................................................................................................ 16-4
ADC and Touch Screen Interface Special Registers .................................................................... 16-7
  ADC Control (ADCCON) Register ............................................................................................ 16-7
  ADC Touch Screen Control (ADCTSC) Register .................................................................... 16-8
  ADC Start Delay (ADCDLY) Register ....................................................................................... 16-9
  ADC Conversion Data (ADCDAT1) Register ............................................................................. 16-11

Chapter 17  RTC (Real Time Clock)

Overview ........................................................................................................................................ 17-1
Features ......................................................................................................................................... 17-1
Real Time Clock Operation .......................................................................................................... 17-2
Leap Year Generator ..................................................................................................................... 17-2
Read/Write Registers .................................................................................................................. 17-3
  Alarm Function .......................................................................................................................... 17-3
  Tick Time Interrupt .................................................................................................................... 17-3
  Round Reset Function ............................................................................................................... 17-3
  32.768kHz X-Tal Connection Example ..................................................................................... 17-4
Real Time Clock Special Registers ............................................................................................... 17-5
  Real Time Clock Control (RTCCON) Register ...................................................................... 17-5
  Tick Time Count (TICNT) Register .......................................................................................... 17-5
  RTC Alarm Control (RTCALM) Register ............................................................................... 17-6
  Alarm Second Data (ALMSEC) Register ................................................................................. 17-7
  Alarm Min Data (ALMMIN) Register ....................................................................................... 17-7
  Alarm Hour Data (ALMHOUR) Register .................................................................................. 17-7
  Alarm Date Data (ALMDATE) Register .................................................................................... 17-8
  Alarm Mon Data (ALMMON) Register ..................................................................................... 17-8
  Alarm Year Data (ALMYEAR) Register .................................................................................... 17-8
  RTC Round Reset (RTCRST) Register ..................................................................................... 17-8
  BCD Second (BCDSEC) Register ............................................................................................ 17-9
  BCD Minute (BCDMIN) Register .............................................................................................. 17-9
  BCD Hour (BCDHOURL) Register ........................................................................................... 17-10
  BCD Date (BCDDATE) Register .............................................................................................. 17-10
  BCD Day (BCDDAY) Register ................................................................................................. 17-10
  BCD Month (BCDMON) Register ............................................................................................. 17-11
  BCD Year (BCDYEAR) Register ............................................................................................... 17-11
# Table of Contents (Continued)

## Chapter 18  Watchdog Timer

Overview ......................................................................................................................... 18-1  
Features ......................................................................................................................... 18-1  
Watchdog Timer Operation ......................................................................................... 18-2  
WTDAT & WTCNT ....................................................................................................... 18-2  
Consideration of Debugging Environment................................................................. 18-2  
Watchdog Timer Special Registers ........................................................................... 18-3  
Watchdog Timer Control (WTCON) Register ......................................................... 18-3  
Watchdog Timer Data (WTDAT) Register ................................................................. 18-4  
Watchdog Timer Count (WTCNT) Register ............................................................. 18-4  

## Chapter 19  SDI Interface

Overview ......................................................................................................................... 19-1  
Features ......................................................................................................................... 19-1  
Block Diagram ............................................................................................................... 19-2  
SDI Operation ............................................................................................................... 19-3  
SDIO Operation ............................................................................................................ 19-4  
SDI Special Registers ................................................................................................. 19-5  

## Chapter 20  IIC-Bus Interface

Overview ......................................................................................................................... 20-1  
The IIC-Bus Interface ................................................................................................. 20-3  
Start and Stop Conditions ......................................................................................... 20-3  
Data Transfer Format ................................................................................................. 20-4  
ACK Signal Transmission .......................................................................................... 20-5  
Read-Write Operation ............................................................................................... 20-6  
Bus Arbitration Procedures ..................................................................................... 20-6  
Abort Conditions ...................................................................................................... 20-6  
Configuring the IIC-Bus ............................................................................................ 20-6  
Flowcharts of the Operations in Each Mode ............................................................ 20-7  
IIC-Bus Interface Special Registers ......................................................................... 20-11  
Multi-Master IIC-Bus Control Register (IICCON) ................................................... 20-11  
Multi-Master IIC-Bus Address Register (IICADD) ................................................... 20-13  
Multi-Master IIC-Bus Transmit/Receive Data Shift Register (IICDS) ....................... 20-13  

---

S3C2410X MICROPROCESSOR  xv
<table>
<thead>
<tr>
<th>Chapter 21</th>
<th>IIS-Bus Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>21-1</td>
</tr>
<tr>
<td>Block Diagram</td>
<td>21-2</td>
</tr>
<tr>
<td>Functional Descriptions</td>
<td>21-2</td>
</tr>
<tr>
<td>Transmit or Receive only Mode</td>
<td>21-2</td>
</tr>
<tr>
<td>Audio Serial Interface Format</td>
<td>21-3</td>
</tr>
<tr>
<td>IIS-Bus Format</td>
<td>21-3</td>
</tr>
<tr>
<td>MSB (Left) Justified</td>
<td>21-3</td>
</tr>
<tr>
<td>Sampling Frequency and Master Clock</td>
<td>21-4</td>
</tr>
<tr>
<td>IIS-Bus Interface Special Registers</td>
<td>21-5</td>
</tr>
<tr>
<td>IIS Control (IISCON) Register</td>
<td>21-5</td>
</tr>
<tr>
<td>IIS Mode Register (IISMOD) Register</td>
<td>21-6</td>
</tr>
<tr>
<td>IIS Prescaler (IISPSR) Register</td>
<td>21-7</td>
</tr>
<tr>
<td>IIS FIFO Control (IISFCON) Register</td>
<td>21-8</td>
</tr>
<tr>
<td>IIS FIFO (IISFIFO) Register</td>
<td>21-8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 22</th>
<th>SPI Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>22-1</td>
</tr>
<tr>
<td>Features</td>
<td>22-1</td>
</tr>
<tr>
<td>Block Diagram</td>
<td>22-2</td>
</tr>
<tr>
<td>SPI Operation</td>
<td>22-3</td>
</tr>
<tr>
<td>SPI Special Registers</td>
<td>22-7</td>
</tr>
<tr>
<td>SPI Control Register</td>
<td>22-7</td>
</tr>
<tr>
<td>SPI Status Register</td>
<td>22-8</td>
</tr>
<tr>
<td>SPI Pin Control Register</td>
<td>22-9</td>
</tr>
</tbody>
</table>
Table of Contents (Concluded)

Chapter 23  BUS PRIORITIES

Overview ........................................................................................................................................... 23-1
Bus Priority Map ........................................................................................................................... 23-1

Chapter 24  Electrical Data

Absolute Maximum Ratings ............................................................................................................. 24-1
Recommended Operating Conditions ............................................................................................ 24-1
D.C. Electrical Characteristics ....................................................................................................... 24-2
A.C. Electrical Characteristics ....................................................................................................... 24-5

Chapter 25  Mechanical Data

Package Dimensions ....................................................................................................................... 25-1
## List of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Title</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>S3C2410X Block Diagram</td>
<td>1-5</td>
</tr>
<tr>
<td>1-2</td>
<td>S3C2410X Pin Assignments (272-FBGA)</td>
<td>1-6</td>
</tr>
<tr>
<td>2-1</td>
<td>Big-Endian Addresses of Bytes within Words</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>Little-Endian Addresses of Bytes within Words</td>
<td>2-2</td>
</tr>
<tr>
<td>2-3</td>
<td>Register Organization in ARM State</td>
<td>2-4</td>
</tr>
<tr>
<td>2-4</td>
<td>Register Organization in THUMB state</td>
<td>2-5</td>
</tr>
<tr>
<td>2-5</td>
<td>Mapping of THUMB State Registers onto ARM State Registers</td>
<td>2-6</td>
</tr>
<tr>
<td>2-6</td>
<td>Program Status Register Format</td>
<td>2-7</td>
</tr>
<tr>
<td>3-1</td>
<td>ARM Instruction Set Format</td>
<td>3-1</td>
</tr>
<tr>
<td>3-2</td>
<td>Branch and Exchange Instructions</td>
<td>3-5</td>
</tr>
<tr>
<td>3-3</td>
<td>Branch Instructions</td>
<td>3-7</td>
</tr>
<tr>
<td>3-4</td>
<td>Data Processing Instructions</td>
<td>3-9</td>
</tr>
<tr>
<td>3-5</td>
<td>ARM Shift Operations</td>
<td>3-12</td>
</tr>
<tr>
<td>3-6</td>
<td>Logical Shift Left</td>
<td>3-12</td>
</tr>
<tr>
<td>3-7</td>
<td>Logical Shift Right</td>
<td>3-13</td>
</tr>
<tr>
<td>3-8</td>
<td>Arithmetic Shift Right</td>
<td>3-13</td>
</tr>
<tr>
<td>3-9</td>
<td>Rotate Right</td>
<td>3-14</td>
</tr>
<tr>
<td>3-10</td>
<td>Rotate Right Extended</td>
<td>3-14</td>
</tr>
<tr>
<td>3-11</td>
<td>PSR Transfer</td>
<td>3-19</td>
</tr>
<tr>
<td>3-12</td>
<td>Multiply Instructions</td>
<td>3-22</td>
</tr>
<tr>
<td>3-13</td>
<td>Multiply Long Instructions</td>
<td>3-25</td>
</tr>
<tr>
<td>3-14</td>
<td>Single Data Transfer Instructions</td>
<td>3-28</td>
</tr>
<tr>
<td>3-15</td>
<td>Little-Endian Offset Addressing</td>
<td>3-30</td>
</tr>
<tr>
<td>3-16</td>
<td>Halfword and Signed Data Transfer with Register Offset</td>
<td>3-34</td>
</tr>
<tr>
<td>3-17</td>
<td>Halfword and Signed Data Transfer with Immediate Offset and Auto-Indexing</td>
<td>3-35</td>
</tr>
<tr>
<td>3-18</td>
<td>Block Data Transfer Instructions</td>
<td>3-40</td>
</tr>
<tr>
<td>3-19</td>
<td>Post-Increment Addressing</td>
<td>3-41</td>
</tr>
<tr>
<td>3-20</td>
<td>Pre-Increment Addressing</td>
<td>3-42</td>
</tr>
<tr>
<td>3-21</td>
<td>Post-Decrement Addressing</td>
<td>3-42</td>
</tr>
<tr>
<td>3-22</td>
<td>Pre-Decrement Addressing</td>
<td>3-43</td>
</tr>
<tr>
<td>3-23</td>
<td>Swap Instruction</td>
<td>3-47</td>
</tr>
<tr>
<td>3-24</td>
<td>Software Interrupt Instruction</td>
<td>3-49</td>
</tr>
<tr>
<td>3-25</td>
<td>Coprocessor Data Operation Instruction</td>
<td>3-51</td>
</tr>
<tr>
<td>3-26</td>
<td>Coprocessor Data Transfer Instructions</td>
<td>3-53</td>
</tr>
<tr>
<td>3-27</td>
<td>Coprocessor Register Transfer Instructions</td>
<td>3-56</td>
</tr>
<tr>
<td>3-28</td>
<td>Undefined Instruction</td>
<td>3-58</td>
</tr>
</tbody>
</table>
### List of Figures (Continued)

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Title</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1</td>
<td>THUMB Instruction Set Formats</td>
<td>4-2</td>
</tr>
<tr>
<td>4-2</td>
<td>Format 1</td>
<td>4-5</td>
</tr>
<tr>
<td>4-3</td>
<td>Format 2</td>
<td>4-7</td>
</tr>
<tr>
<td>4-4</td>
<td>Format 3</td>
<td>4-9</td>
</tr>
<tr>
<td>4-5</td>
<td>Format 4</td>
<td>4-11</td>
</tr>
<tr>
<td>4-6</td>
<td>Format 5</td>
<td>4-13</td>
</tr>
<tr>
<td>4-7</td>
<td>Format 6</td>
<td>4-16</td>
</tr>
<tr>
<td>4-8</td>
<td>Format 7</td>
<td>4-18</td>
</tr>
<tr>
<td>4-9</td>
<td>Format 8</td>
<td>4-20</td>
</tr>
<tr>
<td>4-10</td>
<td>Format 9</td>
<td>4-22</td>
</tr>
<tr>
<td>4-11</td>
<td>Format 10</td>
<td>4-24</td>
</tr>
<tr>
<td>4-12</td>
<td>Format 11</td>
<td>4-26</td>
</tr>
<tr>
<td>4-13</td>
<td>Format 12</td>
<td>4-28</td>
</tr>
<tr>
<td>4-14</td>
<td>Format 13</td>
<td>4-30</td>
</tr>
<tr>
<td>4-15</td>
<td>Format 14</td>
<td>4-31</td>
</tr>
<tr>
<td>4-16</td>
<td>Format 15</td>
<td>4-33</td>
</tr>
<tr>
<td>4-17</td>
<td>Format 16</td>
<td>4-34</td>
</tr>
<tr>
<td>4-18</td>
<td>Format 17</td>
<td>4-36</td>
</tr>
<tr>
<td>4-19</td>
<td>Format 18</td>
<td>4-37</td>
</tr>
<tr>
<td>4-20</td>
<td>Format 19</td>
<td>4-38</td>
</tr>
<tr>
<td>5-1</td>
<td>S3C2410X Memory Map after Reset</td>
<td>5-2</td>
</tr>
<tr>
<td>5-2</td>
<td>S3C2410X External nWAIT Timing Diagram (Tacc=4)</td>
<td>5-5</td>
</tr>
<tr>
<td>5-3</td>
<td>S3C2410X nXBREQ/nXBACK Timing Diagram</td>
<td>5-6</td>
</tr>
<tr>
<td>5-4</td>
<td>Memory Interface with 8-bit ROM</td>
<td>5-7</td>
</tr>
<tr>
<td>5-5</td>
<td>Memory Interface with 8-bit ROM × 2</td>
<td>5-7</td>
</tr>
<tr>
<td>5-6</td>
<td>Memory Interface with 8-bit ROM × 4</td>
<td>5-8</td>
</tr>
<tr>
<td>5-7</td>
<td>Memory Interface with 16-bit ROM</td>
<td>5-8</td>
</tr>
<tr>
<td>5-8</td>
<td>Memory Interface with 16-bit SRAM</td>
<td>5-9</td>
</tr>
<tr>
<td>5-9</td>
<td>Memory Interface with 16-bit SRAM × 2</td>
<td>5-9</td>
</tr>
<tr>
<td>5-10</td>
<td>Memory Interface with 16-bit SDRAM (8MB: 1Mb × 16 × 4banks)</td>
<td>5-10</td>
</tr>
<tr>
<td>5-11</td>
<td>Memory Interface with 16-bit SDRAM (16MB: 1Mb × 16 × 4banks × 2ea)</td>
<td>5-10</td>
</tr>
<tr>
<td>5-12</td>
<td>S3C2410X nGCS Timing Diagram</td>
<td>5-11</td>
</tr>
<tr>
<td>5-13</td>
<td>S3C2410X SDRAM Timing Diagram</td>
<td>5-12</td>
</tr>
</tbody>
</table>
List of Figures (Continued)

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Title</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-1</td>
<td>NAND Flash Controller Block Diagram</td>
<td>6-2</td>
</tr>
<tr>
<td>6-2</td>
<td>NAND Flash Operation Scheme</td>
<td>6-2</td>
</tr>
<tr>
<td>6-3</td>
<td>TACLS=0, TWRPH0=1, TWRPH1=0</td>
<td>6-3</td>
</tr>
<tr>
<td>6-4</td>
<td>NAND Flash Memory Mapping</td>
<td>6-5</td>
</tr>
<tr>
<td>7-1</td>
<td>Clock Generator Block Diagram</td>
<td>7-3</td>
</tr>
<tr>
<td>7-2</td>
<td>PLL (Phase-Locked Loop) Block Diagram</td>
<td>7-5</td>
</tr>
<tr>
<td>7-3</td>
<td>Main Oscillator Circuit Examples</td>
<td>7-5</td>
</tr>
<tr>
<td>7-4</td>
<td>Power-On Reset Sequence (when the external clock source is a crystal oscillator)</td>
<td>7-6</td>
</tr>
<tr>
<td>7-5</td>
<td>Changing Slow Clock by Setting PMS Value</td>
<td>7-7</td>
</tr>
<tr>
<td>7-6</td>
<td>Changing CLKDIVN Register Value</td>
<td>7-8</td>
</tr>
<tr>
<td>7-7</td>
<td>The Clock Distribution Block Diagram</td>
<td>7-9</td>
</tr>
<tr>
<td>7-8</td>
<td>Power Management State Diagram</td>
<td>7-10</td>
</tr>
<tr>
<td>7-9</td>
<td>Issuing Exit_from_Slow_mode Command in PLL on State</td>
<td>7-12</td>
</tr>
<tr>
<td>7-10</td>
<td>Issuing Exit_from_Slow_mode Command after Lock Time</td>
<td>7-12</td>
</tr>
<tr>
<td>7-11</td>
<td>Issuing Exit_from_Slow_mode Command and the Instant PLL_on Command Simultaneously</td>
<td>7-13</td>
</tr>
<tr>
<td>7-12</td>
<td>Power-OFF Mode</td>
<td>7-16</td>
</tr>
<tr>
<td>8-1</td>
<td>Basic DMA Timing Diagram</td>
<td>8-3</td>
</tr>
<tr>
<td>8-2</td>
<td>Demand/Handshake Mode Comparison</td>
<td>8-4</td>
</tr>
<tr>
<td>8-3</td>
<td>Burst 4 Transfer Size</td>
<td>8-5</td>
</tr>
<tr>
<td>8-4</td>
<td>Single service in Demand Mode with Unit Transfer Size</td>
<td>8-6</td>
</tr>
<tr>
<td>8-5</td>
<td>Single service in Handshake Mode with Unit Transfer Size</td>
<td>8-6</td>
</tr>
<tr>
<td>8-6</td>
<td>Whole service in Handshake Mode with Unit Transfer Size</td>
<td>8-6</td>
</tr>
<tr>
<td>10-1</td>
<td>16-bit PWM Timer Block Diagram</td>
<td>10-2</td>
</tr>
<tr>
<td>10-2</td>
<td>Timer Operations</td>
<td>10-3</td>
</tr>
<tr>
<td>10-3</td>
<td>Example of Double Buffering Function</td>
<td>10-4</td>
</tr>
<tr>
<td>10-4</td>
<td>Example of a Timer Operation</td>
<td>10-6</td>
</tr>
<tr>
<td>10-5</td>
<td>Example of PWM</td>
<td>10-7</td>
</tr>
<tr>
<td>10-6</td>
<td>Inverter On/Off</td>
<td>10-8</td>
</tr>
<tr>
<td>10-7</td>
<td>The Wave Form When a Dead Zone Feature is Enabled</td>
<td>10-9</td>
</tr>
<tr>
<td>10-8</td>
<td>Timer4 DMA Mode Operation</td>
<td>10-10</td>
</tr>
<tr>
<td>11-1</td>
<td>UART Block Diagram (with FIFO)</td>
<td>11-2</td>
</tr>
<tr>
<td>11-2</td>
<td>UART AFC Interface</td>
<td>11-4</td>
</tr>
<tr>
<td>11-3</td>
<td>UART Receiving 5 Characters with 2 Errors</td>
<td>11-6</td>
</tr>
<tr>
<td>11-4</td>
<td>IrDA Function Block Diagram</td>
<td>11-8</td>
</tr>
<tr>
<td>11-5</td>
<td>Serial I/O Frame Timing Diagram (Normal UART)</td>
<td>11-9</td>
</tr>
<tr>
<td>11-6</td>
<td>Infra-Red Transmit Mode Frame Timing Diagram</td>
<td>11-9</td>
</tr>
<tr>
<td>11-7</td>
<td>Infra-Red Receive Mode Frame Timing Diagram</td>
<td>11-9</td>
</tr>
<tr>
<td>11-8</td>
<td>nCTS and Delta CTS Timing Diagram</td>
<td>11-18</td>
</tr>
<tr>
<td>12-1</td>
<td>USB Host Controller Block Diagram</td>
<td>12-1</td>
</tr>
<tr>
<td>13-1</td>
<td>USB Device Controller Block Diagram</td>
<td>13-2</td>
</tr>
</tbody>
</table>
### List of Figures (Continued)

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Title</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-1</td>
<td>Interrupt Process Diagram</td>
<td>14-1</td>
</tr>
<tr>
<td>14-2</td>
<td>Priority Generating Block</td>
<td>14-4</td>
</tr>
<tr>
<td>15-1</td>
<td>LCD Controller Block Diagram</td>
<td>15-3</td>
</tr>
<tr>
<td>15-2</td>
<td>Monochrome Display Types (STN)</td>
<td>15-11</td>
</tr>
<tr>
<td>15-3</td>
<td>Color Display Types (STN)</td>
<td>15-12</td>
</tr>
<tr>
<td>15-4</td>
<td>8-bit Single Scan Display Type STN LCD Timing</td>
<td>15-14</td>
</tr>
<tr>
<td>15-5</td>
<td>16BPP Display Types (TFT)</td>
<td>15-21</td>
</tr>
<tr>
<td>15-6</td>
<td>TFT LCD Timing Example</td>
<td>15-22</td>
</tr>
<tr>
<td>15-7</td>
<td>Example of Scrolling in Virtual Display (Single Scan)</td>
<td>15-24</td>
</tr>
<tr>
<td>15-8</td>
<td>Example of PWREN Function (PWREN = 1, INVPWREN = 0)</td>
<td>15-25</td>
</tr>
<tr>
<td>16-1</td>
<td>ADC and Touch Screen Interface Functional Block Diagram</td>
<td>16-2</td>
</tr>
<tr>
<td>16-2</td>
<td>Example of ADC and Touch Screen Interface</td>
<td>16-3</td>
</tr>
<tr>
<td>16-3</td>
<td>Timing Diagram in Auto (Sequential) X/Y Position Conversion Mode</td>
<td>16-6</td>
</tr>
<tr>
<td>17-1</td>
<td>Real Time Clock Block Diagram</td>
<td>17-2</td>
</tr>
<tr>
<td>17-2</td>
<td>Main Oscillator Circuit Example</td>
<td>17-4</td>
</tr>
<tr>
<td>18-1</td>
<td>Watchdog Timer Block Diagram</td>
<td>18-2</td>
</tr>
<tr>
<td>19-1</td>
<td>Block Diagram</td>
<td>19-2</td>
</tr>
<tr>
<td>20-1</td>
<td>IIC-Bus Block Diagram</td>
<td>20-2</td>
</tr>
<tr>
<td>20-2</td>
<td>Start and Stop Condition</td>
<td>20-3</td>
</tr>
<tr>
<td>20-3</td>
<td>IIC-Bus Interface Data Format</td>
<td>20-4</td>
</tr>
<tr>
<td>20-4</td>
<td>Data Transfer on the IIC-Bus</td>
<td>20-5</td>
</tr>
<tr>
<td>20-5</td>
<td>Acknowledge on the IIC-Bus</td>
<td>20-5</td>
</tr>
<tr>
<td>20-8</td>
<td>Operations for Slave/Transmitter Mode</td>
<td>20-9</td>
</tr>
<tr>
<td>20-9</td>
<td>Operations for Slave/Receiver Mode</td>
<td>20-10</td>
</tr>
<tr>
<td>21-1</td>
<td>IIS-Bus Block Diagram</td>
<td>21-2</td>
</tr>
<tr>
<td>21-2</td>
<td>IIS-Bus and MSB (Left)-justified Data Interface Formats</td>
<td>21-4</td>
</tr>
<tr>
<td>22-1</td>
<td>SPI Block Diagram</td>
<td>22-2</td>
</tr>
<tr>
<td>24-1</td>
<td>Typical Operating Voltage/Frequency Range (VDDIO=3.3V, @Room temperature &amp; SMDK2410 board)</td>
<td>24-4</td>
</tr>
<tr>
<td>24-2</td>
<td>XT pll Clock Timing</td>
<td>24-5</td>
</tr>
<tr>
<td>24-3</td>
<td>EXTCLK Clock Input Timing</td>
<td>24-5</td>
</tr>
<tr>
<td>24-4</td>
<td>EXTCLK/HCLK in case that EXTCLK is used without the PLL</td>
<td>24-5</td>
</tr>
<tr>
<td>24-5</td>
<td>HCLK/CLKOUT/SCLK in case that EXTCLK is used</td>
<td>24-6</td>
</tr>
</tbody>
</table>
## List of Figures (Concluded)

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Title</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-6</td>
<td>Manual Reset Input Timing</td>
<td>24-6</td>
</tr>
<tr>
<td>24-7</td>
<td>Power-On Oscillation Setting Timing</td>
<td>24-7</td>
</tr>
<tr>
<td>24-8</td>
<td>Power OFF Mode Return Oscillation Setting Timing</td>
<td>24-8</td>
</tr>
<tr>
<td>24-9</td>
<td>ROM/SRAM Burst READ Timing(I)</td>
<td>24-9</td>
</tr>
<tr>
<td></td>
<td>(Tacs=0, Tcos=0, Tacc=2, Toch=0, Tcah=0, PMC=0, ST=0, DW=16bit)</td>
<td></td>
</tr>
<tr>
<td>24-10</td>
<td>ROM/SRAM Burst READ Timing(II)</td>
<td>24-10</td>
</tr>
<tr>
<td></td>
<td>(Tacs=0, Tcos=0, Tacc=2, Toch=0, Tcah=0, PMC=0, ST=1, DW=16bit)</td>
<td></td>
</tr>
<tr>
<td>24-11</td>
<td>External Bus Request in ROM/SRAM Cycle</td>
<td>24-11</td>
</tr>
<tr>
<td></td>
<td>(Tacs=0, Tcos=0, Tacc=8, Toch=0, Tcah=0, PMC=0, ST=0)</td>
<td></td>
</tr>
<tr>
<td>24-12</td>
<td>ROM/SRAM READ Timing(I)</td>
<td>24-12</td>
</tr>
<tr>
<td></td>
<td>(Tacs=2, Tcos=2, Tacc=2, Toch=2, Tcah=2, PMC=0, ST=0)</td>
<td></td>
</tr>
<tr>
<td>24-13</td>
<td>ROM/SRAM READ Timing(II)</td>
<td>24-13</td>
</tr>
<tr>
<td></td>
<td>(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2cycle, PMC=0, ST=1)</td>
<td></td>
</tr>
<tr>
<td>24-14</td>
<td>ROM/SRAM WRITE Timing(I)</td>
<td>24-14</td>
</tr>
<tr>
<td></td>
<td>(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2, PMC=0, ST=0)</td>
<td></td>
</tr>
<tr>
<td>24-15</td>
<td>ROM/SRAM WRITE Timing(II)</td>
<td>24-15</td>
</tr>
<tr>
<td></td>
<td>(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2, PMC=0, ST=1)</td>
<td></td>
</tr>
<tr>
<td>24-16</td>
<td>External nWAIT READ Timing</td>
<td>24-16</td>
</tr>
<tr>
<td></td>
<td>(Tacs=0, Tcos=0, Tacc=6, Toch=0, Tcah=0, PMC=0, ST=0)</td>
<td></td>
</tr>
<tr>
<td>24-17</td>
<td>External nWAIT WRITE Timing</td>
<td>24-17</td>
</tr>
<tr>
<td></td>
<td>(Tacs=0, Tcos=0, Tacc=4, Toch=0, Tcah=0, PMC=0, ST=0)</td>
<td></td>
</tr>
<tr>
<td>24-18</td>
<td>Masked-ROM Single READ Timing (Tacs=2, Tcos=2, Tacc=8, PMC=01/10/11)</td>
<td>24-17</td>
</tr>
<tr>
<td>24-19</td>
<td>Masked-ROM Consecutive READ Timing</td>
<td>24-17</td>
</tr>
<tr>
<td></td>
<td>(Tacs=0, Tcos=0, Tacc=3, Tpac=2, PMC=01/10/11)</td>
<td></td>
</tr>
<tr>
<td>24-20</td>
<td>SDRAM Single Burst READ Timing (Trp=2, Trcd=2, Tcl=2, DW=16bit)</td>
<td>24-18</td>
</tr>
<tr>
<td>24-21</td>
<td>External Bus Request in SDRAM Timing (Trp=2, Trcd=2, Tcl=2)</td>
<td>24-19</td>
</tr>
<tr>
<td>24-22</td>
<td>SDRAM MRS Timing</td>
<td>24-20</td>
</tr>
<tr>
<td>24-23</td>
<td>SDRAM Single READ Timing(I)</td>
<td>24-21</td>
</tr>
<tr>
<td></td>
<td>(Trp=2, Trcd=2, Tcl=2)</td>
<td></td>
</tr>
<tr>
<td>24-24</td>
<td>SDRAM Single READ Timing(II)</td>
<td>24-22</td>
</tr>
<tr>
<td></td>
<td>(Trp=2, Trcd=2, Tcl=3)</td>
<td></td>
</tr>
<tr>
<td>24-25</td>
<td>SDRAM Auto Refresh Timing</td>
<td>24-23</td>
</tr>
<tr>
<td></td>
<td>(Trp=2, Trc=4)</td>
<td></td>
</tr>
<tr>
<td>24-26</td>
<td>SDRAM Page Hit-Miss READ Timing (Trp=2, Trcd=2, Tcl=2)</td>
<td>24-24</td>
</tr>
<tr>
<td>24-27</td>
<td>SDRAM Self Refresh Timing</td>
<td>24-25</td>
</tr>
<tr>
<td></td>
<td>(Trp=2, Trc=4)</td>
<td></td>
</tr>
<tr>
<td>24-28</td>
<td>SDRAM Single Write Timing</td>
<td>24-26</td>
</tr>
<tr>
<td></td>
<td>(Trp=2, Trcd=2)</td>
<td></td>
</tr>
<tr>
<td>24-29</td>
<td>SDRAM Page Hit-Miss Write Timing (Trp=2, Trcd=2, Tcl=2)</td>
<td>24-27</td>
</tr>
<tr>
<td>24-30</td>
<td>External DMA Timing (Handshake, Single transfer)</td>
<td>24-28</td>
</tr>
<tr>
<td>24-31</td>
<td>TFT LCD Controller Timing</td>
<td>24-28</td>
</tr>
<tr>
<td>24-32</td>
<td>IIS Interface Timing</td>
<td>24-29</td>
</tr>
<tr>
<td>24-33</td>
<td>IIC Interface Timing</td>
<td>24-29</td>
</tr>
<tr>
<td>24-34</td>
<td>SD/MMC Interface Timing</td>
<td>24-30</td>
</tr>
<tr>
<td>24-35</td>
<td>SPI Interface Timing (CPHA=1, CPOL=1)</td>
<td>24-30</td>
</tr>
<tr>
<td>24-36</td>
<td>NAND Flash Address/Command Timing</td>
<td>24-31</td>
</tr>
<tr>
<td>24-37</td>
<td>NAND Flash Data Timing</td>
<td>24-31</td>
</tr>
<tr>
<td>25-1</td>
<td>272-FBGA-1414 Package Dimension 1</td>
<td>25-1</td>
</tr>
<tr>
<td>25-2</td>
<td>272-FBGA-1414 Package Dimension 2</td>
<td>25-2</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Title</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>272-Pin FBGA Pin Assignments – Pin Number Order (Sheet 1 of 3)</td>
<td>1-7</td>
</tr>
<tr>
<td>1-2</td>
<td>272-Pin FBGA Pin Assignments (Sheet 1 of 9)</td>
<td>1-10</td>
</tr>
<tr>
<td>1-3</td>
<td>S3C2410X Signal Descriptions (Sheet 1 of 5)</td>
<td>1-20</td>
</tr>
<tr>
<td>1-4</td>
<td>S3C2410X Special Registers (Sheet of 1 of 11)</td>
<td>1-25</td>
</tr>
<tr>
<td>2-1</td>
<td>PSR Mode Bit Values</td>
<td>2-9</td>
</tr>
<tr>
<td>2-2</td>
<td>Exception Entry/Exit</td>
<td>2-11</td>
</tr>
<tr>
<td>2-3</td>
<td>Exception Vectors</td>
<td>2-13</td>
</tr>
<tr>
<td>3-1</td>
<td>The ARM Instruction Set</td>
<td>3-2</td>
</tr>
<tr>
<td>3-2</td>
<td>Condition Code Summary</td>
<td>3-4</td>
</tr>
<tr>
<td>3-3</td>
<td>ARM Data Processing Instructions</td>
<td>3-11</td>
</tr>
<tr>
<td>3-4</td>
<td>Incremental Cycle Times</td>
<td>3-16</td>
</tr>
<tr>
<td>3-5</td>
<td>Assembler Syntax Descriptions</td>
<td>3-27</td>
</tr>
<tr>
<td>3-6</td>
<td>Addressing Mode Names</td>
<td>3-45</td>
</tr>
<tr>
<td>4-1</td>
<td>THUMB Instruction Set Opcodes</td>
<td>4-3</td>
</tr>
<tr>
<td>4-2</td>
<td>Summary of Format 1 Instructions</td>
<td>4-5</td>
</tr>
<tr>
<td>4-3</td>
<td>Summary of Format 2 Instructions</td>
<td>4-7</td>
</tr>
<tr>
<td>4-4</td>
<td>Summary of Format 3 Instructions</td>
<td>4-9</td>
</tr>
<tr>
<td>4-5</td>
<td>Summary of Format 4 Instructions</td>
<td>4-11</td>
</tr>
<tr>
<td>4-6</td>
<td>Summary of Format 5 Instructions</td>
<td>4-13</td>
</tr>
<tr>
<td>4-7</td>
<td>Summary of PC-Relative Load Instruction</td>
<td>4-16</td>
</tr>
<tr>
<td>4-8</td>
<td>Summary of Format 7 Instructions</td>
<td>4-19</td>
</tr>
<tr>
<td>4-9</td>
<td>Summary of format 8 instructions</td>
<td>4-20</td>
</tr>
<tr>
<td>4-10</td>
<td>Summary of Format 9 Instructions</td>
<td>4-23</td>
</tr>
<tr>
<td>4-11</td>
<td>Halfword Data Transfer Instructions</td>
<td>4-24</td>
</tr>
<tr>
<td>4-12</td>
<td>SP-Relative Load/Store Instructions</td>
<td>4-26</td>
</tr>
<tr>
<td>4-13</td>
<td>Load Address</td>
<td>4-28</td>
</tr>
<tr>
<td>4-14</td>
<td>The ADD SP Instruction</td>
<td>4-30</td>
</tr>
<tr>
<td>4-15</td>
<td>PUSH and POP Instructions</td>
<td>4-31</td>
</tr>
<tr>
<td>4-16</td>
<td>The Multiple Load/Store Instructions</td>
<td>4-33</td>
</tr>
<tr>
<td>4-17</td>
<td>The Conditional Branch Instructions</td>
<td>4-34</td>
</tr>
<tr>
<td>4-18</td>
<td>The SWI Instruction</td>
<td>4-36</td>
</tr>
<tr>
<td>4-19</td>
<td>Summary of Branch Instruction</td>
<td>4-37</td>
</tr>
<tr>
<td>4-20</td>
<td>The BL Instruction</td>
<td>4-39</td>
</tr>
<tr>
<td>5-1</td>
<td>Bank 6/7 Address</td>
<td>5-2</td>
</tr>
<tr>
<td>5-2</td>
<td>SDRAM Bank Address configuration</td>
<td>5-4</td>
</tr>
<tr>
<td>7-1</td>
<td>Clock Source Selection at Boot-Up</td>
<td>7-2</td>
</tr>
<tr>
<td>7-2</td>
<td>Clock and Power State in Each Power Mode</td>
<td>7-10</td>
</tr>
<tr>
<td>7-3</td>
<td>CLKSLOW and CLKDIVN Register Settings for SLOW Clock</td>
<td>7-11</td>
</tr>
</tbody>
</table>
List of Tables (Continued)

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Title</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-1</td>
<td>DMA request sources for each channel</td>
<td>8-2</td>
</tr>
<tr>
<td>8-2</td>
<td>DMA Controller Module Signal Timing Constants</td>
<td>8-3</td>
</tr>
<tr>
<td>9-1</td>
<td>S3C2410X Port Configuration Overview (Sheet 1 of 5)</td>
<td>9-2</td>
</tr>
<tr>
<td>11-1</td>
<td>Interrupts in Connection with FIFO</td>
<td>11-5</td>
</tr>
<tr>
<td>12-1</td>
<td>OHCI Registers for USB Host Controller</td>
<td>12-1</td>
</tr>
<tr>
<td>15-1</td>
<td>Relation Between VCLK and CLKVAL (STN, HCLK = 60 MHz)</td>
<td>15-5</td>
</tr>
<tr>
<td>15-2</td>
<td>Dither Duty Cycle Examples</td>
<td>15-7</td>
</tr>
<tr>
<td>15-3</td>
<td>Relation Between VCLK and CLKVAL (TFT, HCLK = 60 MHz)</td>
<td>15-15</td>
</tr>
<tr>
<td>15-4</td>
<td>5:6:5 Format</td>
<td>15-20</td>
</tr>
<tr>
<td>15-5</td>
<td>5:5:5:1 Format</td>
<td>15-20</td>
</tr>
<tr>
<td>15-6</td>
<td>MV Value for Each Display Mode</td>
<td>15-39</td>
</tr>
<tr>
<td>16-1</td>
<td>Condition of Touch Screen Panel Pads in Separate X/Y Position</td>
<td>16-4</td>
</tr>
<tr>
<td>16-2</td>
<td>Condition of Touch Screen Panel Pads in Auto (Sequential) X/Y Position Conversion Mode</td>
<td>16-5</td>
</tr>
<tr>
<td>16-3</td>
<td>Condition of Touch Screen Panel Pads in Waiting for Interrupt Mode.</td>
<td>16-5</td>
</tr>
<tr>
<td>21-1</td>
<td>CODEC clock (CODECLK = 256 or 384fs)</td>
<td>21-4</td>
</tr>
<tr>
<td>21-2</td>
<td>Usable Serial Bit Clock Frequency (IISCLK = 16 or 32 or 48fs)</td>
<td>21-5</td>
</tr>
<tr>
<td>24-1</td>
<td>Absolute Maximum Rating</td>
<td>24-1</td>
</tr>
<tr>
<td>24-2</td>
<td>Recommended Operating Conditions</td>
<td>24-1</td>
</tr>
<tr>
<td>24-3</td>
<td>Normal I/O PAD DC Electrical Characteristics</td>
<td>24-2</td>
</tr>
<tr>
<td>24-4</td>
<td>USB DC Electrical Characteristics</td>
<td>24-3</td>
</tr>
<tr>
<td>24-5</td>
<td>S3C2410X Power Supply Voltage and Current</td>
<td>24-3</td>
</tr>
<tr>
<td>24-6</td>
<td>Typical Current Decrease by CLKCON Register (<a href="mailto:FCLK@202.8MHz">FCLK@202.8MHz</a>)</td>
<td>24-4</td>
</tr>
<tr>
<td>24-7</td>
<td>Clock Timing Constants</td>
<td>24-32</td>
</tr>
<tr>
<td>24-8</td>
<td>ROM/ SRAM Bus Timing Constants</td>
<td>24-33</td>
</tr>
<tr>
<td>24-9</td>
<td>Memory Interface Timing Constants (3.3V)</td>
<td>24-33</td>
</tr>
<tr>
<td>24-10</td>
<td>External Bus Request Timing Constants</td>
<td>24-34</td>
</tr>
<tr>
<td>24-11</td>
<td>DMA Controller Module Signal Timing Constants</td>
<td>24-34</td>
</tr>
<tr>
<td>24-12</td>
<td>TFT LCD Controller Module Signal Timing Constants</td>
<td>24-35</td>
</tr>
<tr>
<td>24-13</td>
<td>IIS Controller Module Signal Timing Constants</td>
<td>24-35</td>
</tr>
<tr>
<td>24-14</td>
<td>IIC BUS Controller Module Signal Timing</td>
<td>24-36</td>
</tr>
<tr>
<td>24-15</td>
<td>SD/MMC Interface Transmit/Receive Timing Constants</td>
<td>24-36</td>
</tr>
<tr>
<td>24-16</td>
<td>SPI Interface Transmit/Receive Timing Constants</td>
<td>24-37</td>
</tr>
<tr>
<td>24-17</td>
<td>USB Electrical Specifications</td>
<td>24-37</td>
</tr>
<tr>
<td>24-18</td>
<td>USB Full Speed Output Buffer Electrical Characteristics</td>
<td>24-38</td>
</tr>
<tr>
<td>24-19</td>
<td>USB Low Speed Output Buffer Electrical Characteristics</td>
<td>24-38</td>
</tr>
<tr>
<td>24-20</td>
<td>NAND Flash Interface Timing Constants</td>
<td>21-39</td>
</tr>
</tbody>
</table>
NOTIFICATION OF REVISIONS

ORIGINATOR: Samsung Electronics, SOC Development Group, Ki-Heung, South Korea

PRODUCT NAME: S3C2410X RISC Microcontroller

DOCUMENT NAME: S3C2410X User's Manual, Revision 1.2

DOCUMENT NUMBER: 21.2-S3-C2410X-012003

EFFECTIVE DATE: April 28, 2003

SUMMARY: As a result of additional product testing and evaluation, some specifications published in the S3C2410X User's Manual, Revision 1.1, have been changed. These changes for S3C2410X microcontroller are described in detail in the Revision Descriptions section below.

DIRECTIONS: Please note the changes in your copy (copies) of the S3C2410X User's Manual, Revision 1.1 Or, simply attach the Revision Descriptions of the next page to S3C2410X User's Manual, Revision 1.1

REVISION HISTORY

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>October, 2002</td>
<td>There is no preliminary specification. This is the first edition.</td>
</tr>
<tr>
<td>2</td>
<td>January, 2003</td>
<td>Revision 1.1</td>
</tr>
<tr>
<td>3</td>
<td>April 28, 2003</td>
<td>Revision 1.2</td>
</tr>
</tbody>
</table>
REVISION DESCRIPTIONS

ERRATA from S3C2410X User’s Manual

This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of application, operating systems, or tools.

This document may also contain information that was not previously published.

[ FORMAT ]

<NO.> <CHAPTER> : <PAGE> : <CHANGED_ITEM> : <CHANGED_DATE>

<CHANGED DESCRIPTIONS>

1. PRODUCT OVERVIEW : 1-31 : TABLE 1-4. S3C2410X SPECIAL REGISTERS(SHEET 7 OF 11) : 04/28/2003

Changed Table 1-4 (Sheet 7 of 11).

Table 1-4. S3C2410X Special Registers (Sheet 7 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB Device</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAXP_REG</td>
<td>0x5200018F</td>
<td>0x5200018C</td>
<td></td>
<td></td>
<td>Endpoint Max Packet</td>
</tr>
</tbody>
</table>
2. MEMORY CONTROLLER : 5-5 : EXTERNAL nWAIT TIMING DIAGRAM : 04/28/2003

Changed Figure 5-2.

![External nWAIT Timing Diagram](image)

Figure 5-2. S3C2410X External nWAIT Timing Diagram (Tacc=4)

3. MEMORY CONTROLLER : 5-17 : REFRESH CONTROL REGISTER : 04/28/2003

Changed descriptions of REFRESH[19:18].

<table>
<thead>
<tr>
<th>REFRESH</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
</table>
| Tsrc    | [19:18] | SDRAM Semi Row Cycle Time  
00 = 4 clocks  
01 = 5 clocks  
10 = 6 clocks  
11 = 7 clocks  
SDRAM's Row Cycle time (Trc) = Tsrc + Trp  
If Trp = 3 clocks & Tsrc = 7 clocks, Trc = 3 + 7 = 10 clocks | 11 |


Changed a sentence as follows (fourth line from the bottom).

Auto boot mode: The boot code is transferred into Steppingstone after reset. After the transfer, the boot code will be executed on the Steppingstone.

Changed a sentence as follows.

3. \[ 0.7 < \frac{3.373}{\sqrt{m}} < 1.8 \]
5. \[ 20e6 \leq F_{out} \leq 300e6 \] (The max. F_{out} of the PLL itself is 300Mhz)


Changed a sentence as follows. (third line from the top)
The UART can support bit rates of up to 230.4K bps using system clock.

7. UART : 11-7 : UART BAUD-RATE GENERATOR ERROR TOLERANCE : 04/28/2003

Added a section at the middle of this page as follows.

**UART baud-rate generator error tolerance**

UART 10-bit time error should be less than 1.87% (3/160).

\[
\begin{align*}
t_{UPCLK} &= \frac{(UBRDIVn + 1) \times 16 \times 10}{PCLK} & t_{UPCLK} : \text{Real UART 10-bit time} \\
t_{UEXACT} &= \frac{10}{\text{baud-rate}} & t_{UEXACT} : \text{Ideal UART 10-bit time} \\
\text{UART error} &= \frac{(t_{UPCLK} - t_{UEXACT})}{t_{UEXACT}} \times 100\%
\end{align*}
\]


Changed a descriptions of UERSTATn[3] and UERSTATn[1].

<table>
<thead>
<tr>
<th>UERSTATn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
</table>
| Reserved | [3] | 0 = No frame error during receive  
1 = Frame error (Interrupt is requested.) | 0 |
| Reserved | [1] | 0 = No frame error during receive  
1 = Frame error (Interrupt is requested.) | 0 |


Changed a sentence as follows
There are two UART modem status registers including UMSTAT0 and UMSTAT1 in the UART block.

Changed an address of MAXP_REG.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXP_REG</td>
<td>0x5200018C(L) 0x5200018F(B)</td>
<td>R/W( byte)</td>
<td>End Point MAX packet register</td>
<td>0x01</td>
</tr>
</tbody>
</table>


Changed descriptions of LCDCON3[25:19].

<table>
<thead>
<tr>
<th>LCDCON3</th>
<th>Bit: [25:19]</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBPD (TFT)</td>
<td></td>
<td><strong>TFT</strong>: Horizontal back porch is the number of VCLK periods between the falling edge of HSYNC and the start of active data.</td>
<td>0000000</td>
</tr>
<tr>
<td>WDLY (STN)</td>
<td></td>
<td><strong>STN</strong>: WDLY[1:0] bits determine the delay between VLINE and VCLK by counting the number of the HCLK. WDLY[7:2] are reserved. 00 = 16 HCLK, 01 = 32 HCLK, 10 = 48 HCLK, 11 = 64 HCLK</td>
<td></td>
</tr>
</tbody>
</table>


Changed descriptions of LCDCON4[7:0].

<table>
<thead>
<tr>
<th>LCDCON4</th>
<th>Bit: [7:0]</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSPW(TFT)</td>
<td>[7:0]</td>
<td><strong>TFT</strong>: Horizontal sync pulse width determines the HSYNC pulse's high level width by counting the number of the VCLK.</td>
<td>0X00</td>
</tr>
<tr>
<td>WLH(STN)</td>
<td>[7:2]</td>
<td><strong>STN</strong>: WLH[1:0] bits determine the VLINE pulse's high level width by counting the number of the HCLK. WLH[7:2] are reserved. 00 = 16 HCLK, 01 = 32 HCLK, 10 = 48 HCLK, 11 = 64 HCLK</td>
<td></td>
</tr>
</tbody>
</table>


Changed a sentence as follows.(fifth line from the top)

Valid prescaler values range from 0 to 2^8-1.

Changed a descriptions of WTCON[15:8].

<table>
<thead>
<tr>
<th>WTCN</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescaler Value</td>
<td>[15:8]</td>
<td>Prescaler value. The valid range is from 0 to (2^{8}-1).</td>
<td>0x80</td>
</tr>
</tbody>
</table>


Changed Table 24-5.

<p>| Table 24-5. S3C2410X Power Supply Voltage and Current |
|---------------------------------|-----------------|-------------------|-----------------|</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical slow mode power (^{NOTE(3)}) (Total (V_{DDI} + V_{DDIO}))</td>
<td>33</td>
<td>mW</td>
<td>FCLK = 12MHz ((F:H:P = 1:1:1))</td>
</tr>
<tr>
<td>Maximum Power_OFF mode power</td>
<td>80</td>
<td>uA</td>
<td>@1.95V/3.6V, Room temperature All other I/O static.</td>
</tr>
<tr>
<td>Typical Power_OFF mode power (^{NOTE(3)})</td>
<td>8</td>
<td>uA</td>
<td></td>
</tr>
<tr>
<td>Maximum RTC power</td>
<td>63</td>
<td>uA</td>
<td>@1.95V/3.6V, Room temperature X-tal = 32.768KHz for RTC</td>
</tr>
<tr>
<td>Typical RTC power (^{NOTE(3)})</td>
<td>3</td>
<td>uA</td>
<td></td>
</tr>
</tbody>
</table>

Changed Figure 24-36 and Figure 24-37.

Figure 24-36. NAND Flash Address/Command Timing

Figure 24-37. NAND Flash Timing
Table 24-20. NAND Flash Interface Timing Constants

\( (V_{DDi}, V_{DDalive}, V_{DDiarm} = 1.8 \, V \pm 0.15 \, V, \, T_A = 0 \, \text{to} \, 70 \, ^\circ \text{C}, \, V_{DDIO} = 3.3V \pm 0.3V) \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFCON Chip Enable delay</td>
<td>( t_{CED} )</td>
<td>–</td>
<td>6.2</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON CLE delay</td>
<td>( t_{CLED} )</td>
<td>–</td>
<td>7.1</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON ALE delay</td>
<td>( t_{ALED} )</td>
<td>–</td>
<td>7.5</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Write Enable delay</td>
<td>( t_{WED} )</td>
<td>–</td>
<td>7.2</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Read Enable delay</td>
<td>( t_{RED} )</td>
<td>–</td>
<td>7.1</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Write Data Setup time</td>
<td>( t_{WDS} )</td>
<td>–</td>
<td>6.5</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Write Data Hold time</td>
<td>( t_{WDH} )</td>
<td>1.7</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Read Data Setup requirement time</td>
<td>( t_{RDS} )</td>
<td>0.3</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Read Data Hold requirement time</td>
<td>( t_{RDH} )</td>
<td>0.3</td>
<td>–</td>
<td>ns</td>
</tr>
</tbody>
</table>
1

PRODUCT OVERVIEW

INTRODUCTION

This manual describes SAMSUNG's S3C2410X 16/32-bit RISC microprocessor. This product is designed to provide hand-held devices and general applications with cost-effective, low-power, and high-performance microcontroller solution in small die size. To reduce total system cost, the S3C2410X includes the following components separate 16KB Instruction and 16KB Data Cache, MMU to handle virtual memory management, LCD Controller (STN & TFT), NAND Flash Boot Loader, System Manager (chip select logic and SDRAM Controller), 3-ch UART, 4-ch DMA, 4-ch Timers with PWM, I/O Ports, RTC, 8-ch 10-bit ADC and Touch Screen Interface, IIC-BUS Interface, IIS-BUS Interface, USB Host, USB Device, SD Host & Multi-Media Card Interface, 2-ch SPI and PLL for clock generation.

The S3C2410X was developed using an ARM920T core, 0.18um CMOS standard cells and a memory compiler. Its low-power, simple, elegant and fully static design is particularly suitable for cost- and power-sensitive applications. It adopts a new bus architecture called Advanced Microcontroller Bus Architecture (AMBA).

The S3C2410X offers outstanding features with its CPU core, a 16/32-bit ARM920T RISC processor designed by Advanced RISC Machines, Ltd. The ARM920T implements MMU, AMBA BUS, and Harvard cache architecture with separate 16KB instruction and 16KB data caches, each with an 8-word line length.

By providing a complete set of common system peripherals, the S3C2410X minimizes overall system costs and eliminates the need to configure additional components. The integrated on-chip functions that are described in this document include:

- 1.8V int., 3.3V memory, 3.3V external I/O microprocessor with 16KB I-Cache/16KB D-Cache/MMU
- External memory controller (SDRAM Control and Chip Select logic)
- LCD controller (up to 4K color STN and 256K color TFT) with 1-ch LCD-dedicated DMA
- 4-ch DMAs with external request pins
- 3-ch UART (IrDA1.0, 16-Byte Tx FIFO, and 16-Byte Rx FIFO) / 2-ch SPI
- 1-ch multi-master IIC-BUS/1-ch IIS-BUS controller
- SD Host interface version 1.0 & Multi-Media Card Protocol version 2.11 compatible
- 2-port USB Host /1- port USB Device (ver 1.1)
- 4-ch PWM timers & 1-ch internal timer
- Watch Dog Timer
- 117-bit general purpose I/O ports / 24-ch external interrupt source
- Power control: Normal, Slow, Idle and Power-off mode
- 8-ch 10-bit ADC and Touch screen interface
- RTC with calendar function
- On-chip clock generator with PLL

SAMSUNG

ELECTRONICS

1-1
FEATURES

Architecture
- Integrated system for hand-held devices and general embedded applications
- 16/32-Bit RISC architecture and powerful instruction set with ARM920T CPU core
- Enhanced ARM architecture MMU to support WinCE, EPOC 32 and Linux
- Instruction cache, data cache, write buffer and Physical address TAG RAM to reduce the effect of main memory bandwidth and latency on performance
- ARM920T CPU core supports the ARM debug architecture.
- Internal Advanced Microcontroller Bus Architecture (AMBA) (AMBA2.0, AHB/APB)

System Manager
- Little/Big Endian support
- Address space: 128M bytes for each bank (total 1G bytes)
- Supports programmable 8/16/32-bit data bus width for each bank
- Fixed bank start address from bank 0 to bank 6
- Programmable bank start address and bank size for bank 7
- Eight memory banks:
  - Six memory banks for ROM, SRAM, and others.
  - Two memory banks for ROM/SRAM/ Synchronous DRAM
- Fully Programmable access cycles for all memory banks
- Supports external wait signals to expend the bus cycle
- Supports self-refresh mode in SDRAM for power-down
- Supports various types of ROM for booting (NOR/NAND Flash, EEPROM, and others)

NAND Flash Boot Loader
- Supports booting from NAND flash memory
- 4KB internal buffer for booting
- Supports storage memory for NAND flash memory after booting

Cache Memory
- 64-way set-associative cache with I-Cache (16KB) and D-Cache (16KB)
- 8words length per line with one valid bit and two dirty bits per line
- Pseudo random or round robin replacement algorithm
- Write-through or write-back cache operation to update the main memory
- The write buffer can hold 16 words of data and four addresses.

Clock & Power Manager
- On-chip MPLL and UPLL: UPLL generates the clock to operate USB Host/Device. MPLL generates the clock to operate MCU at maximum 203Mhz @ 1.8V.
- Clock can be fed selectively to each function block by software.
- Power mode: Normal, Slow, Idle, and Power-off mode
  Normal mode: Normal operating mode
  Slow mode: Low frequency clock without PLL
  Idle mode: The clock for only CPU is stopped.
  Power-off mode: The Core power including all peripherals is shut down.
- Woken up by EINT[15:0] or RTC alarm interrupt from Power-Off mode
FEATURES (Continued)

Interrupt Controller
- 55 Interrupt sources
  (One Watch dog timer, 5 timers, 9 UARTs, 24 external interrupts, 4 DMA, 2 RTC, 2 ADC, 1 IIC, 2 SPI, 1 SDI, 2 USB, 1 LCD, and 1 Battery Fault)
- Level/Edge mode on external interrupt source
- Programmable polarity of edge and level
- Supports Fast Interrupt request (FIQ) for very urgent interrupt request

Timer with Pulse Width Modulation (PWM)
- 4-ch 16-bit Timer with PWM / 1-ch 16-bit internal timer with DMA-based or interrupt-based operation
- Programmable duty cycle, frequency, and polarity
- Dead-zone generation
- Supports external clock sources

RTC (Real Time Clock)
- Full clock feature: second, minute, hour, date, day, month, and year
- 32.768 KHz operation
- Alarm interrupt
- Time tick interrupt

General Purpose Input/Output Ports
- 24 external interrupt ports
- multiplexed input/output ports

UART
- 3-channel UART with DMA-based or interrupt-based operation
- Supports 5-bit, 6-bit, 7-bit, or 8-bit serial data transmit/receive (Tx/Rx)
- Supports external clocks for the UART operation (UCLK)
- Programmable baud rate
- Supports IrDA 1.0
- Loopback mode for testing
- Each channel has internal 16-byte Tx FIFO and 16-byte Rx FIFO.

DMA Controller
- 4-ch DMA controller
- Supports memory to memory, IO to memory, memory to IO, and IO to IO transfers
- Burst transfer mode to enhance the transfer rate

A/D Converter & Touch Screen Interface
- 8-ch multiplexed ADC
- Max. 500KSPS and 10-bit Resolution

LCD Controller STN LCD Displays Feature
- Supports 3 types of STN LCD panels: 4-bit dual scan, 4-bit single scan, 8-bit single scan display type
- Supports monochrome mode, 4 gray levels, 16 gray levels, 256 colors and 4096 colors for STN LCD
- Supports multiple screen size
  - Typical actual screen size: 640x480, 320x240, 160x160, and others
  - Maximum virtual screen size is 4 Mbytes.
  - Maximum virtual screen size in 256 color mode: 4096x1024, 2048x2048, 1024x4096, and others

TFT(Thin Film Transistor) Color Displays Feature
- Supports 1, 2, 4 or 8 bpp (bit-per-pixel) palette color displays for color TFT
- Supports 16 bpp non-palette true-color displays for color TFT
- Supports maximum 16M color TFT at 24 bpp mode
- Supports multiple screen size
  - Typical actual screen size: 640x480, 320x240, 160x160, and others
  - Maximum virtual screen size is 4Mbytes.
  - Maximum virtual screen size in 64K color mode: 2048x1024, and others
FEATURES (Continued)

Watchdog Timer
- 16-bit Watchdog Timer
- Interrupt request or system reset at time-out

IIC-Bus Interface
- 1-ch Multi-Master IIC-Bus
- Serial, 8-bit oriented and bi-directional data transfers can be made at up to 100 Kbit/s in Standard mode or up to 400 Kbit/s in Fast mode.

IIS-Bus Interface
- 1-ch IIS-bus for audio interface with DMA-based operation
- Serial, 8-/16-bit per channel data transfers
- 128 Bytes (64-Byte + 64-Byte) FIFO for Tx/Rx
- Supports IIS format and MSB-justified data format

USB Host
- 2-port USB Host
- Complies with OHCI Rev. 1.0
- Compatible with USB Specification version 1.1

USB Device
- 1-port USB Device
- 5 Endpoints for USB Device
- Compatible with USB Specification version 1.1

SD Host Interface
- Compatible with SD Memory Card Protocol version 1.0
- Compatible with SDIO Card Protocol version 1.0
- Bytes FIFO for Tx/Rx
- DMA based or Interrupt based operation
- Compatible with Multimedia Card Protocol version 2.11

SPI Interface
- Compatible with 2-ch Serial Peripheral Interface Protocol version 2.11
- 2x8 bits Shift register for Tx/Rx
- DMA-based or interrupt-based operation

Operating Voltage Range
- Core: 1.8V
- Memory : 3.3V SDRAM or VDD = 1.8/2.5V & VDDQ = 3.0/3.3V Mobile SDRAM
- I/O : 3.3V

Operating Frequency
- Up to 203MHz

Package
- 272-FBGA
BLOCK DIAGRAM

Figure 1-1. S3C2410X Block Diagram
PIN ASSIGNMENTS

Figure 1-2. S3C2410X Pin Assignments (272-FBGA)
Table 1-1. 272-Pin FBGA Pin Assignments – Pin Number Order (Sheet 1 of 3)

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Pin Number</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>DATA19</td>
<td>B14</td>
<td>ADDR0/GPA0</td>
<td>D10</td>
<td>ADDR19/GPA4</td>
</tr>
<tr>
<td>A2</td>
<td>DATA18</td>
<td>B15</td>
<td>nSRAS</td>
<td>D11</td>
<td>VDDi</td>
</tr>
<tr>
<td>A3</td>
<td>DATA16</td>
<td>B16</td>
<td>nBE1:nWBE1:DQM1</td>
<td>D12</td>
<td>ADDR10</td>
</tr>
<tr>
<td>A4</td>
<td>DATA15</td>
<td>B17</td>
<td>VSSI</td>
<td>D13</td>
<td>ADDR5</td>
</tr>
<tr>
<td>A5</td>
<td>DATA11</td>
<td>C1</td>
<td>DATA24</td>
<td>D14</td>
<td>ADDR1</td>
</tr>
<tr>
<td>A6</td>
<td>VDDMOP</td>
<td>C2</td>
<td>DATA23</td>
<td>D15</td>
<td>VSSMOP</td>
</tr>
<tr>
<td>A7</td>
<td>DATA6</td>
<td>C3</td>
<td>DATA21</td>
<td>D16</td>
<td>SCKE</td>
</tr>
<tr>
<td>A8</td>
<td>DATA1</td>
<td>C4</td>
<td>VDDi</td>
<td>D17</td>
<td>nGCS0</td>
</tr>
<tr>
<td>A9</td>
<td>ADDR21/GPA6</td>
<td>C5</td>
<td>DATA12</td>
<td>E1</td>
<td>DATA31</td>
</tr>
<tr>
<td>A10</td>
<td>ADDR16/GPA1</td>
<td>C6</td>
<td>DATA7</td>
<td>E2</td>
<td>DATA29</td>
</tr>
<tr>
<td>A11</td>
<td>ADDR13</td>
<td>C7</td>
<td>DATA4</td>
<td>E3</td>
<td>DATA28</td>
</tr>
<tr>
<td>A12</td>
<td>VDDMOP</td>
<td>C8</td>
<td>VDDi</td>
<td>E4</td>
<td>DATA30</td>
</tr>
<tr>
<td>A13</td>
<td>ADDR6</td>
<td>C9</td>
<td>ADDR25/GPA10</td>
<td>E5</td>
<td>VDDMOP</td>
</tr>
<tr>
<td>A14</td>
<td>ADDR2</td>
<td>C10</td>
<td>VSSMOP</td>
<td>E6</td>
<td>VSSMOP</td>
</tr>
<tr>
<td>A15</td>
<td>VDDMOP</td>
<td>C11</td>
<td>ADDR14</td>
<td>E7</td>
<td>DATA3</td>
</tr>
<tr>
<td>A16</td>
<td>nBE3:nWBE3:DQM3</td>
<td>C12</td>
<td>ADDR7</td>
<td>E8</td>
<td>ADDR26/GPA11</td>
</tr>
<tr>
<td>A17</td>
<td>nBE0:nWBE0:DQM0</td>
<td>C13</td>
<td>ADDR3</td>
<td>E9</td>
<td>ADDR23/GPA8</td>
</tr>
<tr>
<td>B1</td>
<td>DATA22</td>
<td>C14</td>
<td>nSCAS</td>
<td>E10</td>
<td>ADDR18/GPA3</td>
</tr>
<tr>
<td>B2</td>
<td>DATA20</td>
<td>C15</td>
<td>nBE2:nWBE2:DQM2</td>
<td>E11</td>
<td>VDDMOP</td>
</tr>
<tr>
<td>B3</td>
<td>DATA17</td>
<td>C16</td>
<td>nOE</td>
<td>E12</td>
<td>ADDR11</td>
</tr>
<tr>
<td>B4</td>
<td>VDDMOP</td>
<td>C17</td>
<td>VDDi</td>
<td>E13</td>
<td>nWE</td>
</tr>
<tr>
<td>B5</td>
<td>DATA13</td>
<td>D1</td>
<td>DATA27</td>
<td>E14</td>
<td>nGCS3/GPA14</td>
</tr>
<tr>
<td>B6</td>
<td>DATA9</td>
<td>D2</td>
<td>DATA25</td>
<td>E15</td>
<td>nGCS1/GPA12</td>
</tr>
<tr>
<td>B7</td>
<td>DATA5</td>
<td>D3</td>
<td>VSSMOP</td>
<td>E16</td>
<td>nGCS2/GPA13</td>
</tr>
<tr>
<td>B8</td>
<td>DATA0</td>
<td>D4</td>
<td>DATA26</td>
<td>E17</td>
<td>nGCS4/GPA15</td>
</tr>
<tr>
<td>B9</td>
<td>ADDR24/GPA9</td>
<td>D5</td>
<td>DATA14</td>
<td>F1</td>
<td>TOUT1/GPB1</td>
</tr>
<tr>
<td>B10</td>
<td>ADDR17/GPA2</td>
<td>D6</td>
<td>DATA10</td>
<td>F2</td>
<td>TOUT0/GPB0</td>
</tr>
<tr>
<td>B11</td>
<td>ADDR12</td>
<td>D7</td>
<td>DATA2</td>
<td>F3</td>
<td>VSSMOP</td>
</tr>
<tr>
<td>B12</td>
<td>ADDR8</td>
<td>D8</td>
<td>VDDMOP</td>
<td>F4</td>
<td>TOUT2/GPB2</td>
</tr>
<tr>
<td>B13</td>
<td>ADDR4</td>
<td>D9</td>
<td>ADDR22/GPA7</td>
<td>F5</td>
<td>VSSOP</td>
</tr>
<tr>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Pin Number</td>
<td>Pin Name</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>F6</td>
<td>VSSi</td>
<td>H4</td>
<td>nXDREQ1/GPB8</td>
<td>K13</td>
<td>TXD2/nRTS1/GPH6</td>
</tr>
<tr>
<td>F7</td>
<td>DATA8</td>
<td>H5</td>
<td>nTRST</td>
<td>K14</td>
<td>RXD1/GPH5</td>
</tr>
<tr>
<td>F8</td>
<td>VSSMOP</td>
<td>H6</td>
<td>TCK</td>
<td>K15</td>
<td>TXD0/GPH2</td>
</tr>
<tr>
<td>F9</td>
<td>VSSi</td>
<td>H12</td>
<td>CLE/GPA17</td>
<td>K16</td>
<td>TXD1/GPH4</td>
</tr>
<tr>
<td>F10</td>
<td>ADDR20/GPA5</td>
<td>H13</td>
<td>VSSOP</td>
<td>K17</td>
<td>RXD0/GPH3</td>
</tr>
<tr>
<td>F11</td>
<td>VSSi</td>
<td>H14</td>
<td>VDDMOP</td>
<td>L1</td>
<td>VD0/GPC8</td>
</tr>
<tr>
<td>F12</td>
<td>VSSMOP</td>
<td>H15</td>
<td>VSSi</td>
<td>L2</td>
<td>VD1/GPC9</td>
</tr>
<tr>
<td>F13</td>
<td>SCLK0</td>
<td>H16</td>
<td>XTopll</td>
<td>L3</td>
<td>LCDVF2/GPC7</td>
</tr>
<tr>
<td>F14</td>
<td>SCLK1</td>
<td>H17</td>
<td>XTipll</td>
<td>L4</td>
<td>LCDVF1/GPC6</td>
</tr>
<tr>
<td>F15</td>
<td>nGCS5/GPA16</td>
<td>J1</td>
<td>TDI</td>
<td>L5</td>
<td>VDDiarm</td>
</tr>
<tr>
<td>F16</td>
<td>nGCS6:nSCS5</td>
<td>J2</td>
<td>VCLK:LCD_HCLK/GPC1</td>
<td>L6</td>
<td>LCDVF1/GPC6</td>
</tr>
<tr>
<td>F17</td>
<td>nGCS7:nSCS1</td>
<td>J3</td>
<td>TMS</td>
<td>L7</td>
<td>IICSL/GPE14</td>
</tr>
<tr>
<td>G1</td>
<td>nXBACK/GPB5</td>
<td>J4</td>
<td>LEND:STH/GPC0</td>
<td>L9</td>
<td>EINT11/nSS1/GPG3</td>
</tr>
<tr>
<td>G2</td>
<td>nXDACK1/GPB7</td>
<td>J5</td>
<td>TDO</td>
<td>L11</td>
<td>VDDi_UPLL</td>
</tr>
<tr>
<td>G3</td>
<td>TOUT3/GPB3</td>
<td>J6</td>
<td>VLINE:HSYNC:CPV/GPC2</td>
<td>L12</td>
<td>nRTS0/GPH1</td>
</tr>
<tr>
<td>G4</td>
<td>TCLK0/GPB4</td>
<td>J7</td>
<td>VSSiarm</td>
<td>L13</td>
<td>UPLL_CAP</td>
</tr>
<tr>
<td>G5</td>
<td>nXBREQ/GPB6</td>
<td>J11</td>
<td>EXTCLK</td>
<td>L14</td>
<td>nCTS0/GPH0</td>
</tr>
<tr>
<td>G6</td>
<td>VDDalive</td>
<td>J12</td>
<td>nRESET</td>
<td>L15</td>
<td>EINT6/GPF6</td>
</tr>
<tr>
<td>G7</td>
<td>VDDiarm</td>
<td>J13</td>
<td>VDDi</td>
<td>L16</td>
<td>UCLK/GPH8</td>
</tr>
<tr>
<td>G9</td>
<td>VSSMOP</td>
<td>J14</td>
<td>VDDalive</td>
<td>L17</td>
<td>EINT7/GPF7</td>
</tr>
<tr>
<td>G11</td>
<td>ADDR15</td>
<td>J15</td>
<td>PWREN</td>
<td>M1</td>
<td>VSSIarm</td>
</tr>
<tr>
<td>G12</td>
<td>ADDR9</td>
<td>J16</td>
<td>nRSTOUT/GPA21</td>
<td>M2</td>
<td>VD5/GPC13</td>
</tr>
<tr>
<td>G13</td>
<td>nWAIT</td>
<td>J17</td>
<td>nBATT_FLT</td>
<td>M3</td>
<td>VD3/GPC11</td>
</tr>
<tr>
<td>G14</td>
<td>ALE/GPA18</td>
<td>K1</td>
<td>VDDOP</td>
<td>M4</td>
<td>VD4/GPC12</td>
</tr>
<tr>
<td>G15</td>
<td>nFWE/GPA19</td>
<td>K2</td>
<td>VM:VDEN:TP/GPC4</td>
<td>M5</td>
<td>VSSIarm</td>
</tr>
<tr>
<td>G16</td>
<td>nFRE/GPA20</td>
<td>K3</td>
<td>VDDiarm</td>
<td>M6</td>
<td>VDDOP</td>
</tr>
<tr>
<td>G17</td>
<td>nFCE/GPA22</td>
<td>K4</td>
<td>VFRAME:VSYNC:STV/GPC3</td>
<td>M7</td>
<td>VDDiarm</td>
</tr>
<tr>
<td>H1</td>
<td>VSSIarm</td>
<td>K5</td>
<td>VSSOP</td>
<td>M8</td>
<td>IICSDA/GPE15</td>
</tr>
<tr>
<td>H2</td>
<td>nXDACK0/GPB9</td>
<td>K6</td>
<td>LCDVF0/GPC5</td>
<td>M9</td>
<td>VSSIarm</td>
</tr>
<tr>
<td>H3</td>
<td>nXDREQ0/GPB10</td>
<td>K12</td>
<td>RXD2/nCTS1/GPH7</td>
<td>M10</td>
<td>DP1/PDP0</td>
</tr>
<tr>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Pin Number</td>
<td>Pin Name</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>M11</td>
<td>EINT23/nYPON/GPG15</td>
<td>P8</td>
<td>SPICLK0/GPE13</td>
<td>T5</td>
<td>I2SLRCK/GPE0</td>
</tr>
<tr>
<td>M12</td>
<td>RTCVDD</td>
<td>P9</td>
<td>EINT12/LCD_PWREN/GPG4</td>
<td>T6</td>
<td>SDCLK/GPE5</td>
</tr>
<tr>
<td>M13</td>
<td>VSSi_MPLL</td>
<td>P10</td>
<td>EINT18/GPG10</td>
<td>T7</td>
<td>SPIMISO0/GPE11</td>
</tr>
<tr>
<td>M14</td>
<td>EINT5/GPF5</td>
<td>P11</td>
<td>EINT20/XMON/GPG12</td>
<td>T8</td>
<td>EINT10/nSS0/GPG2</td>
</tr>
<tr>
<td>M15</td>
<td>EINT4/GPF4</td>
<td>P12</td>
<td>VSSOP</td>
<td>T9</td>
<td>VSSOP</td>
</tr>
<tr>
<td>M16</td>
<td>EINT2/GPF2</td>
<td>P13</td>
<td>DP0</td>
<td>T10</td>
<td>EINT17/GPG9</td>
</tr>
<tr>
<td>M17</td>
<td>EINT3/GPF3</td>
<td>P14</td>
<td>VDDi_MPLL</td>
<td>T11</td>
<td>EINT22/YMON/GPG14</td>
</tr>
<tr>
<td>N1</td>
<td>VD6/GPC14</td>
<td>P15</td>
<td>VDDA_ADC</td>
<td>T12</td>
<td>DN0</td>
</tr>
<tr>
<td>N2</td>
<td>VD8/GPD0</td>
<td>P16</td>
<td>XTIrtc</td>
<td>T13</td>
<td>OM3</td>
</tr>
<tr>
<td>N3</td>
<td>VD7/GPC15</td>
<td>P17</td>
<td>MPLLVCAP</td>
<td>T14</td>
<td>VSSA_ADC</td>
</tr>
<tr>
<td>N4</td>
<td>VD9/GPD1</td>
<td>R1</td>
<td>VDDiarm</td>
<td>T15</td>
<td>AIN1</td>
</tr>
<tr>
<td>N5</td>
<td>VDDiarm</td>
<td>R2</td>
<td>VD14/GPD6</td>
<td>T16</td>
<td>AIN3</td>
</tr>
<tr>
<td>N6</td>
<td>CDCLK/GPE2</td>
<td>R3</td>
<td>VD17/GPD9</td>
<td>T17</td>
<td>AIN5</td>
</tr>
<tr>
<td>N7</td>
<td>SDDAT1/GPE8</td>
<td>R4</td>
<td>VD18/GPD10</td>
<td>U1</td>
<td>VD15/GPD7</td>
</tr>
<tr>
<td>N8</td>
<td>VSSIarm</td>
<td>R5</td>
<td>VSSOP</td>
<td>U2</td>
<td>VD19/GPD11</td>
</tr>
<tr>
<td>N9</td>
<td>VDDiarm</td>
<td>R6</td>
<td>SDDAT0/GPD7</td>
<td>U3</td>
<td>VD21/GPD13</td>
</tr>
<tr>
<td>N10</td>
<td>VDDiarm</td>
<td>R7</td>
<td>SDDAT3/GPE10</td>
<td>U4</td>
<td>VSSIarm</td>
</tr>
<tr>
<td>N11</td>
<td>DN1/PDN0</td>
<td>R8</td>
<td>EINT8/GPG0</td>
<td>U5</td>
<td>I2SSDI/nSS0/GPE3</td>
</tr>
<tr>
<td>N12</td>
<td>Vref</td>
<td>R9</td>
<td>EINT14/SPIMOSI1/GPG6</td>
<td>U6</td>
<td>I2SSDO/I2SSDI/GPE4</td>
</tr>
<tr>
<td>N13</td>
<td>AIN7</td>
<td>R10</td>
<td>EINT15/SPICLK1/GPG7</td>
<td>U7</td>
<td>SPIMOSI0/GPE12</td>
</tr>
<tr>
<td>N14</td>
<td>EINT0/GPF0</td>
<td>R11</td>
<td>EINT19/TCLK1/GPG11</td>
<td>U8</td>
<td>EINT9/GPG1</td>
</tr>
<tr>
<td>N15</td>
<td>VSSi_UPLL</td>
<td>R12</td>
<td>CLKOUT0/GPH9</td>
<td>U9</td>
<td>EINT13/SPIMISO1/GPG5</td>
</tr>
<tr>
<td>N16</td>
<td>VDDO</td>
<td>R13</td>
<td>R/nB</td>
<td>U10</td>
<td>EINT16/GPG8</td>
</tr>
<tr>
<td>N17</td>
<td>EINT1/GPF1</td>
<td>R14</td>
<td>OM0</td>
<td>U11</td>
<td>EINT21/nXPON/GPG13</td>
</tr>
<tr>
<td>P10</td>
<td>VD10/GPD2</td>
<td>R15</td>
<td>AIN4</td>
<td>U12</td>
<td>CLKOUT1/GPH10</td>
</tr>
<tr>
<td>P1</td>
<td>VD12/GPD4</td>
<td>R16</td>
<td>AIN6</td>
<td>U13</td>
<td>NCON</td>
</tr>
<tr>
<td>P3</td>
<td>VD11/GPD3</td>
<td>R17</td>
<td>XTOrtc</td>
<td>U14</td>
<td>OM2</td>
</tr>
<tr>
<td>P4</td>
<td>VD23/nSS0/GPD15</td>
<td>T1</td>
<td>VD13/GPD5</td>
<td>U15</td>
<td>OM1</td>
</tr>
<tr>
<td>P5</td>
<td>I2SSCLK/GPE1</td>
<td>T2</td>
<td>VD16/GPD8</td>
<td>U16</td>
<td>AIN0</td>
</tr>
<tr>
<td>P6</td>
<td>SDCMD/GPE6</td>
<td>T3</td>
<td>VD20/GPD12</td>
<td>U17</td>
<td>AIN2</td>
</tr>
<tr>
<td>P7</td>
<td>SDDAT2/GPE9</td>
<td>T4</td>
<td>VD22/nSS1/GPD14</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Table 1-2. 272-Pin FBGA Pin Assignments (Sheet 1 of 9)

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Default Function</th>
<th>I/O State @BUS REQ</th>
<th>I/O State @PWR-off</th>
<th>I/O State @nRESET</th>
<th>I/O Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
<td>DATA21</td>
<td>DATA21</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>B1</td>
<td>DATA22</td>
<td>DATA22</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>C2</td>
<td>DATA23</td>
<td>DATA23</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>D3</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>E5</td>
<td>VDDMOP</td>
<td>VDDMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>C1</td>
<td>DATA24</td>
<td>DATA24</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>D2</td>
<td>DATA25</td>
<td>DATA25</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>D4</td>
<td>DATA26</td>
<td>DATA26</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>D1</td>
<td>DATA27</td>
<td>DATA27</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>E3</td>
<td>DATA28</td>
<td>DATA28</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>E2</td>
<td>DATA29</td>
<td>DATA29</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>E4</td>
<td>DATA30</td>
<td>DATA30</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>E1</td>
<td>DATA31</td>
<td>DATA31</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>I</td>
<td>t12</td>
</tr>
<tr>
<td>F3</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>F5</td>
<td>VSSOP</td>
<td>VSSOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>F2</td>
<td>TOUT0/GPB0</td>
<td>GPB0</td>
<td>–/–</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>F1</td>
<td>TOUT1/GPB1</td>
<td>GPB1</td>
<td>–/–</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>F4</td>
<td>TOUT2/GPB2</td>
<td>GPB2</td>
<td>–/–</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>G3</td>
<td>TOUT3/GPB3</td>
<td>GPB3</td>
<td>–/–</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>G4</td>
<td>TCLK0/GPB4</td>
<td>GPB4</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>G1</td>
<td>nXBACK/GPB5</td>
<td>GPB5</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>G5</td>
<td>nXBREQ/GPB6</td>
<td>GPB6</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>G2</td>
<td>nXDACK1/GPB7</td>
<td>GPB7</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>G6</td>
<td>VDDalive</td>
<td>VDDalive</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1i</td>
</tr>
<tr>
<td>G7</td>
<td>VDDiarm</td>
<td>VDDiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>H1</td>
<td>VSSiarm</td>
<td>VSSiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>H4</td>
<td>nXDREQ1/GPB8</td>
<td>GPB8</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>H2</td>
<td>nXDACK0/GPB9</td>
<td>GPB9</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>H3</td>
<td>nXDREQ0/GPB10</td>
<td>GPB10</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>H5</td>
<td>nTRST</td>
<td>nTRST</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>is</td>
</tr>
<tr>
<td>H6</td>
<td>TCK</td>
<td>TCK</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>is</td>
</tr>
<tr>
<td>J1</td>
<td>TDI</td>
<td>TDI</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>is</td>
</tr>
<tr>
<td>J3</td>
<td>TMS</td>
<td>TMS</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>is</td>
</tr>
<tr>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Default Function</td>
<td>I/O State @BUS REQ</td>
<td>I/O State @PWR-off</td>
<td>I/O State @nRESET</td>
<td>I/O Type</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>J5</td>
<td>TDO</td>
<td>TDO</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>ot</td>
</tr>
<tr>
<td>J4</td>
<td>LEND:STH/GPC0</td>
<td>GPC0</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>J2</td>
<td>VCLK:LCD_HCLK/GPC1</td>
<td>GPC1</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>J6</td>
<td>VLINE:HSYNC:CPV/GPC2</td>
<td>GPC2</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>K3</td>
<td>VDDiarm</td>
<td>VDDiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>J7</td>
<td>VSSiarm</td>
<td>VSSiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>K2</td>
<td>VM:VDEN:TP/GPC4</td>
<td>GPC4</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>K4</td>
<td>VFRAME:VSYNC:STV/GPC3</td>
<td>GPC3</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>K1</td>
<td>VDDiop</td>
<td>VDDiop</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>K5</td>
<td>VSSop</td>
<td>VSSop</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>K6</td>
<td>LCDVF0/GPC5</td>
<td>GPC5</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>L6</td>
<td>LCDVF1/GPC6</td>
<td>GPC6</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>L3</td>
<td>LCDVF2/GPC7</td>
<td>GPC7</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>L1</td>
<td>VD0/GPC8</td>
<td>GPC8</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>L2</td>
<td>VD1/GPC9</td>
<td>GPC9</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>L4</td>
<td>VD2/GPC10</td>
<td>GPC10</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>M3</td>
<td>VD3/GPC11</td>
<td>GPC11</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>L5</td>
<td>VDDiarm</td>
<td>VDDiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>M1</td>
<td>VSSiarm</td>
<td>VSSiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>M4</td>
<td>VD4/GPC12</td>
<td>GPC12</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>M2</td>
<td>VD5/GPC13</td>
<td>GPC13</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>N1</td>
<td>VD6/GPC14</td>
<td>GPC14</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>N3</td>
<td>VD7/GPC15</td>
<td>GPC15</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>N2</td>
<td>VD8/GPD0</td>
<td>GPD0</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>N4</td>
<td>VD9/GPD1</td>
<td>GPD1</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>P1</td>
<td>VD10/GPD2</td>
<td>GPD2</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>P3</td>
<td>VD11/GPD3</td>
<td>GPD3</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>P2</td>
<td>VD12/GPD4</td>
<td>GPD4</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>R1</td>
<td>VDDiarm</td>
<td>VDDiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>M5</td>
<td>VSSiarm</td>
<td>VSSiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>T1</td>
<td>VD13/GPD5</td>
<td>GPD5</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>R2</td>
<td>VD14/GPD6</td>
<td>GPD6</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>U1</td>
<td>VD15/GPD7</td>
<td>GPD7</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>t8</td>
</tr>
</tbody>
</table>
Table 1-2. 272-Pin FBGA Pin Assignments (Sheet 3 of 9)

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Default Function</th>
<th>I/O State @BUS REQ</th>
<th>I/O State @PWR-off</th>
<th>I/O State @nRESET</th>
<th>I/O Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>VD16/GPD8</td>
<td>GPD8</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>R3</td>
<td>VD17/GPD9</td>
<td>GPD9</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>R4</td>
<td>VD18/GPD10</td>
<td>GPD10</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>U2</td>
<td>VD19/GPD11</td>
<td>GPD11</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>T3</td>
<td>VD20/GPD12</td>
<td>GPD12</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>U3</td>
<td>VD21/GPD13</td>
<td>GPD13</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>T4</td>
<td>VD22/nSS1/GPD14</td>
<td>GPD14</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>P4</td>
<td>VD23/nSS0/GPD15</td>
<td>GPD15</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>N5</td>
<td>VDDiarm</td>
<td>VDDiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>U4</td>
<td>VSSiarm</td>
<td>VSSiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>M6</td>
<td>VDDOP</td>
<td>VDDOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>R5</td>
<td>VSSOP</td>
<td>VSSOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>T5</td>
<td>I2SLRCK/GPE0</td>
<td>GPE0</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>P5</td>
<td>I2SSCLK/GPE1</td>
<td>GPE1</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>N6</td>
<td>CDCLK/GPE2</td>
<td>GPE2</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>U5</td>
<td>I2SSDI/nSS0/GPE3</td>
<td>GPE3</td>
<td>--/--</td>
<td>--/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>U6</td>
<td>I2SSDO/I2SSDI/GPE4</td>
<td>GPE4</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>T6</td>
<td>SDCLK/GPE5</td>
<td>GPE5</td>
<td>--/--</td>
<td>O(L)/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>P6</td>
<td>SDCMD/GPE6</td>
<td>GPE6</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>R6</td>
<td>SDDAT0/GPE7</td>
<td>GPE7</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>N7</td>
<td>SDDAT1/GPE8</td>
<td>GPE8</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>P7</td>
<td>SDDAT2/GPE9</td>
<td>GPE9</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>R7</td>
<td>SDDAT3/GPE10</td>
<td>GPE10</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>T7</td>
<td>SPIMISO0/GPE11</td>
<td>GPE11</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>U7</td>
<td>SPIMOSI0/GPE12</td>
<td>GPE12</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>P8</td>
<td>SPICLK0/GPE13</td>
<td>GPE13</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>M7</td>
<td>VDDiarm</td>
<td>VDDiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>N8</td>
<td>VSSiarm</td>
<td>VSSiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>L7</td>
<td>IICSCL/GPE14</td>
<td>GPE14</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>d8</td>
</tr>
<tr>
<td>M8</td>
<td>IICSDA/GPE15</td>
<td>GPE15</td>
<td>--/--</td>
<td>Hi-z/--</td>
<td>I</td>
<td>d8</td>
</tr>
<tr>
<td>R8</td>
<td>EINT8/GPG0</td>
<td>GPG0</td>
<td>--/--</td>
<td>--/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>U8</td>
<td>EINT9/GPG1</td>
<td>GPG1</td>
<td>--/--</td>
<td>--/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>T8</td>
<td>EINT10/nSS0/GPG2</td>
<td>GPG2</td>
<td>--/--</td>
<td>--/--</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Default Function</td>
<td>I/O State @BUS REQ</td>
<td>I/O State @PWR-off</td>
<td>I/O State @nRESET</td>
<td>I/O Type</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>L9</td>
<td>EINT11/nSS1/GPG3</td>
<td>GPG3</td>
<td>–/–/–</td>
<td>–/–/–</td>
<td>–/–/–</td>
<td>I</td>
</tr>
<tr>
<td>P9</td>
<td>EINT12/LCD_PWREN/GPG4</td>
<td>GPG4</td>
<td>–/–/–</td>
<td>–/O(L)/–</td>
<td>–/Hi-z/-</td>
<td>t8</td>
</tr>
<tr>
<td>U9</td>
<td>EINT13/SPIMISO1/GPG5</td>
<td>GPG5</td>
<td>–/–/–</td>
<td>–/Hi-z/-</td>
<td>–/–/–</td>
<td>t8</td>
</tr>
<tr>
<td>R9</td>
<td>EINT14/SPIMOSI1/GPG6</td>
<td>GPG6</td>
<td>–/–/–</td>
<td>–/Hi-z/-</td>
<td>–/–/–</td>
<td>t8</td>
</tr>
<tr>
<td>T9</td>
<td>VSSOP</td>
<td>VSSOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>N9</td>
<td>VDDOP</td>
<td>VDDOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>N10</td>
<td>VDDiarm</td>
<td>VDDiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>M9</td>
<td>VSSiarm</td>
<td>VSSiarm</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>R10</td>
<td>EINT15/SPICLK1/GPG7</td>
<td>GPG7</td>
<td>–/–/–</td>
<td>–/Hi-z/-</td>
<td>–/–/–</td>
<td>t8</td>
</tr>
<tr>
<td>U10</td>
<td>EINT16/GPG8</td>
<td>GPG8</td>
<td>–/–</td>
<td>–/–</td>
<td>–/–/–</td>
<td>t6</td>
</tr>
<tr>
<td>T10</td>
<td>EINT17/GPG9</td>
<td>GPG9</td>
<td>–/–</td>
<td>–/–</td>
<td>–/–/–</td>
<td>t6</td>
</tr>
<tr>
<td>P10</td>
<td>EINT18/GPG10</td>
<td>GPG10</td>
<td>–/–</td>
<td>–/–</td>
<td>–/–/–</td>
<td>t6</td>
</tr>
<tr>
<td>R11</td>
<td>EINT19/TCLK1/GPG11</td>
<td>GPG11</td>
<td>–/–/–</td>
<td>–/–/–</td>
<td>–/–/–</td>
<td>t12</td>
</tr>
<tr>
<td>P11</td>
<td>EINT20/XMON/GPG12</td>
<td>GPG12</td>
<td>–/–/–</td>
<td>–/O(L)/–</td>
<td>–/–/–</td>
<td>t12</td>
</tr>
<tr>
<td>U11</td>
<td>EINT21/nXPON/GPG13</td>
<td>GPG13</td>
<td>–/–/–</td>
<td>–/O(L)/–</td>
<td>–/–/–</td>
<td>t12</td>
</tr>
<tr>
<td>T11</td>
<td>EINT22/YMON/GPG14</td>
<td>GPG14</td>
<td>–/–/–</td>
<td>–/O(L)/–</td>
<td>–/–/–</td>
<td>t12</td>
</tr>
<tr>
<td>M11</td>
<td>EINT23/nYPON/GPG15</td>
<td>GPG15</td>
<td>–/–/–</td>
<td>–/O(L)/–</td>
<td>–/–/–</td>
<td>t12</td>
</tr>
<tr>
<td>R12</td>
<td>CLKOUT0/GPH9</td>
<td>GPH9</td>
<td>–/–</td>
<td>–/O(L)/–</td>
<td>–/–/–</td>
<td>t12</td>
</tr>
<tr>
<td>U12</td>
<td>CLKOUT1/GPH10</td>
<td>GPH10</td>
<td>–/–</td>
<td>–/O(L)/–</td>
<td>–/–/–</td>
<td>t12</td>
</tr>
<tr>
<td>M10</td>
<td>DP1/PDP0</td>
<td>DP1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>N11</td>
<td>DN1/PDN0</td>
<td>DN1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>P13</td>
<td>DP0</td>
<td>DP0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>T12</td>
<td>DN0</td>
<td>DN0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>U13</td>
<td>NCON</td>
<td>NCON</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>R13</td>
<td>R/nB</td>
<td>R/nB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>T13</td>
<td>OM3</td>
<td>OM3</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>U14</td>
<td>OM2</td>
<td>OM2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>U15</td>
<td>OM1</td>
<td>OM1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>R14</td>
<td>OM0</td>
<td>OM0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Al</td>
</tr>
<tr>
<td>P12</td>
<td>VSSOP</td>
<td>VSSOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>T14</td>
<td>VSSA_ADC</td>
<td>VSSA_ADC</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3t</td>
</tr>
<tr>
<td>N12</td>
<td>Vref</td>
<td>Vref</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Ai</td>
</tr>
<tr>
<td>U16</td>
<td>AIN0</td>
<td>AIN0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>r10</td>
</tr>
</tbody>
</table>
Table 1-2. 272-Pin FBGA Pin Assignments (Sheet 5 of 9)

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Default Function</th>
<th>I/O State @BUS REQ</th>
<th>I/O State @PWR-off</th>
<th>I/O State @nRESET</th>
<th>I/O Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T15</td>
<td>AIN1</td>
<td>AIN1</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>r10</td>
</tr>
<tr>
<td>U17</td>
<td>AIN2</td>
<td>AIN2</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>r10</td>
</tr>
<tr>
<td>T16</td>
<td>AIN3</td>
<td>AIN3</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>r10</td>
</tr>
<tr>
<td>R15</td>
<td>AIN4</td>
<td>AIN4</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>r10</td>
</tr>
<tr>
<td>T17</td>
<td>AIN5</td>
<td>AIN5</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>r10</td>
</tr>
<tr>
<td>R16</td>
<td>AIN6</td>
<td>AIN6</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>r10</td>
</tr>
<tr>
<td>N13</td>
<td>AIN7</td>
<td>AIN7</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>r10</td>
</tr>
<tr>
<td>P15</td>
<td>VDDA_ADC</td>
<td>VDDA_ADC</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3t</td>
</tr>
<tr>
<td>R17</td>
<td>XToRtc</td>
<td>XToRtc</td>
<td>–</td>
<td>–</td>
<td>AO</td>
<td>gp</td>
</tr>
<tr>
<td>P16</td>
<td>XTrtc</td>
<td>XTrtc</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>gp</td>
</tr>
<tr>
<td>M12</td>
<td>RTCVDD</td>
<td>RTCVDD</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1I</td>
</tr>
<tr>
<td>P14</td>
<td>VDDi_MPLL</td>
<td>VDDi_MPLL</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>M13</td>
<td>VSSI_MPLL</td>
<td>VSSI_MPLL</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>P17</td>
<td>MPLLCAP</td>
<td>MPLLCAP</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>gp</td>
</tr>
<tr>
<td>L11</td>
<td>VDDi_UPLL</td>
<td>VDDi_UPLL</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>N15</td>
<td>VSSI_UPLL</td>
<td>VSSI_UPLL</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>L13</td>
<td>UPLLCAP</td>
<td>UPLLCAP</td>
<td>–</td>
<td>–</td>
<td>Al</td>
<td>gp</td>
</tr>
<tr>
<td>N16</td>
<td>VDDOP</td>
<td>VDDOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>N14</td>
<td>EINT0/GPF0</td>
<td>GPF0</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>N17</td>
<td>EINT1/GPF1</td>
<td>GPF1</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>M16</td>
<td>EINT2/GPF2</td>
<td>GPF2</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>M17</td>
<td>EINT3/GPF3</td>
<td>GPF3</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>M15</td>
<td>EINT4/GPF4</td>
<td>GPF4</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>M14</td>
<td>EINT5/GPF5</td>
<td>GPF5</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>L15</td>
<td>EINT6/GPF6</td>
<td>GPF6</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>L17</td>
<td>EINT7/GPF7</td>
<td>GPF7</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>L16</td>
<td>UCLK/GPH8</td>
<td>GPH8</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>L14</td>
<td>nCTS0/GPH0</td>
<td>GPH0</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>L12</td>
<td>nRTS0/GPH1</td>
<td>GPH1</td>
<td>–/–</td>
<td>O(H)/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>K15</td>
<td>TXD0/GPH2</td>
<td>GPH2</td>
<td>–/–</td>
<td>O(H)/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>K17</td>
<td>RXD0/GPH3</td>
<td>GPH3</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>K16</td>
<td>TXD1/GPH4</td>
<td>GPH4</td>
<td>–/–</td>
<td>O(H)/–</td>
<td>I</td>
<td>18</td>
</tr>
<tr>
<td>K14</td>
<td>RXD1/GPH5</td>
<td>GPH5</td>
<td>–/–</td>
<td>–/–</td>
<td>I</td>
<td>18</td>
</tr>
</tbody>
</table>
Table 1-2. 272-Pin FBGA Pin Assignments (Sheet 6 of 9)

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Default Function</th>
<th>I/O State @BUS REQ</th>
<th>I/O State @PWR-off</th>
<th>I/O State @nRESET</th>
<th>I/O Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>K13</td>
<td>TXD2/nRTS1/GPH6</td>
<td>GPH6</td>
<td>–/-</td>
<td>O(H)/–</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>K12</td>
<td>RXD2/nCTS1/GPH7</td>
<td>GPH7</td>
<td>–/-</td>
<td>–/-</td>
<td>I</td>
<td>t8</td>
</tr>
<tr>
<td>J17</td>
<td>nBATT_FLT</td>
<td>nBATT_FLT</td>
<td>–</td>
<td>–</td>
<td>I</td>
<td>is</td>
</tr>
<tr>
<td>J16</td>
<td>nRSTOUT/GPA21</td>
<td>nRSTOUT</td>
<td>–/-</td>
<td>O(L)/–</td>
<td>O(L)</td>
<td>b8</td>
</tr>
<tr>
<td>J15</td>
<td>PWREN</td>
<td>PWREN</td>
<td>O(H)</td>
<td>O(L)</td>
<td>O(H)</td>
<td>b8</td>
</tr>
<tr>
<td>J12</td>
<td>nRESET</td>
<td>nRESET</td>
<td>–</td>
<td>–</td>
<td>I</td>
<td>is</td>
</tr>
<tr>
<td>J14</td>
<td>VDDalive</td>
<td>VDDalive</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1i</td>
</tr>
<tr>
<td>J11</td>
<td>EXTCLK</td>
<td>EXTCLK</td>
<td>–</td>
<td>–</td>
<td>A</td>
<td>is</td>
</tr>
<tr>
<td>J13</td>
<td>VDDi</td>
<td>VDDi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>H17</td>
<td>XTIpll</td>
<td>XTIpll</td>
<td>–</td>
<td>–</td>
<td>A</td>
<td>m26</td>
</tr>
<tr>
<td>H16</td>
<td>XTopll</td>
<td>XTopll</td>
<td>–</td>
<td>–</td>
<td>AO</td>
<td>m26</td>
</tr>
<tr>
<td>H15</td>
<td>VSSi</td>
<td>VSSi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>H13</td>
<td>VSSOP</td>
<td>VSSOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>H14</td>
<td>VDDMOP</td>
<td>VDDMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>G17</td>
<td>nFCE/GPA22</td>
<td>nFCE</td>
<td>O(H)/–</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>b8</td>
</tr>
<tr>
<td>G16</td>
<td>nFRE/GPA20</td>
<td>nFRE</td>
<td>O(H)/–</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>b8</td>
</tr>
<tr>
<td>G15</td>
<td>nFWE/GPA19</td>
<td>nFWE</td>
<td>O(H)/–</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>b8</td>
</tr>
<tr>
<td>G14</td>
<td>ALE/GPA18</td>
<td>ALE</td>
<td>O(L)/–</td>
<td>O(L)/–</td>
<td>O(L)</td>
<td>b8</td>
</tr>
<tr>
<td>H12</td>
<td>CLE/GPA17</td>
<td>CLE</td>
<td>O(L)/–</td>
<td>O(L)/–</td>
<td>O(L)</td>
<td>b8</td>
</tr>
<tr>
<td>G13</td>
<td>nWAIT</td>
<td>nWAIT</td>
<td>–</td>
<td>–</td>
<td>I</td>
<td>is</td>
</tr>
<tr>
<td>F17</td>
<td>nGCS7:nSCS1</td>
<td>nGCS7</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>F16</td>
<td>nGCS6:nSCS0</td>
<td>nGCS6</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>F15</td>
<td>nGCS5/GPA16</td>
<td>nGCS5</td>
<td>Hi-z</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>E17</td>
<td>nGCS4/GPA15</td>
<td>nGCS4</td>
<td>Hi-z</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>E14</td>
<td>nGCS3/GPA14</td>
<td>nGCS3</td>
<td>Hi-z</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>E16</td>
<td>nGCS2/GPA13</td>
<td>nGCS2</td>
<td>Hi-z</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>E15</td>
<td>nGCS1/GPA12</td>
<td>nGCS1</td>
<td>Hi-z</td>
<td>O(H)/–</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>D17</td>
<td>nGCS0</td>
<td>nGCS0</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>D16</td>
<td>SCKE</td>
<td>SCKE</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>D15</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>F14</td>
<td>SCLK1</td>
<td>SCLK1</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(SCLK)</td>
<td>t16</td>
</tr>
<tr>
<td>C17</td>
<td>VDDi</td>
<td>VDDi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>F13</td>
<td>SCLK0</td>
<td>SCLK0</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(SCLK)</td>
<td>t16</td>
</tr>
<tr>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Default Function</td>
<td>I/O State @BUS REQ</td>
<td>I/O State @PWR-off</td>
<td>I/O State @nRESET</td>
<td>I/O Type</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>------------------</td>
<td>----------</td>
</tr>
<tr>
<td>B17</td>
<td>VSSi</td>
<td>VSSi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>E13</td>
<td>nWE</td>
<td>nWE</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>C16</td>
<td>nOE</td>
<td>nOE</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>A17</td>
<td>nBE0:nWBE0:DQM0</td>
<td>DQM0</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>B16</td>
<td>nBE1:nWBE1:DQM1</td>
<td>DQM1</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>C15</td>
<td>nBE2:nWBE2:DQM2</td>
<td>DQM2</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>A16</td>
<td>nBE3:nWBE3:DQM3</td>
<td>DQM3</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>B15</td>
<td>nSRAS</td>
<td>nSRAS</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>C14</td>
<td>nSCAS</td>
<td>nSCAS</td>
<td>Hi-z</td>
<td>O(H)</td>
<td>O(H)</td>
<td>ot</td>
</tr>
<tr>
<td>A15</td>
<td>VDDMOP</td>
<td>VDDMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>F12</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>B14</td>
<td>ADDR0/GPA0</td>
<td>ADDR0</td>
<td>Hi-z/~</td>
<td>O(L)/~</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>D14</td>
<td>ADDR1</td>
<td>ADDR1</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>A14</td>
<td>ADDR2</td>
<td>ADDR2</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>C13</td>
<td>ADDR3</td>
<td>ADDR3</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>B13</td>
<td>ADDR4</td>
<td>ADDR4</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>D13</td>
<td>ADDR5</td>
<td>ADDR5</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>A13</td>
<td>ADDR6</td>
<td>ADDR6</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>C12</td>
<td>ADDR7</td>
<td>ADDR7</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>B12</td>
<td>ADDR8</td>
<td>ADDR8</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>G12</td>
<td>ADDR9</td>
<td>ADDR9</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>A12</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>E11</td>
<td>VDDMOP</td>
<td>VDDMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>D12</td>
<td>ADDR10</td>
<td>ADDR10</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>E12</td>
<td>ADDR11</td>
<td>ADDR11</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>D11</td>
<td>VDDi</td>
<td>VDDi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>F11</td>
<td>VSSi</td>
<td>VSSi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>B11</td>
<td>ADDR12</td>
<td>ADDR12</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>A11</td>
<td>ADDR13</td>
<td>ADDR13</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>C11</td>
<td>ADDR14</td>
<td>ADDR14</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>G11</td>
<td>ADDR15</td>
<td>ADDR15</td>
<td>Hi-z</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>A10</td>
<td>ADDR16/GPA1</td>
<td>ADDR16</td>
<td>Hi-z</td>
<td>O(L)/~</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>B10</td>
<td>ADDR17/GPA2</td>
<td>ADDR17</td>
<td>Hi-z</td>
<td>O(L)/~</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Default Function</td>
<td>I/O State @BUS REQ</td>
<td>I/O State @PWR-off</td>
<td>I/O State @nRESET</td>
<td>I/O Type</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>C10</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>E10</td>
<td>ADDR18/GPA3</td>
<td>ADDR18</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>D10</td>
<td>ADDR19/GPA4</td>
<td>ADDR19</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>F10</td>
<td>ADDR20/GPA5</td>
<td>ADDR20</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>A9</td>
<td>ADDR21/GPA6</td>
<td>ADDR21</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>D9</td>
<td>ADDR22/GPA7</td>
<td>ADDR22</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>E9</td>
<td>ADDR23/GPA8</td>
<td>ADDR23</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>B9</td>
<td>ADDR24/GPA9</td>
<td>ADDR24</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>C9</td>
<td>ADDR25/GPA10</td>
<td>ADDR25</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>E8</td>
<td>ADDR26/GPA11</td>
<td>ADDR26</td>
<td>Hi-z/–</td>
<td>O(L)</td>
<td>O(L)</td>
<td>ot</td>
</tr>
<tr>
<td>C8</td>
<td>VDDi</td>
<td>VDDi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>F9</td>
<td>VSSI</td>
<td>VSSI</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>D8</td>
<td>VDDMOP</td>
<td>VDDMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>G9</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>B8</td>
<td>DATA0</td>
<td>DATA0</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>A8</td>
<td>DATA1</td>
<td>DATA1</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>D7</td>
<td>DATA2</td>
<td>DATA2</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>E7</td>
<td>DATA3</td>
<td>DATA3</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>C7</td>
<td>DATA4</td>
<td>DATA4</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>B7</td>
<td>DATA5</td>
<td>DATA5</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>A7</td>
<td>DATA6</td>
<td>DATA6</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>C6</td>
<td>DATA7</td>
<td>DATA7</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>A6</td>
<td>VDDMOP</td>
<td>VDDMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
<tr>
<td>F8</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>F7</td>
<td>DATA8</td>
<td>DATA8</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>B6</td>
<td>DATA9</td>
<td>DATA9</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>D6</td>
<td>DATA10</td>
<td>DATA10</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>A5</td>
<td>DATA11</td>
<td>DATA11</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>C5</td>
<td>DATA12</td>
<td>DATA12</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>B5</td>
<td>DATA13</td>
<td>DATA13</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>D5</td>
<td>DATA14</td>
<td>DATA14</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>A4</td>
<td>DATA15</td>
<td>DATA15</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>B4</td>
<td>VDDMOP</td>
<td>VDDMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d3o</td>
</tr>
</tbody>
</table>
Table 1-2. 272-Pin FBGA Pin Assignments (Sheet 9 of 9)

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Default Function</th>
<th>I/O State @BUS REQ</th>
<th>I/O State @PWR-off</th>
<th>I/O State @nRESET</th>
<th>I/O Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>E6</td>
<td>VSSMOP</td>
<td>VSSMOP</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3o</td>
</tr>
<tr>
<td>C4</td>
<td>VDDi</td>
<td>VDDi</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>d1c</td>
</tr>
<tr>
<td>F6</td>
<td>VSSI</td>
<td>VSSI</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>s3i</td>
</tr>
<tr>
<td>A3</td>
<td>DATA16</td>
<td>DATA16</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>B3</td>
<td>DATA17</td>
<td>DATA17</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>A2</td>
<td>DATA18</td>
<td>DATA18</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>A1</td>
<td>DATA19</td>
<td>DATA19</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
<tr>
<td>B2</td>
<td>DATA20</td>
<td>DATA20</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>Hi-z</td>
<td>t12</td>
</tr>
</tbody>
</table>

NOTES:
1. The @BUS REQ. shows the pin states at the external bus, which is used by the other bus master.
2. ‘—’ mark indicates the unchanged pin state at Bus Request mode.
3. Hi-z or Pre means Hi-z or Previous state and it is determined by the setting of MISCCR register.
4. AI/AO means analog input/analog output.
5. P, I, and O mean power, input and output respectively.
6. The I/O state @nRESET shows the pin status in the @nRESET duration below.
7. The table below shows I/O types and the descriptions.

<table>
<thead>
<tr>
<th>I/O Type</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1i(vdd1ih), s3i(vss3i)</td>
<td>1.8V Vdd/Vss for internal logic</td>
</tr>
<tr>
<td>d1c(vdd1ih_core), s3i(vss3i)</td>
<td>1.8V Vdd/Vss for internal logic without input driver</td>
</tr>
<tr>
<td>d3o(vdd3op), s3o(vss3op)</td>
<td>3.3V Vdd/Vss for external logic</td>
</tr>
<tr>
<td>d3t(vdd3t_abb), s3t(vss3t_abb)</td>
<td>3.3V Vdd/Vss for analog circuitry</td>
</tr>
<tr>
<td>is(phis)</td>
<td>Input pad, LVCMOS schmitt-trigger level</td>
</tr>
<tr>
<td>us(pbusb)</td>
<td>USB pad</td>
</tr>
<tr>
<td>ot(phot8)</td>
<td>Output pad, tri-state, Io=8mA</td>
</tr>
<tr>
<td>b8(phob8)</td>
<td>Output pad, Io=8mA</td>
</tr>
<tr>
<td>t16(phot16sm)</td>
<td>Output pad, tri-state, medium slew rate, Io=16mA</td>
</tr>
<tr>
<td>r10(phiar10_abb)</td>
<td>Analog input pad with 10-ohm resistor</td>
</tr>
<tr>
<td>ia(phia_abb)</td>
<td>Analog input pad</td>
</tr>
<tr>
<td>gp(phgpad_option)</td>
<td>Pad for analog pin</td>
</tr>
<tr>
<td>m26(phsoscm26)</td>
<td>Oscillator cell with enable and feedback resistor</td>
</tr>
<tr>
<td>t6(phtbsu100ct6sm)</td>
<td>Bi-directional pad, 5V tolerant LVCMOS schmitt-trigger, 100Kohm pull-up resistor with control, tri-state, Io=6mA</td>
</tr>
<tr>
<td>t8(phbsu100ct8sm)</td>
<td>Bi-directional pad, LVCMOS schmitt-trigger, 100Kohm pull-up resistor with control, tri-state, Io=8mA</td>
</tr>
<tr>
<td>t12(phbsu100ct12sm)</td>
<td>Bi-directional pad, LVCMOS schmitt-trigger, 100Kohm pull-up resistor with control, tri-state, Io=12mA</td>
</tr>
<tr>
<td>d8(phbsu100cd8sm)</td>
<td>Bi-directional pad, LVCMOS schmitt-trigger, 100Kohm pull-up resistor with control, open-drain, Io=8mA</td>
</tr>
</tbody>
</table>
### SIGNAL DESCRIPTIONS

**Table 1-3. S3C2410X Signal Descriptions (Sheet 1 of 5)**

<table>
<thead>
<tr>
<th>Signal Controller</th>
<th>I/O</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>OM[1:0]</td>
<td>I</td>
<td>OM[1:0] sets S3C2410X in the TEST mode, which is used only at fabrication. Also, it determines the bus width of nGCS0. The pull-up/down resistor determines the logic level during the RESET cycle. 00: Nand-boot 01: 16-bit 10: 32-bit 11: Test mode</td>
</tr>
<tr>
<td>ADDR[26:0]</td>
<td>O</td>
<td>ADDR[26:0] (Address Bus) outputs the memory address of the corresponding bank.</td>
</tr>
<tr>
<td>DATA[31:0]</td>
<td>IO</td>
<td>DATA[31:0] (Data Bus) inputs data during memory read and outputs data during memory write. The bus width is programmable among 8/16/32-bit.</td>
</tr>
<tr>
<td>nGCS[7:0]</td>
<td>O</td>
<td>nGCS[7:0] (General Chip Select) are activated when the address of a memory is within the address region of each bank. The number of access cycles and the bank size can be programmed.</td>
</tr>
<tr>
<td>nWE</td>
<td>O</td>
<td>nWE (Write Enable) indicates that the current bus cycle is a write cycle.</td>
</tr>
<tr>
<td>nOE</td>
<td>O</td>
<td>nOE (Output Enable) indicates that the current bus cycle is a read cycle.</td>
</tr>
<tr>
<td>nXBREQ</td>
<td>I</td>
<td>nXBREQ (Bus Hold Request) allows another bus master to request control of the local bus. BACK active indicates that bus control has been granted.</td>
</tr>
<tr>
<td>nXBACK</td>
<td>O</td>
<td>nXBACK (Bus Hold Acknowledge) indicates that the S3C2410X has surrendered control of the local bus to another bus master.</td>
</tr>
<tr>
<td>nWAIT</td>
<td>I</td>
<td>nWAIT requests to prolong a current bus cycle. As long as nWAIT is L, the current bus cycle cannot be completed.</td>
</tr>
<tr>
<td>SDRAM/SRAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nSRAS</td>
<td>O</td>
<td>SDRAM Row Address Strobe</td>
</tr>
<tr>
<td>nSCAS</td>
<td>O</td>
<td>SDRAM Column Address Strobe</td>
</tr>
<tr>
<td>nSCS[1:0]</td>
<td>O</td>
<td>SDRAM Chip Select</td>
</tr>
<tr>
<td>DQM[3:0]</td>
<td>O</td>
<td>SDRAM Data Mask</td>
</tr>
<tr>
<td>SCLK[1:0]</td>
<td>O</td>
<td>SDRAM Clock</td>
</tr>
<tr>
<td>SCKE</td>
<td>O</td>
<td>SDRAM Clock Enable</td>
</tr>
<tr>
<td>nBE[3:0]</td>
<td>O</td>
<td>Upper Byte/Lower Byte Enable (In case of 16-bit SRAM)</td>
</tr>
<tr>
<td>nWBE[3:0]</td>
<td>O</td>
<td>Write Byte Enable</td>
</tr>
<tr>
<td>NAND Flash</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLE</td>
<td>O</td>
<td>Command Latch Enable</td>
</tr>
<tr>
<td>ALE</td>
<td>O</td>
<td>Address Latch Enable</td>
</tr>
<tr>
<td>nFCE</td>
<td>O</td>
<td>Nand Flash Chip Enable</td>
</tr>
<tr>
<td>nFRE</td>
<td>O</td>
<td>Nand Flash Read Enable</td>
</tr>
<tr>
<td>nFWE</td>
<td>O</td>
<td>Nand Flash Write Enable</td>
</tr>
<tr>
<td>NCON</td>
<td>I</td>
<td>Nand Flash Configuration</td>
</tr>
<tr>
<td>R/nB</td>
<td>I</td>
<td>Nand Flash Ready/Busy</td>
</tr>
<tr>
<td>Signal</td>
<td>I/O</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------</td>
<td>-----</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LCD Control Unit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VD[23:0]</td>
<td>O</td>
<td>STN/TFT/SEC TFT: LCD Data Bus</td>
</tr>
<tr>
<td>LCD_PWREN</td>
<td>O</td>
<td>STN/TFT/SEC TFT: LCD panel power enable control signal</td>
</tr>
<tr>
<td>VCLK</td>
<td>O</td>
<td>STN/TFT: LCD clock signal</td>
</tr>
<tr>
<td>VFRAME</td>
<td>O</td>
<td>STN: LCD Frame signal</td>
</tr>
<tr>
<td>VLINE</td>
<td>O</td>
<td>STN: LCD line signal</td>
</tr>
<tr>
<td>VM</td>
<td>O</td>
<td>STN: VM alternates the polarity of the row and column voltage</td>
</tr>
<tr>
<td>VSYNC</td>
<td>O</td>
<td>TFT: Vertical synchronous signal</td>
</tr>
<tr>
<td>HSYNC</td>
<td>O</td>
<td>TFT: Horizontal synchronous signal</td>
</tr>
<tr>
<td>VDEN</td>
<td>O</td>
<td>TFT: Data enable signal</td>
</tr>
<tr>
<td>LEND</td>
<td>O</td>
<td>TFT: Line End signal</td>
</tr>
<tr>
<td>STV</td>
<td>O</td>
<td>SEC TFT: SEC(Samsung Electronics Company) TFT LCD panel control signal</td>
</tr>
<tr>
<td>CPV</td>
<td>O</td>
<td>SEC TFT: SEC(Samsung Electronics Company) TFT LCD panel control signal</td>
</tr>
<tr>
<td>LCD_HCLK</td>
<td>O</td>
<td>SEC TFT: SEC(Samsung Electronics Company) TFT LCD panel control signal</td>
</tr>
<tr>
<td>TP</td>
<td>O</td>
<td>SEC TFT: SEC(Samsung Electronics Company) TFT LCD panel control signal</td>
</tr>
<tr>
<td>STH</td>
<td>O</td>
<td>SEC TFT: SEC(Samsung Electronics Company) TFT LCD panel control signal</td>
</tr>
<tr>
<td><strong>Interrupt Control Unit</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EINT[23:0]</td>
<td>I</td>
<td>External Interrupt request</td>
</tr>
<tr>
<td><strong>DMA</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nXDREQ[1:0]</td>
<td>I</td>
<td>External DMA request</td>
</tr>
<tr>
<td>nXDACK[1:0]</td>
<td>O</td>
<td>External DMA acknowledge</td>
</tr>
<tr>
<td><strong>UART</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RxD[2:0]</td>
<td>I</td>
<td>UART receives data input</td>
</tr>
<tr>
<td>TxD[2:0]</td>
<td>O</td>
<td>UART transmits data output</td>
</tr>
<tr>
<td>nCTS[1:0]</td>
<td>I</td>
<td>UART clear to send input signal</td>
</tr>
<tr>
<td>nRTS[1:0]</td>
<td>O</td>
<td>UART request to send output signal</td>
</tr>
<tr>
<td>UCLK</td>
<td>I</td>
<td>UART clock signal</td>
</tr>
<tr>
<td><strong>ADC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIN[7:0]</td>
<td>AI</td>
<td>ADC input[7:0]. If it isn’t used pin, it has to be Low (Ground).</td>
</tr>
<tr>
<td>Vref</td>
<td>AI</td>
<td>ADC Vref</td>
</tr>
<tr>
<td><strong>IIC-Bus</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IICSDA</td>
<td>IO</td>
<td>IIC-bus data</td>
</tr>
<tr>
<td>IICSCL</td>
<td>IO</td>
<td>IIC-bus clock</td>
</tr>
</tbody>
</table>
### Table 1-3. S3C2410X Signal Descriptions (Sheet 3 of 5)

<table>
<thead>
<tr>
<th>Signal</th>
<th>I/O</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IIS-Bus</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2SLRCK</td>
<td>IO</td>
<td>IIS-bus channel select clock</td>
</tr>
<tr>
<td>I2SSDO</td>
<td>O</td>
<td>IIS-bus serial data output</td>
</tr>
<tr>
<td>I2SSDI</td>
<td>I</td>
<td>IIS-bus serial data input</td>
</tr>
<tr>
<td>I2SSCLK</td>
<td>IO</td>
<td>IIS-bus serial clock</td>
</tr>
<tr>
<td>CDCLK</td>
<td>O</td>
<td>CODEC system clock</td>
</tr>
<tr>
<td><strong>Touch Screen</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nXPON</td>
<td>O</td>
<td>Plus X-axis on-off control signal</td>
</tr>
<tr>
<td>XMON</td>
<td>O</td>
<td>Minus X-axis on-off control signal</td>
</tr>
<tr>
<td>nYPON</td>
<td>O</td>
<td>Plus Y-axis on-off control signal</td>
</tr>
<tr>
<td>YMON</td>
<td>O</td>
<td>Minus Y-axis on-off control signal</td>
</tr>
<tr>
<td><strong>USB Host</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DN[1:0]</td>
<td>IO</td>
<td>DATA(–) from USB host</td>
</tr>
<tr>
<td>DP[1:0]</td>
<td>IO</td>
<td>DATA(+) from USB host</td>
</tr>
<tr>
<td><strong>USB Device</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDN0</td>
<td>IO</td>
<td>DATA(–) for USB peripheral</td>
</tr>
<tr>
<td>PDP0</td>
<td>IO</td>
<td>DATA(+) for USB peripheral</td>
</tr>
<tr>
<td><strong>SPI</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIMISO[1:0]</td>
<td>IO</td>
<td>SPIMISO is the master data input line, when SPI is configured as a master. When SPI is configured as a slave, these pins reverse its role.</td>
</tr>
<tr>
<td>SPIMOSI[1:0]</td>
<td>IO</td>
<td>SPIMOSI is the master data output line, when SPI is configured as a master. When SPI is configured as a slave, these pins reverse its role.</td>
</tr>
<tr>
<td>SPICLK[1:0]</td>
<td>IO</td>
<td>SPI clock</td>
</tr>
<tr>
<td>nSS[1:0]</td>
<td>I</td>
<td>SPI chip select (only for slave mode)</td>
</tr>
<tr>
<td><strong>SD</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDDAT[3:0]</td>
<td>IO</td>
<td>SD receive/transmit data</td>
</tr>
<tr>
<td>SDCMD</td>
<td>IO</td>
<td>SD receive response/ transmit command</td>
</tr>
<tr>
<td>SDCLK</td>
<td>O</td>
<td>SD clock</td>
</tr>
<tr>
<td><strong>General Port</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPn[116:0]</td>
<td>IO</td>
<td>General input/output ports (some ports are output only)</td>
</tr>
<tr>
<td><strong>TIMMER/PWM</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOUT[3:0]</td>
<td>O</td>
<td>Timer output[3:0]</td>
</tr>
<tr>
<td>TCLK[1:0]</td>
<td>I</td>
<td>External timer clock input</td>
</tr>
</tbody>
</table>
Table 1-3. S3C2410X Signal Descriptions (Sheet 4 of 5)

<table>
<thead>
<tr>
<th>Signal</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JTAG TEST LOGIC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nTRST</td>
<td>I</td>
<td>nTRST (TAP Controller Reset) resets the TAP controller at start.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If debugger is used, A 10K pull-up resistor has to be connected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If debugger (black ICE) is not used, nTRST pin must be issued by a low active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pulse (Typically connected to nRESET).</td>
</tr>
<tr>
<td>TMS</td>
<td>I</td>
<td>TMS (TAP Controller Mode Select) controls the sequence of the TAP controller's</td>
</tr>
<tr>
<td></td>
<td></td>
<td>states. A 10K pull-up resistor has to be connected to TMS pin.</td>
</tr>
<tr>
<td>TCK</td>
<td>I</td>
<td>TCK (TAP Controller Clock) provides the clock input for the JTAG logic.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A 10K pull-up resistor must be connected to TCK pin.</td>
</tr>
<tr>
<td>TDI</td>
<td>I</td>
<td>TDI (TAP Controller Data Input) is the serial input for test instructions and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data. A 10K pull-up resistor must be connected to TDI pin.</td>
</tr>
<tr>
<td>TDO</td>
<td>O</td>
<td>TDO (TAP Controller Data Output) is the serial output for test instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and data.</td>
</tr>
<tr>
<td><strong>Reset, Clock &amp; Power</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nRESET</td>
<td>ST</td>
<td>nRESET suspends any operation in progress and places S3C2410X into a known</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reset state. For a reset, nRESET must be held to L level for at least 4 FCLK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>after the processor power has been stabilized.</td>
</tr>
<tr>
<td>nRSTOUT</td>
<td>O</td>
<td>For external device reset control (nRSTOUT = nRESET &amp; nWDTRST &amp; SW_RESET)</td>
</tr>
<tr>
<td>PWREN</td>
<td>O</td>
<td>1.8V core power on-off control signal</td>
</tr>
<tr>
<td>nBATT_FLT</td>
<td>I</td>
<td>Probe for battery state (Does not wake up at power-off mode in case of low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>battery state). If it isn’t used, it has to be High (3.3V).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OM[3:2] = 00b, Crystal is used for MPLL CLK source and UPLL CLK source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OM[3:2] = 01b, Crystal is used for MPLL CLK source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and EXTCLK is used for UPLL CLK source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OM[3:2] = 10b, EXTCLK is used for MPLL CLK source and Crystal is used for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UPLL CLK source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OM[3:2] = 11b, EXTCLK is used for MPLL CLK source and UPLL CLK source.</td>
</tr>
<tr>
<td>EXTCLK</td>
<td>I</td>
<td>External clock source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 11b, EXTCLK is used for MPLL CLK source and UPLL CLK source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 10b, EXTCLK is used for MPLL CLK source only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 01b, EXTCLK is used for UPLL CLK source only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If it isn’t used, it has to be High (3.3V).</td>
</tr>
<tr>
<td>XTIpII</td>
<td>AI</td>
<td>Crystal Input for internal os circuit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 00b, XTIpII is used for MPLL CLK source and UPLL CLK source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 01b, XTIpII is used for MPLL CLK source only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 10b, XTIpII is used for UPLL CLK source only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If it isn’t used, XTIpII has to be High (3.3V).</td>
</tr>
<tr>
<td>XTOpII</td>
<td>AO</td>
<td>Crystal Output for internal os circuit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 00b, XTOpII is used for MPLL CLK source and UPLL CLK source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 01b, XTOpII is used for MPLL CLK source only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When OM[3:2] = 10b, XTOpII is used for UPLL CLK source only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If it isn’t used, it has to be a floating pin.</td>
</tr>
</tbody>
</table>
### Table 1-3. S3C2410X Signal Descriptions (Sheet 5 of 5)

<table>
<thead>
<tr>
<th>Signal</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reset, Clock &amp; Power (continued)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPLLCAP</td>
<td>AI</td>
<td>Loop filter capacitor for main clock.</td>
</tr>
<tr>
<td>U PLLCAP</td>
<td>AI</td>
<td>Loop filter capacitor for USB clock.</td>
</tr>
<tr>
<td>XTIrtc</td>
<td>AI</td>
<td>32.768 kHz crystal input for RTC. If it isn’t used, it has to be High (1.8V).</td>
</tr>
<tr>
<td>XTO rtc</td>
<td>AO</td>
<td>32.768 kHz crystal output for RTC. If it isn’t used, it has to be Float.</td>
</tr>
<tr>
<td>CLKOUT[1:0]</td>
<td>O</td>
<td>Clock output signal. The CLKSEL of MISCCR register configures the clock output mode among the MPLL CLK, UPLL CLK, FCLK, HCLK, PCLK.</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDDalive</td>
<td>P</td>
<td>S3C2410X reset block and port status register VDD(1.8V). It should be always supplied whether in normal mode or in power-off mode.</td>
</tr>
<tr>
<td>VDDi/VDDiarm</td>
<td>P</td>
<td>S3C2410X core logic VDD(1.8V) for CPU.</td>
</tr>
<tr>
<td>VSSi/VSSiarm</td>
<td>P</td>
<td>S3C2410X core logic VSS</td>
</tr>
<tr>
<td>VDDi_MPLL</td>
<td>P</td>
<td>S3C2410X MPLL analog and digital VDD (1.8 V).</td>
</tr>
<tr>
<td>VSSi_MPLL</td>
<td>P</td>
<td>S3C2410X MPLL analog and digital VSS.</td>
</tr>
<tr>
<td>VDDOP</td>
<td>P</td>
<td>S3C2410X I/O port VDD(3.3V)</td>
</tr>
<tr>
<td>VDDMOP</td>
<td>P</td>
<td>S3C2410X Memory I/O VDD 3.3V : SCLK up to 100MHz</td>
</tr>
<tr>
<td>VSSMOP</td>
<td>P</td>
<td>S3C2410X Memory I/O VSS</td>
</tr>
<tr>
<td>VSSOP</td>
<td>P</td>
<td>S3C2410X I/O port VSS</td>
</tr>
<tr>
<td>RTCVDD</td>
<td>P</td>
<td>RTC VDD (1.8 V, Not support 3.3V) (This pin must be connected to power properly if RTC isn’t used)</td>
</tr>
<tr>
<td>VDDi_UPLL</td>
<td>P</td>
<td>S3C2410X UPLL analog and digital VDD (1.8V)</td>
</tr>
<tr>
<td>VSSi_UPLL</td>
<td>P</td>
<td>S3C2410X UPLL analog and digital VSS.</td>
</tr>
<tr>
<td>VDDA_ADC</td>
<td>P</td>
<td>S3C2410X ADC VDD(3.3V)</td>
</tr>
<tr>
<td>VSSA_ADC</td>
<td>P</td>
<td>S3C2410X ADC VSS</td>
</tr>
</tbody>
</table>

**NOTES:**
1. I/O means input/output.
2. AI/AO means analog input/analog output.
3. ST means schmitt-trigger.
4. P means power.
## S3C2410X SPECIAL REGISTERS

### Table 1-4. S3C2410X Special Registers (Sheet 1 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BWSCON</td>
<td>0x48000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Bus Width &amp; Wait Status Control</td>
</tr>
<tr>
<td>BANKCON0</td>
<td>0x48000004</td>
<td></td>
<td></td>
<td></td>
<td>Boot ROM Control</td>
</tr>
<tr>
<td>BANKCON1</td>
<td>0x48000008</td>
<td></td>
<td></td>
<td></td>
<td>BANK1 Control</td>
</tr>
<tr>
<td>BANKCON2</td>
<td>0x4800000C</td>
<td></td>
<td></td>
<td></td>
<td>BANK2 Control</td>
</tr>
<tr>
<td>BANKCON3</td>
<td>0x48000010</td>
<td></td>
<td></td>
<td></td>
<td>BANK3 Control</td>
</tr>
<tr>
<td>BANKCON4</td>
<td>0x48000014</td>
<td></td>
<td></td>
<td></td>
<td>BANK4 Control</td>
</tr>
<tr>
<td>BANKCON5</td>
<td>0x48000018</td>
<td></td>
<td></td>
<td></td>
<td>BANK5 Control</td>
</tr>
<tr>
<td>BANKCON6</td>
<td>0x4800001C</td>
<td></td>
<td></td>
<td></td>
<td>BANK6 Control</td>
</tr>
<tr>
<td>BANKCON7</td>
<td>0x48000020</td>
<td></td>
<td></td>
<td></td>
<td>BANK7 Control</td>
</tr>
<tr>
<td>REFRESH</td>
<td>0x48000024</td>
<td></td>
<td></td>
<td></td>
<td>DRAM/SDRAM Refresh Control</td>
</tr>
<tr>
<td>BANKSIZE</td>
<td>0x48000028</td>
<td></td>
<td></td>
<td></td>
<td>Flexible Bank Size</td>
</tr>
<tr>
<td>MRSRB6</td>
<td>0x4800002C</td>
<td></td>
<td></td>
<td></td>
<td>Mode register set for SDRAM</td>
</tr>
<tr>
<td>MRSRB7</td>
<td>0x48000030</td>
<td></td>
<td></td>
<td></td>
<td>Mode register set for SDRAM</td>
</tr>
</tbody>
</table>
### Table 1-4. S3C2410X Special Registers (Sheet 2 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USB Host Controller</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcRevision</td>
<td>0x49000000</td>
<td>←</td>
<td>W</td>
<td></td>
<td>Control and Status Group</td>
</tr>
<tr>
<td>HcControl</td>
<td>0x49000004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcCommonStatus</td>
<td>0x49000008</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcInterruptStatus</td>
<td>0x4900000C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcInterruptEnable</td>
<td>0x49000010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcInterruptDisable</td>
<td>0x49000014</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcHCCA</td>
<td>0x49000018</td>
<td></td>
<td></td>
<td></td>
<td>Memory Pointer Group</td>
</tr>
<tr>
<td>HcPeriodCuttentED</td>
<td>0x4900001C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcControlHeadED</td>
<td>0x49000020</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcControlCurrentED</td>
<td>0x49000024</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcBulkHeadED</td>
<td>0x49000028</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcBulkCurrentED</td>
<td>0x4900002C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcDoneHead</td>
<td>0x49000030</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcRmInterval</td>
<td>0x49000034</td>
<td></td>
<td></td>
<td></td>
<td>Frame Counter Group</td>
</tr>
<tr>
<td>HcFmRemaining</td>
<td>0x49000038</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcFmNumber</td>
<td>0x4900003C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcPeriodicStart</td>
<td>0x49000040</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcLSThreshold</td>
<td>0x49000044</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcRhDescriptorA</td>
<td>0x49000048</td>
<td></td>
<td></td>
<td></td>
<td>Root Hub Group</td>
</tr>
<tr>
<td>HcRhDescriptorB</td>
<td>0x4900004C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcRhStatus</td>
<td>0x49000050</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcRhPortStatus1</td>
<td>0x49000054</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HcRhPortStatus2</td>
<td>0x49000058</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Interrupt Controller</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRCPND</td>
<td>0X4A000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Interrupt Request Status</td>
</tr>
<tr>
<td>INTMOD</td>
<td>0X4A000004</td>
<td></td>
<td></td>
<td></td>
<td>Interrupt Mode Control</td>
</tr>
<tr>
<td>INTMSK</td>
<td>0X4A000008</td>
<td></td>
<td></td>
<td>R/W</td>
<td>Interrupt Mask Control</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>0X4A00000C</td>
<td></td>
<td></td>
<td></td>
<td>IRQ Priority Control</td>
</tr>
<tr>
<td>INTPND</td>
<td>0X4A000010</td>
<td></td>
<td></td>
<td>R/W</td>
<td>Interrupt Request Status</td>
</tr>
<tr>
<td>INTOFFSET</td>
<td>0X4A000014</td>
<td></td>
<td></td>
<td></td>
<td>Interrupt request source offset</td>
</tr>
<tr>
<td>SUBSRCPND</td>
<td>0X4A000018</td>
<td></td>
<td></td>
<td>R/W</td>
<td>Sub source pending</td>
</tr>
<tr>
<td>INTSUBMSK</td>
<td>0X4A00001C</td>
<td></td>
<td></td>
<td>R/W</td>
<td>Interrupt sub mask</td>
</tr>
</tbody>
</table>
### Table 1-4. S3C2410X Special Registers (Sheet 3 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/ Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA DISRC0</td>
<td>0x4B000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 0 Initial Source</td>
</tr>
<tr>
<td>DMA DISRCC0</td>
<td>0x4B000004</td>
<td></td>
<td></td>
<td></td>
<td>DMA 0 Initial Source Control</td>
</tr>
<tr>
<td>DMA DIDST0</td>
<td>0x4B000008</td>
<td></td>
<td></td>
<td></td>
<td>DMA 0 Initial Destination</td>
</tr>
<tr>
<td>DMA DIDSTC0</td>
<td>0x4B00000C</td>
<td></td>
<td></td>
<td></td>
<td>DMA 0 Initial Destination Control</td>
</tr>
<tr>
<td>DMA DCON0</td>
<td>0x4B000010</td>
<td></td>
<td></td>
<td></td>
<td>DMA 0 Control</td>
</tr>
<tr>
<td>DMA DSTAT0</td>
<td>0x4B000014</td>
<td></td>
<td></td>
<td>R</td>
<td>DMA 0 Count</td>
</tr>
<tr>
<td>DMA DCSRC0</td>
<td>0x4B000018</td>
<td></td>
<td></td>
<td></td>
<td>DMA 0 Current Source</td>
</tr>
<tr>
<td>DMA DCST0</td>
<td>0x4B00001C</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 0 Current Destination</td>
</tr>
<tr>
<td>DMA DMASKTRIG0</td>
<td>0x4B000020</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 0 Mask Trigger</td>
</tr>
<tr>
<td>DMA DISRC1</td>
<td>0x4B000040</td>
<td></td>
<td></td>
<td></td>
<td>DMA 1 Initial Source</td>
</tr>
<tr>
<td>DMA DISRCC1</td>
<td>0x4B000044</td>
<td></td>
<td></td>
<td></td>
<td>DMA 1 Initial Source Control</td>
</tr>
<tr>
<td>DMA DIDST1</td>
<td>0x4B000048</td>
<td></td>
<td></td>
<td></td>
<td>DMA 1 Initial Destination</td>
</tr>
<tr>
<td>DMA DIDSTC1</td>
<td>0x4B00004C</td>
<td></td>
<td></td>
<td></td>
<td>DMA 1 Initial Destination Control</td>
</tr>
<tr>
<td>DMA DCON1</td>
<td>0x4B000050</td>
<td></td>
<td></td>
<td></td>
<td>DMA 1 Control</td>
</tr>
<tr>
<td>DMA DSTAT1</td>
<td>0x4B000054</td>
<td></td>
<td></td>
<td>R</td>
<td>DMA 1 Count</td>
</tr>
<tr>
<td>DMA DCSRC1</td>
<td>0x4B000058</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 1 Current Source</td>
</tr>
<tr>
<td>DMA DCST1</td>
<td>0x4B00005C</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 1 Current Destination</td>
</tr>
<tr>
<td>DMA DMASKTRIG1</td>
<td>0x4B000060</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 1 Mask Trigger</td>
</tr>
<tr>
<td>DMA DISRC2</td>
<td>0x4B000080</td>
<td></td>
<td></td>
<td></td>
<td>DMA 2 Initial Source</td>
</tr>
<tr>
<td>DMA DISRCC2</td>
<td>0x4B000084</td>
<td></td>
<td></td>
<td></td>
<td>DMA 2 Initial Source Control</td>
</tr>
<tr>
<td>DMA DIDST2</td>
<td>0x4B000088</td>
<td></td>
<td></td>
<td></td>
<td>DMA 2 Initial Destination</td>
</tr>
<tr>
<td>DMA DIDSTC2</td>
<td>0x4B00008C</td>
<td></td>
<td></td>
<td></td>
<td>DMA 2 Initial Destination Control</td>
</tr>
<tr>
<td>DMA DCON2</td>
<td>0x4B000090</td>
<td></td>
<td></td>
<td></td>
<td>DMA 2 Control</td>
</tr>
<tr>
<td>DMA DSTAT2</td>
<td>0x4B000094</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 2 Count</td>
</tr>
<tr>
<td>DMA DCSRC2</td>
<td>0x4B000098</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 2 Current Source</td>
</tr>
<tr>
<td>DMA DCST2</td>
<td>0x4B00009C</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 2 Current Destination</td>
</tr>
<tr>
<td>DMA DMASKTRIG2</td>
<td>0x4B0000A0</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 2 Mask Trigger</td>
</tr>
<tr>
<td>DMA DISRC3</td>
<td>0x4B0000C0</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 3 Initial Source</td>
</tr>
<tr>
<td>DMA DISRCC3</td>
<td>0x4B0000C4</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 3 Initial Source Control</td>
</tr>
<tr>
<td>DMA DIDST3</td>
<td>0x4B0000C8</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 3 Initial Destination</td>
</tr>
<tr>
<td>DMA DIDSTC3</td>
<td>0x4B0000CC</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 3 Initial Destination Control</td>
</tr>
<tr>
<td>DMA DCON3</td>
<td>0x4B0000D0</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 3 Control</td>
</tr>
<tr>
<td>DMA DSTAT3</td>
<td>0x4B0000D4</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 3 Count</td>
</tr>
<tr>
<td>DMA DCSRC3</td>
<td>0x4B0000D8</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 3 Current Source</td>
</tr>
<tr>
<td>DMA DCST3</td>
<td>0x4B0000DC</td>
<td>←</td>
<td>W</td>
<td>R</td>
<td>DMA 3 Current Destination</td>
</tr>
<tr>
<td>DMA DMASKTRIG3</td>
<td>0x4B0000E0</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DMA 3 Mask Trigger</td>
</tr>
<tr>
<td>Register Name</td>
<td>Address (B. Endian)</td>
<td>Address (L. Endian)</td>
<td>Acc. Unit</td>
<td>Read/Write</td>
<td>Function</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Clock &amp; Power Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCKTIME</td>
<td>0x4C0000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>PLL Lock Time Counter</td>
</tr>
<tr>
<td>MPLLCON</td>
<td>0x4C0000004</td>
<td></td>
<td></td>
<td></td>
<td>MPLL Control</td>
</tr>
<tr>
<td>UPLLCON</td>
<td>0x4C0000008</td>
<td></td>
<td></td>
<td></td>
<td>UPLL Control</td>
</tr>
<tr>
<td>CLKCON</td>
<td>0x4C000000C</td>
<td></td>
<td></td>
<td></td>
<td>Clock Generator Control</td>
</tr>
<tr>
<td>CLKSLow</td>
<td>0x4C000010</td>
<td></td>
<td></td>
<td></td>
<td>Slow Clock Control</td>
</tr>
<tr>
<td>CLKDIVN</td>
<td>0x4C000014</td>
<td></td>
<td></td>
<td></td>
<td>Clock divider Control</td>
</tr>
<tr>
<td>LCD Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCDCON1</td>
<td>0x4D0000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>LCD Control 1</td>
</tr>
<tr>
<td>LCDCON2</td>
<td>0x4D0000004</td>
<td></td>
<td></td>
<td></td>
<td>LCD Control 2</td>
</tr>
<tr>
<td>LCDCON3</td>
<td>0x4D0000008</td>
<td></td>
<td></td>
<td></td>
<td>LCD Control 3</td>
</tr>
<tr>
<td>LCDCON4</td>
<td>0x4D000000C</td>
<td></td>
<td></td>
<td></td>
<td>LCD Control 4</td>
</tr>
<tr>
<td>LCDCON5</td>
<td>0x4D0000010</td>
<td></td>
<td></td>
<td></td>
<td>LCD Control 5</td>
</tr>
<tr>
<td>LCDSADDR1</td>
<td>0x4D000014</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>STN/TFT: Frame Buffer Start Address1</td>
</tr>
<tr>
<td>LCDSADDR2</td>
<td>0x4D000018</td>
<td></td>
<td></td>
<td></td>
<td>STN/TFT: Frame Buffer Start Address2</td>
</tr>
<tr>
<td>LCDSADDR3</td>
<td>0x4D00001C</td>
<td></td>
<td></td>
<td></td>
<td>STN/TFT: Virtual Screen Address Set</td>
</tr>
<tr>
<td>REDLUT</td>
<td>0x4D000020</td>
<td></td>
<td></td>
<td></td>
<td>STN: Red Lookup Table</td>
</tr>
<tr>
<td>GREENLUT</td>
<td>0x4D000024</td>
<td></td>
<td></td>
<td></td>
<td>STN: Green Lookup Table</td>
</tr>
<tr>
<td>BLUELUT</td>
<td>0x4D000028</td>
<td></td>
<td></td>
<td></td>
<td>STN: Blue Lookup Table</td>
</tr>
<tr>
<td>DITHMODE</td>
<td>0x4D00004C</td>
<td></td>
<td></td>
<td></td>
<td>STN: Dithering Mode</td>
</tr>
<tr>
<td>TPAL</td>
<td>0x4D000050</td>
<td></td>
<td></td>
<td></td>
<td>TFT: Temporary Palette</td>
</tr>
<tr>
<td>LCDINTPND</td>
<td>0x4D000054</td>
<td></td>
<td></td>
<td></td>
<td>LCD Interrupt Pending</td>
</tr>
<tr>
<td>LCDSRCPND</td>
<td>0x4D000058</td>
<td></td>
<td></td>
<td></td>
<td>LCD Interrupt Source</td>
</tr>
<tr>
<td>LCDINTMSK</td>
<td>0x4D00005C</td>
<td></td>
<td></td>
<td></td>
<td>LCD Interrupt Mask</td>
</tr>
<tr>
<td>LPCSEL</td>
<td>0x4D000060</td>
<td></td>
<td></td>
<td></td>
<td>LPC3600 Control</td>
</tr>
<tr>
<td>NAND Flash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCONF</td>
<td>0x4E000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>NAND Flash Configuration</td>
</tr>
<tr>
<td>NFCMD</td>
<td>0x4E000004</td>
<td></td>
<td></td>
<td></td>
<td>NAND Flash Command</td>
</tr>
<tr>
<td>NFADDR</td>
<td>0x4E000008</td>
<td></td>
<td></td>
<td></td>
<td>NAND Flash Address</td>
</tr>
<tr>
<td>NFDATA</td>
<td>0x4E00000C</td>
<td></td>
<td></td>
<td></td>
<td>NAND Flash Data</td>
</tr>
<tr>
<td>NFSTAT</td>
<td>0x4E000010</td>
<td>←</td>
<td>R</td>
<td></td>
<td>NAND Flash Operation Status</td>
</tr>
<tr>
<td>NFECC</td>
<td>0x4E000014</td>
<td></td>
<td>R/W</td>
<td></td>
<td>NAND Flash ECC</td>
</tr>
</tbody>
</table>
Table 1-4. S3C2410X Special Registers (Sheet 5 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ULCON0</td>
<td>0x50000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>UART 0 Line Control</td>
</tr>
<tr>
<td>UCON0</td>
<td>0x50000004</td>
<td></td>
<td></td>
<td></td>
<td>UART 0 Control</td>
</tr>
<tr>
<td>UFCON0</td>
<td>0x50000008</td>
<td></td>
<td></td>
<td></td>
<td>UART 0 FIFO Control</td>
</tr>
<tr>
<td>UMCON0</td>
<td>0x5000000C</td>
<td></td>
<td></td>
<td></td>
<td>UART 0 Modem Control</td>
</tr>
<tr>
<td>UTRSTAT0</td>
<td>0x50000010</td>
<td></td>
<td>R</td>
<td></td>
<td>UART 0 Tx/Rx Status</td>
</tr>
<tr>
<td>UERSTAT0</td>
<td>0x50000014</td>
<td></td>
<td></td>
<td></td>
<td>UART 0 Rx Error Status</td>
</tr>
<tr>
<td>UFSTAT0</td>
<td>0x50000018</td>
<td></td>
<td></td>
<td></td>
<td>UART 0 FIFO Status</td>
</tr>
<tr>
<td>UMSTAT0</td>
<td>0x5000001C</td>
<td></td>
<td></td>
<td></td>
<td>UART 0 Modem Status</td>
</tr>
<tr>
<td>UTXH0</td>
<td>0x50000023 0x50000020</td>
<td>B</td>
<td>W</td>
<td></td>
<td>UART 0 Transmission Hold</td>
</tr>
<tr>
<td>URXH0</td>
<td>0x50000027 0x50000024</td>
<td>R</td>
<td>W</td>
<td></td>
<td>UART 0 Receive Buffer</td>
</tr>
<tr>
<td>UBRDIV0</td>
<td>0x50000028</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>UART 0 Baud Rate Divisor</td>
</tr>
<tr>
<td>UART</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ULCON1</td>
<td>0x50004000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>UART 1 Line Control</td>
</tr>
<tr>
<td>UCON1</td>
<td>0x50004004</td>
<td></td>
<td></td>
<td></td>
<td>UART 1 Control</td>
</tr>
<tr>
<td>UFCON1</td>
<td>0x50004008</td>
<td></td>
<td></td>
<td></td>
<td>UART 1 FIFO Control</td>
</tr>
<tr>
<td>UMCON1</td>
<td>0x5000400C</td>
<td></td>
<td></td>
<td></td>
<td>UART 1 Modem Control</td>
</tr>
<tr>
<td>UTRSTAT1</td>
<td>0x50004010</td>
<td></td>
<td>R</td>
<td></td>
<td>UART 1 Tx/Rx Status</td>
</tr>
<tr>
<td>UERSTAT1</td>
<td>0x50004014</td>
<td></td>
<td></td>
<td></td>
<td>UART 1 Rx Error Status</td>
</tr>
<tr>
<td>UFSTAT1</td>
<td>0x50004018</td>
<td></td>
<td></td>
<td></td>
<td>UART 1 FIFO Status</td>
</tr>
<tr>
<td>UMSTAT1</td>
<td>0x5000401C</td>
<td></td>
<td></td>
<td></td>
<td>UART 1 Modem Status</td>
</tr>
<tr>
<td>UTXH1</td>
<td>0x50004023 0x50004020</td>
<td>B</td>
<td>W</td>
<td></td>
<td>UART 1 Transmission Hold</td>
</tr>
<tr>
<td>URXH1</td>
<td>0x50004027 0x50004024</td>
<td>R</td>
<td>W</td>
<td></td>
<td>UART 1 Receive Buffer</td>
</tr>
<tr>
<td>UBRDIV1</td>
<td>0x50004028</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>UART 1 Baud Rate Divisor</td>
</tr>
<tr>
<td>UART</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ULCON2</td>
<td>0x50008000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>UART 2 Line Control</td>
</tr>
<tr>
<td>UCON2</td>
<td>0x50008004</td>
<td></td>
<td></td>
<td></td>
<td>UART 2 Control</td>
</tr>
<tr>
<td>UFCON2</td>
<td>0x50008008</td>
<td></td>
<td></td>
<td></td>
<td>UART 2 FIFO Control</td>
</tr>
<tr>
<td>UTRSTAT2</td>
<td>0x50008010</td>
<td></td>
<td>R</td>
<td></td>
<td>UART 2 Tx/Rx Status</td>
</tr>
<tr>
<td>UERSTAT2</td>
<td>0x50008014</td>
<td></td>
<td></td>
<td></td>
<td>UART 2 Rx Error Status</td>
</tr>
<tr>
<td>UFSTAT2</td>
<td>0x50008018</td>
<td></td>
<td></td>
<td></td>
<td>UART 2 FIFO Status</td>
</tr>
<tr>
<td>UTXH2</td>
<td>0x50008023 0x50008020</td>
<td>B</td>
<td>W</td>
<td></td>
<td>UART 2 Transmission Hold</td>
</tr>
<tr>
<td>URXH2</td>
<td>0x50008027 0x50008024</td>
<td>R</td>
<td>W</td>
<td></td>
<td>UART 2 Receive Buffer</td>
</tr>
<tr>
<td>UBRDIV2</td>
<td>0x50008028</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>UART 2 Baud Rate Divisor</td>
</tr>
<tr>
<td>Register Name</td>
<td>Address (B. Endian)</td>
<td>Address (L. Endian)</td>
<td>Acc. Unit</td>
<td>Read/Write</td>
<td>Function</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>-----------</td>
<td>------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>PWM Timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCFG0</td>
<td>0x51000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Timer Configuration</td>
</tr>
<tr>
<td>TCFG1</td>
<td>0x51000004</td>
<td></td>
<td></td>
<td></td>
<td>Timer Configuration</td>
</tr>
<tr>
<td>TCON</td>
<td>0x51000008</td>
<td></td>
<td></td>
<td></td>
<td>Timer Control</td>
</tr>
<tr>
<td>TCNTB0</td>
<td>0x5100000C</td>
<td></td>
<td></td>
<td></td>
<td>Timer Count Buffer 0</td>
</tr>
<tr>
<td>TCMPB0</td>
<td>0x51000010</td>
<td></td>
<td></td>
<td></td>
<td>Timer Compare Buffer 0</td>
</tr>
<tr>
<td>TCNTO0</td>
<td>0x51000014</td>
<td></td>
<td>R</td>
<td></td>
<td>Timer Count Observation 0</td>
</tr>
<tr>
<td>TCNTB1</td>
<td>0x51000018</td>
<td></td>
<td>R/W</td>
<td></td>
<td>Timer Count Buffer 1</td>
</tr>
<tr>
<td>TCMPB1</td>
<td>0x5100001C</td>
<td></td>
<td></td>
<td></td>
<td>Timer Compare Buffer 1</td>
</tr>
<tr>
<td>TCNTO1</td>
<td>0x51000020</td>
<td></td>
<td>R</td>
<td></td>
<td>Timer Count Observation 1</td>
</tr>
<tr>
<td>TCNTB2</td>
<td>0x51000024</td>
<td></td>
<td>R/W</td>
<td></td>
<td>Timer Count Buffer 2</td>
</tr>
<tr>
<td>TCMPB2</td>
<td>0x51000028</td>
<td></td>
<td></td>
<td></td>
<td>Timer Compare Buffer 2</td>
</tr>
<tr>
<td>TCNTO2</td>
<td>0x5100002C</td>
<td></td>
<td>R</td>
<td></td>
<td>Timer Count Observation 2</td>
</tr>
<tr>
<td>TCNTB3</td>
<td>0x51000030</td>
<td></td>
<td>R/W</td>
<td></td>
<td>Timer Count Buffer 3</td>
</tr>
<tr>
<td>TCMPB3</td>
<td>0x51000034</td>
<td></td>
<td></td>
<td></td>
<td>Timer Compare Buffer 3</td>
</tr>
<tr>
<td>TCNTO3</td>
<td>0x51000038</td>
<td></td>
<td>R</td>
<td></td>
<td>Timer Count Observation 3</td>
</tr>
<tr>
<td>TCNTB4</td>
<td>0x5100003C</td>
<td></td>
<td>R/W</td>
<td></td>
<td>Timer Count Buffer 4</td>
</tr>
<tr>
<td>TCNTO4</td>
<td>0x51000040</td>
<td></td>
<td>R</td>
<td></td>
<td>Timer Count Observation 4</td>
</tr>
<tr>
<td>Register Name</td>
<td>Address (B. Endian)</td>
<td>Address (L. Endian)</td>
<td>Acc. Unit</td>
<td>Read/Write</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>USB Device</td>
<td></td>
<td></td>
<td></td>
<td>R/W</td>
<td>Function Address</td>
</tr>
<tr>
<td>FUNC_ADDR_REG</td>
<td>0x52000143</td>
<td>0x52000140</td>
<td>B</td>
<td>R/W</td>
<td>Function Address</td>
</tr>
<tr>
<td>PWR_REG</td>
<td>0x52000147</td>
<td>0x52000144</td>
<td></td>
<td>R/W</td>
<td>Power Management</td>
</tr>
<tr>
<td>EP_INT_REG</td>
<td>0x5200014B</td>
<td>0x52000148</td>
<td></td>
<td>R/W</td>
<td>EP Interrupt Pending and Clear</td>
</tr>
<tr>
<td>USB_INT_REG</td>
<td>0x5200015B</td>
<td>0x52000158</td>
<td></td>
<td>R/W</td>
<td>USB Interrupt Pending and Clear</td>
</tr>
<tr>
<td>EP_INT_EN_REG</td>
<td>0x5200015F</td>
<td>0x5200015C</td>
<td></td>
<td></td>
<td>Interrupt Enable</td>
</tr>
<tr>
<td>USB_INT_EN_REG</td>
<td>0x5200016F</td>
<td>0x5200016C</td>
<td></td>
<td></td>
<td>Interrupt Enable</td>
</tr>
<tr>
<td>FRAME_NUM1_REG</td>
<td>0x52000173</td>
<td>0x52000170</td>
<td>R</td>
<td></td>
<td>Frame Number Lower Byte</td>
</tr>
<tr>
<td>INDEX_REG</td>
<td>0x5200017B</td>
<td>0x52000178</td>
<td>R/W</td>
<td></td>
<td>Register Index</td>
</tr>
<tr>
<td>EP0_CSR</td>
<td>0x52000187</td>
<td>0x52000184</td>
<td></td>
<td>R/W</td>
<td>Endpoint 0 Status</td>
</tr>
<tr>
<td>IN_CSR1_REG</td>
<td>0x52000187</td>
<td>0x52000184</td>
<td></td>
<td>R/W</td>
<td>In Endpoint Control Status</td>
</tr>
<tr>
<td>IN_CSR2_REG</td>
<td>0x5200018B</td>
<td>0x52000188</td>
<td></td>
<td>R/W</td>
<td>In Endpoint Control Status</td>
</tr>
<tr>
<td>MAXP_REG</td>
<td>0x5200018F</td>
<td>0x5200018C</td>
<td></td>
<td>R/W</td>
<td>Endpoint Max Packet</td>
</tr>
<tr>
<td>OUT_CSR1_REG</td>
<td>0x52000193</td>
<td>0x52000190</td>
<td></td>
<td>R/W</td>
<td>Out Endpoint Control Status</td>
</tr>
<tr>
<td>OUT_CSR2_REG</td>
<td>0x52000197</td>
<td>0x52000194</td>
<td></td>
<td>R/W</td>
<td>Out Endpoint Control Status</td>
</tr>
<tr>
<td>OUT_FIFO_CNT1_REG</td>
<td>0x5200019B</td>
<td>0x52000198</td>
<td></td>
<td>R/W</td>
<td>Endpoint Out Write Count</td>
</tr>
<tr>
<td>OUT_FIFO_CNT2_REG</td>
<td>0x5200019F</td>
<td>0x5200019C</td>
<td></td>
<td>R/W</td>
<td>Endpoint Out Write Count</td>
</tr>
<tr>
<td>EP0_FIFO</td>
<td>0x520001C3</td>
<td>0x520001C0</td>
<td></td>
<td>R/W</td>
<td>Endpoint 0 FIFO</td>
</tr>
<tr>
<td>EP1_FIFO</td>
<td>0x520001C7</td>
<td>0x520001C4</td>
<td></td>
<td>R/W</td>
<td>Endpoint 1 FIFO</td>
</tr>
<tr>
<td>EP2_FIFO</td>
<td>0x520001CB</td>
<td>0x520001C8</td>
<td></td>
<td>R/W</td>
<td>Endpoint 2 FIFO</td>
</tr>
<tr>
<td>EP3_FIFO</td>
<td>0x520001CF</td>
<td>0x520001CC</td>
<td></td>
<td>R/W</td>
<td>Endpoint 3 FIFO</td>
</tr>
<tr>
<td>EP4_FIFO</td>
<td>0x520001D3</td>
<td>0x520001D0</td>
<td></td>
<td>R/W</td>
<td>Endpoint 4 FIFO</td>
</tr>
<tr>
<td>EP1_DMA_CON</td>
<td>0x52000203</td>
<td>0x52000200</td>
<td></td>
<td>R/W</td>
<td>EP1 DMA Interface Control</td>
</tr>
<tr>
<td>EP1_DMA_UNIT</td>
<td>0x52000207</td>
<td>0x52000204</td>
<td></td>
<td>R/W</td>
<td>EP1 DMA Tx Unit Counter</td>
</tr>
<tr>
<td>EP1_DMA_FIFO</td>
<td>0x5200020B</td>
<td>0x52000208</td>
<td></td>
<td>R/W</td>
<td>EP1 DMA Tx FIFO Counter</td>
</tr>
<tr>
<td>EP1_DMA_TX_LO</td>
<td>0x5200020F</td>
<td>0x5200020C</td>
<td></td>
<td>R/W</td>
<td>EP1 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP1_DMA_TX_MD</td>
<td>0x52000213</td>
<td>0x52000210</td>
<td></td>
<td>R/W</td>
<td>EP1 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP1_DMA_TX_HI</td>
<td>0x52000217</td>
<td>0x52000214</td>
<td></td>
<td>R/W</td>
<td>EP1 DMA Total Tx Counter</td>
</tr>
</tbody>
</table>
### Table 1-4. S3C2410X Special Registers (Sheet 8 of 11))

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/W rite</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USB Device (Continued)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP2_DMA_CON</td>
<td>0x5200021B</td>
<td>0x52000218</td>
<td>B</td>
<td>R/W</td>
<td>EP2 DMA Interface Control</td>
</tr>
<tr>
<td>EP2_DMA_UNIT</td>
<td>0x5200021F</td>
<td>0x5200021C</td>
<td></td>
<td></td>
<td>EP2 DMA Tx Unit Counter</td>
</tr>
<tr>
<td>EP2_DMA_FIFO</td>
<td>0x52000223</td>
<td>0x52000220</td>
<td></td>
<td></td>
<td>EP2 DMA Tx FIFO Counter</td>
</tr>
<tr>
<td>EP2_DMA_TX_LO</td>
<td>0x52000227</td>
<td>0x52000224</td>
<td></td>
<td></td>
<td>EP2 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP2_DMA_TX_MD</td>
<td>0x5200022B</td>
<td>0x52000228</td>
<td></td>
<td></td>
<td>EP2 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP2_DMA_TX_HI</td>
<td>0x5200022F</td>
<td>0x5200022C</td>
<td></td>
<td></td>
<td>EP2 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP3_DMA_CON</td>
<td>0x52000243</td>
<td>0x52000240</td>
<td></td>
<td></td>
<td>EP3 DMA Interface Control</td>
</tr>
<tr>
<td>EP3_DMA_UNIT</td>
<td>0x52000247</td>
<td>0x52000244</td>
<td></td>
<td></td>
<td>EP3 DMA Tx Unit Counter</td>
</tr>
<tr>
<td>EP3_DMA_FIFO</td>
<td>0x5200024B</td>
<td>0x52000248</td>
<td></td>
<td></td>
<td>EP3 DMA Tx FIFO Counter</td>
</tr>
<tr>
<td>EP3_DMA_TX_LO</td>
<td>0x5200024F</td>
<td>0x5200024C</td>
<td></td>
<td></td>
<td>EP3 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP3_DMA_TX_MD</td>
<td>0x52000253</td>
<td>0x52000250</td>
<td></td>
<td></td>
<td>EP3 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP3_DMA_TX_HI</td>
<td>0x52000257</td>
<td>0x52000254</td>
<td></td>
<td></td>
<td>EP3 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP4_DMA_CON</td>
<td>0x5200025B</td>
<td>0x52000258</td>
<td></td>
<td></td>
<td>EP4 DMA Interface Control</td>
</tr>
<tr>
<td>EP4_DMA_UNIT</td>
<td>0x5200025F</td>
<td>0x5200025C</td>
<td></td>
<td></td>
<td>EP4 DMA Tx Unit Counter</td>
</tr>
<tr>
<td>EP4_DMA_FIFO</td>
<td>0x52000263</td>
<td>0x52000260</td>
<td></td>
<td></td>
<td>EP4 DMA Tx FIFO Counter</td>
</tr>
<tr>
<td>EP4_DMA_TX_LO</td>
<td>0x52000267</td>
<td>0x52000264</td>
<td></td>
<td></td>
<td>EP4 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP4_DMA_TX_MD</td>
<td>0x5200026B</td>
<td>0x52000268</td>
<td></td>
<td></td>
<td>EP4 DMA Total Tx Counter</td>
</tr>
<tr>
<td>EP4_DMA_TX_HI</td>
<td>0x5200026F</td>
<td>0x5200026C</td>
<td></td>
<td></td>
<td>EP4 DMA Total Tx Counter</td>
</tr>
<tr>
<td><strong>Watchdog Timer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WTCN</td>
<td>0x53000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Watchdog Timer Mode</td>
</tr>
<tr>
<td>WTDAT</td>
<td>0x53000004</td>
<td></td>
<td></td>
<td></td>
<td>Watchdog Timer Data</td>
</tr>
<tr>
<td>WTCNT</td>
<td>0x53000008</td>
<td></td>
<td></td>
<td></td>
<td>Watchdog Timer Count</td>
</tr>
<tr>
<td><strong>IIC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IICCON</td>
<td>0x54000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>IIC Control</td>
</tr>
<tr>
<td>IICSTAT</td>
<td>0x54000004</td>
<td></td>
<td></td>
<td></td>
<td>IIC Status</td>
</tr>
<tr>
<td>IICADD</td>
<td>0x54000008</td>
<td></td>
<td></td>
<td></td>
<td>IIC Address</td>
</tr>
<tr>
<td>IICDS</td>
<td>0x5400000C</td>
<td></td>
<td></td>
<td></td>
<td>IIC Data Shift</td>
</tr>
<tr>
<td><strong>IIS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IISCON</td>
<td>0x55000000,02</td>
<td>0x55000000</td>
<td>HW,W</td>
<td>R/W</td>
<td>IIS Control</td>
</tr>
<tr>
<td>IISMOD</td>
<td>0x55000004,06</td>
<td>0x55000004</td>
<td>HW,W</td>
<td></td>
<td>IIS Mode</td>
</tr>
<tr>
<td>IISPSR</td>
<td>0x55000008,0A</td>
<td>0x55000008</td>
<td>HW,W</td>
<td></td>
<td>IIS Prescaler</td>
</tr>
<tr>
<td>IISFCON</td>
<td>0x5500000C,0E</td>
<td>0x5500000C</td>
<td>HW,W</td>
<td></td>
<td>IIS FIFO Control</td>
</tr>
<tr>
<td>IISFIFO</td>
<td>0x55000012</td>
<td>0x55000010</td>
<td>HW</td>
<td></td>
<td>IIS FIFO Entry</td>
</tr>
</tbody>
</table>
### Table 1-4. S3C2410X Special Registers (Sheet 9 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I/O port</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPACEA</td>
<td>0x56000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Port A Control</td>
</tr>
<tr>
<td>GPACEB</td>
<td>0x56000004</td>
<td></td>
<td></td>
<td></td>
<td>Port A Data</td>
</tr>
<tr>
<td>GPACEC</td>
<td>0x56000010</td>
<td></td>
<td></td>
<td></td>
<td>Port B Control</td>
</tr>
<tr>
<td>GPACED</td>
<td>0x56000014</td>
<td></td>
<td></td>
<td></td>
<td>Port B Data</td>
</tr>
<tr>
<td>GPBUP</td>
<td>0x56000018</td>
<td></td>
<td></td>
<td></td>
<td>Pull-up Control B</td>
</tr>
<tr>
<td>GPCCON</td>
<td>0x56000020</td>
<td></td>
<td></td>
<td></td>
<td>Port C Control</td>
</tr>
<tr>
<td>GPCDAT</td>
<td>0x56000024</td>
<td></td>
<td></td>
<td></td>
<td>Port C Data</td>
</tr>
<tr>
<td>GPCEA</td>
<td>0x56000028</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Pull-up Control C</td>
</tr>
<tr>
<td>GPCEB</td>
<td>0x56000030</td>
<td></td>
<td></td>
<td></td>
<td>Port D Control</td>
</tr>
<tr>
<td>GPCEC</td>
<td>0x56000034</td>
<td></td>
<td></td>
<td></td>
<td>Port D Data</td>
</tr>
<tr>
<td>GPCEP</td>
<td>0x56000038</td>
<td></td>
<td></td>
<td></td>
<td>Pull-up Control D</td>
</tr>
<tr>
<td>GPDEA</td>
<td>0x56000040</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Port E Control</td>
</tr>
<tr>
<td>GPDEB</td>
<td>0x56000044</td>
<td></td>
<td></td>
<td></td>
<td>Port E Data</td>
</tr>
<tr>
<td>GPDEC</td>
<td>0x56000048</td>
<td></td>
<td></td>
<td></td>
<td>Pull-up Control E</td>
</tr>
<tr>
<td>GPFCON</td>
<td>0x56000050</td>
<td></td>
<td></td>
<td></td>
<td>Port F Control</td>
</tr>
<tr>
<td>GPFDAT</td>
<td>0x56000054</td>
<td></td>
<td></td>
<td></td>
<td>Port F Data</td>
</tr>
<tr>
<td>GPFEA</td>
<td>0x56000058</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Pull-up Control F</td>
</tr>
<tr>
<td>GPFEB</td>
<td>0x5600005C</td>
<td></td>
<td></td>
<td></td>
<td>Port G Control</td>
</tr>
<tr>
<td>GPFEF</td>
<td>0x56000060</td>
<td></td>
<td></td>
<td></td>
<td>Port G Data</td>
</tr>
<tr>
<td>GPFDAT</td>
<td>0x56000064</td>
<td></td>
<td></td>
<td></td>
<td>Pull-up Control G</td>
</tr>
<tr>
<td>GPFDAT</td>
<td>0x56000068</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Port H Control</td>
</tr>
<tr>
<td>GPFDAT</td>
<td>0x56000070</td>
<td></td>
<td></td>
<td></td>
<td>Port H Data</td>
</tr>
<tr>
<td>GPFDAT</td>
<td>0x56000074</td>
<td></td>
<td></td>
<td></td>
<td>Pull-up Control H</td>
</tr>
<tr>
<td>GPFDAT</td>
<td>0x56000078</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Miscellaneous Control</td>
</tr>
<tr>
<td>MISCCR</td>
<td>0x56000080</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>DCLK0/1 Control</td>
</tr>
<tr>
<td>DCLKCON</td>
<td>0x56000084</td>
<td></td>
<td></td>
<td></td>
<td>External Interrupt Control Register 0</td>
</tr>
<tr>
<td>EXTINT0</td>
<td>0x56000088</td>
<td></td>
<td></td>
<td></td>
<td>External Interrupt Control Register 1</td>
</tr>
<tr>
<td>EXTINT1</td>
<td>0x5600008C</td>
<td></td>
<td></td>
<td></td>
<td>External Interrupt Control Register 2</td>
</tr>
<tr>
<td>EXTINT2</td>
<td>0x56000090</td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>EINTFLT0</td>
<td>0x56000094</td>
<td></td>
<td></td>
<td></td>
<td>External Interrupt Filter Control Register 2</td>
</tr>
<tr>
<td>EINTFLT1</td>
<td>0x56000098</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>Reserved</td>
</tr>
<tr>
<td>EINTFLT2</td>
<td>0x5600009C</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>External Interrupt Filter Control Register 3</td>
</tr>
<tr>
<td>EINTFLT3</td>
<td>0x560000A0</td>
<td></td>
<td></td>
<td></td>
<td>External Interrupt Mask</td>
</tr>
<tr>
<td>EINTMASK</td>
<td>0x560000A4</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>External Interrupt Pending</td>
</tr>
<tr>
<td>EINTPEND</td>
<td>0x560000A8</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>External Pin Status</td>
</tr>
<tr>
<td>GSTATUS0</td>
<td>0x560000AC</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>External Pin Status</td>
</tr>
<tr>
<td>GSTATUS1</td>
<td>0x560000B0</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>External Pin Status</td>
</tr>
</tbody>
</table>
### Table 1-4. S3C2410X Special Registers (Sheet 10 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTCCON</td>
<td>0x570000043</td>
<td>0x57000040</td>
<td>B</td>
<td>R/W</td>
<td>RTC Control</td>
</tr>
<tr>
<td>TICNT</td>
<td>0x57000047</td>
<td>0x57000044</td>
<td></td>
<td>R/W</td>
<td>Tick time count</td>
</tr>
<tr>
<td>RTCALM</td>
<td>0x57000053</td>
<td>0x57000050</td>
<td></td>
<td>R/W</td>
<td>RTC Alarm Control</td>
</tr>
<tr>
<td>ALMSEC</td>
<td>0x57000057</td>
<td>0x57000054</td>
<td></td>
<td>R/W</td>
<td>Alarm Second</td>
</tr>
<tr>
<td>ALMMIN</td>
<td>0x5700005B</td>
<td>0x57000058</td>
<td></td>
<td>R/W</td>
<td>Alarm Minute</td>
</tr>
<tr>
<td>ALMOUR</td>
<td>0x5700005F</td>
<td>0x5700005C</td>
<td></td>
<td>R/W</td>
<td>Alarm Hour</td>
</tr>
<tr>
<td>ALMDATE</td>
<td>0x57000063</td>
<td>0x57000060</td>
<td></td>
<td>R/W</td>
<td>Alarm Day</td>
</tr>
<tr>
<td>ALMMON</td>
<td>0x57000067</td>
<td>0x57000064</td>
<td></td>
<td>R/W</td>
<td>Alarm Month</td>
</tr>
<tr>
<td>ALMYEAR</td>
<td>0x5700006B</td>
<td>0x57000068</td>
<td></td>
<td>R/W</td>
<td>Alarm Year</td>
</tr>
<tr>
<td>RTCRST</td>
<td>0x5700006F</td>
<td>0x5700006C</td>
<td></td>
<td>R/W</td>
<td>RTC Round Reset</td>
</tr>
<tr>
<td>BCDSEC</td>
<td>0x57000073</td>
<td>0x57000070</td>
<td></td>
<td>R/W</td>
<td>BCD Second</td>
</tr>
<tr>
<td>BCDMIN</td>
<td>0x57000077</td>
<td>0x57000074</td>
<td></td>
<td>R/W</td>
<td>BCD Minute</td>
</tr>
<tr>
<td>BCDHOUR</td>
<td>0x5700007B</td>
<td>0x57000078</td>
<td></td>
<td>R/W</td>
<td>BCD Hour</td>
</tr>
<tr>
<td>BCDDATE</td>
<td>0x5700007F</td>
<td>0x5700007C</td>
<td></td>
<td>R/W</td>
<td>BCD Day</td>
</tr>
<tr>
<td>BCDDAY</td>
<td>0x57000083</td>
<td>0x57000080</td>
<td></td>
<td>R/W</td>
<td>BCD Date</td>
</tr>
<tr>
<td>BCDMON</td>
<td>0x57000087</td>
<td>0x57000084</td>
<td></td>
<td>R/W</td>
<td>BCD Month</td>
</tr>
<tr>
<td>BCDYEAR</td>
<td>0x5700008B</td>
<td>0x57000088</td>
<td></td>
<td>R/W</td>
<td>BCD Year</td>
</tr>
<tr>
<td>A/D converter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCCON</td>
<td>0x58000000</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>ADC Control</td>
</tr>
<tr>
<td>ADCTSC</td>
<td>0x58000004</td>
<td></td>
<td></td>
<td>R/W</td>
<td>ADC Touch Screen Control</td>
</tr>
<tr>
<td>ADCDLY</td>
<td>0x58000008</td>
<td></td>
<td></td>
<td>R/W</td>
<td>ADC Start or Interval Delay</td>
</tr>
<tr>
<td>ADCDAT0</td>
<td>0x5800000C</td>
<td></td>
<td>R</td>
<td>ADC Conversion Data</td>
<td></td>
</tr>
<tr>
<td>ADCDAT1</td>
<td>0x58000010</td>
<td></td>
<td></td>
<td>ADC Conversion Data</td>
<td></td>
</tr>
<tr>
<td>SPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPCON0,1</td>
<td>0x59000000,20</td>
<td>←</td>
<td>W</td>
<td>R/W</td>
<td>SPI Control</td>
</tr>
<tr>
<td>SPSTA0,1</td>
<td>0x59000004,24</td>
<td></td>
<td>R</td>
<td>SPI Status</td>
<td></td>
</tr>
<tr>
<td>SPPIN0,1</td>
<td>0x59000008,28</td>
<td></td>
<td>R/W</td>
<td>SPI Pin Control</td>
<td></td>
</tr>
<tr>
<td>SPPRE0,1</td>
<td>0x59000000C,2C</td>
<td></td>
<td>R/W</td>
<td>SPI Baud Rate Prescaler</td>
<td></td>
</tr>
<tr>
<td>SPTDAT0,1</td>
<td>0x59000010,30</td>
<td></td>
<td>R</td>
<td>SPI Tx Data</td>
<td></td>
</tr>
<tr>
<td>SPRDAT0,1</td>
<td>0x59000014,34</td>
<td></td>
<td>R</td>
<td>SPI Rx Data</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1-4. S3C2410X Special Registers (Sheet 11 of 11)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address (B. Endian)</th>
<th>Address (L. Endian)</th>
<th>Acc. Unit</th>
<th>Read/Write</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDI interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDICON</td>
<td>0x5A000000</td>
<td></td>
<td>W</td>
<td>R/W</td>
<td>SDI Control</td>
</tr>
<tr>
<td>SDIPRE</td>
<td>0x5A000004</td>
<td></td>
<td></td>
<td></td>
<td>SDI Baud Rate Prescaler</td>
</tr>
<tr>
<td>SDICmdArg</td>
<td>0x5A000008</td>
<td></td>
<td></td>
<td></td>
<td>SDI Command Argument</td>
</tr>
<tr>
<td>SDICmdCon</td>
<td>0x5A00000C</td>
<td></td>
<td></td>
<td></td>
<td>SDI Command Control</td>
</tr>
<tr>
<td>SDICmdSta</td>
<td>0x5A000010</td>
<td></td>
<td>R/(C)</td>
<td></td>
<td>SDI Command Status</td>
</tr>
<tr>
<td>SDIRSP0</td>
<td>0x5A000014</td>
<td></td>
<td>R</td>
<td></td>
<td>SDI Response</td>
</tr>
<tr>
<td>SDIRSP1</td>
<td>0x5A000018</td>
<td></td>
<td></td>
<td></td>
<td>SDI Response</td>
</tr>
<tr>
<td>SDIRSP2</td>
<td>0x5A00001C</td>
<td></td>
<td></td>
<td></td>
<td>SDI Response</td>
</tr>
<tr>
<td>SDIRSP3</td>
<td>0x5A000020</td>
<td></td>
<td></td>
<td></td>
<td>SDI Response</td>
</tr>
<tr>
<td>SDIDTimer</td>
<td>0x5A000024</td>
<td></td>
<td>R/W</td>
<td></td>
<td>SDI Data / Busy Timer</td>
</tr>
<tr>
<td>SDIBSize</td>
<td>0x5A000028</td>
<td></td>
<td></td>
<td></td>
<td>SDI Block Size</td>
</tr>
<tr>
<td>SDIDatCon</td>
<td>0x5A00002C</td>
<td></td>
<td></td>
<td></td>
<td>SDI Data control</td>
</tr>
<tr>
<td>SDIDatCnt</td>
<td>0x5A000030</td>
<td></td>
<td>R</td>
<td></td>
<td>SDI Data Remain Counter</td>
</tr>
<tr>
<td>SDIDatSta</td>
<td>0x5A000034</td>
<td></td>
<td>R/(C)</td>
<td></td>
<td>SDI Data Status</td>
</tr>
<tr>
<td>SDIFSTA</td>
<td>0x5A000038</td>
<td></td>
<td>R</td>
<td></td>
<td>SDI FIFO Status</td>
</tr>
<tr>
<td>SDIDAT</td>
<td>0x5A00003F 0x5A00003C</td>
<td></td>
<td>B</td>
<td>R/W</td>
<td>SDI Data</td>
</tr>
<tr>
<td>SDIIntMsk</td>
<td>0x5A000040</td>
<td></td>
<td>W</td>
<td></td>
<td>SDI Interrupt Mask</td>
</tr>
</tbody>
</table>
Cautions on S3C2410X Special Registers

1. In the little endian mode, L. endian address must be used. In the big endian mode, B. endian address must be used.

2. The special registers have to be accessed for each recommended access unit.

3. All registers except ADC registers, RTC registers and UART registers must be read/written in word unit (32bit) at little/big endian.

4. Make sure that the ADC registers, RTC registers and UART registers be read/written by the specified access unit and the specified address. Moreover, one must carefully consider which endian mode is used.

5. W : 32-bit register, which must be accessed by LDR/STR or int type pointer(int *).
   HW : 16-bit register, which must be accessed by LDRH/STRH or short int type pointer(short int *).
   B : 8-bit register, which must be accessed by LDRB/STRB or char type pointer(char int *).
OVERVIEW

S3C2410X has been developed using the advanced ARM920T core, which has been designed by Advanced RISC Machines, Ltd.

PROCESSOR OPERATING STATES

From the programmer's point of view, the ARM920T can be in one of two states:

- ARM state which executes 32-bit, word-aligned ARM instructions.
- THUMB state which can execute 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

NOTES

Transition between these two states does not affect the processor mode or the contents of the registers.

SWITCHING STATE

Entering THUMB State

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register.

Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM State

Entry into ARM state happens:

- On execution of the BX instruction with the state bit clear in the operand register.
- On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.). In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

MEMORY FORMATS

ARM920T views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. ARM920T can treat words in memory as being stored either in Big-Endian or Little-Endian format.
**BIG-ENDIAN FORMAT**

In Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.

![Figure 2-1. Big-Endian Addresses of Bytes within Words](image)

**LITTLE-ENDIAN FORMAT**

In Little-Endian format, the lowest numbered byte in a word is considered the word's least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.

![Figure 2-2. Little-Endian Addresses of Bytes within Words](image)

**INSTRUCTION LENGTH**

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

**Data Types**

ARM920T supports byte (8-bit), halfword (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.
OPERATING MODES

ARM920T supports seven modes of operation:

- User (usr): The normal ARM program execution state
- FIQ (fiq): Designed to support a data transfer or channel process
- IRQ (irq): Used for general-purpose interrupt handling
- Supervisor (svc): Protected mode for the operating system
- Abort mode (abt): Entered after a data or instruction prefetch abort
- System (sys): A privileged user mode for the operating system
- Undefined (und): Entered when an undefined instruction is executed

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The non-user modes known as privileged modes are entered in order to service interrupts or exceptions, or to access protected resources.

REGISTERS

ARM920T has a total of 37 registers - 31 general-purpose 32-bit registers and six status registers - but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

The ARM State Register Set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-User) modes, mode-specific banked registers are switched in. Figure 2-3 shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information.

Register 14 is used as the subroutine link register. This receives a copy of R15 when a Branch and Link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.

Register 15 holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.

Register 16 is the CPSR (Current Program Status Register). This contains condition code flags and the current mode bits.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.
### ARM State General Registers and Program Counter

<table>
<thead>
<tr>
<th>System &amp; User</th>
<th>FIQ</th>
<th>Supervisor</th>
<th>Abort</th>
<th>IRQ</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
</tr>
<tr>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
</tr>
<tr>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
</tr>
<tr>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
</tr>
<tr>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
</tr>
<tr>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
</tr>
<tr>
<td>R8</td>
<td>R8_fiq</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
</tr>
<tr>
<td>R9</td>
<td>R9_fiq</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
</tr>
<tr>
<td>R10</td>
<td>R10_fiq</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
</tr>
<tr>
<td>R11</td>
<td>R11_fiq</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
</tr>
<tr>
<td>R12</td>
<td>R12_fiq</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
</tr>
<tr>
<td>R13</td>
<td>R13_fiq</td>
<td>R13_svc</td>
<td>R13_svc</td>
<td>R13_abt</td>
<td>R13_abt</td>
</tr>
<tr>
<td>R14</td>
<td>R14_fiq</td>
<td>R14_svc</td>
<td>R14_svc</td>
<td>R14_irq</td>
<td>R14_irq</td>
</tr>
</tbody>
</table>

### ARM State Program Status Registers

- CPSR
- SPSR_fiq
- SPSR_svc
- SPSR_abt
- SPSR_irq
- SPSR_und

= banked register

---

**Figure 2-3. Register Organization in ARM State**
The THUMB State Register Set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in Figure 2-4.

<table>
<thead>
<tr>
<th>System &amp; User</th>
<th>FIQ</th>
<th>Supervisor</th>
<th>Abort</th>
<th>IRQ</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
</tr>
<tr>
<td></td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
</tr>
<tr>
<td></td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
</tr>
<tr>
<td></td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
</tr>
<tr>
<td></td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
</tr>
<tr>
<td></td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
</tr>
<tr>
<td></td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
</tr>
<tr>
<td></td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>SP_fiq</td>
<td>SP_svc</td>
<td>SP_abt</td>
<td>SP_und</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>LR_fiq</td>
<td>LR_svc</td>
<td>LR_abt</td>
<td>LR_und</td>
</tr>
<tr>
<td></td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
</tr>
</tbody>
</table>

= banked register

Figure 2-4. Register Organization in THUMB state
The relationship between ARM and THUMB state registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0-R7 and ARM state R0-R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13
- THUMB state LR maps onto ARM state R14
- The THUMB state Program Counter maps onto the ARM state Program Counter (R15)

This relationship is shown in Figure 2-5.

![Figure 2-5. Mapping of THUMB State Registers onto ARM State Registers](image)
Accessing Hi-Registers in THUMB State

In THUMB state, registers R8-R15 (the Hi registers) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0-R7 (a Lo register) to a Hi register, and from a Hi register to a Lo register, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values with the CMP and ADD instructions. For more information, refer to Figure 3-34.

THE PROGRAM STATUS REGISTERS

The ARM920T contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These register's functions are:

- Hold information about the most recently performed ALU operation
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in Figure 2-6.

---

**Figure 2-6. Program Status Register Format**
The Condition Code Flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see Table 3-2 for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see Figure 3-46 for details.

The Control Bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will be changed when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit
This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the TBIT external signal. Note that the software must never change the state of the TBIT in the CPSR. If this happens, the processor will enter an unpredictable state.

Interrupt disable bits
The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.

The mode bits
The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor’s operating mode, as shown in Table 2-1. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.

Reserved bits
The remaining bits in the PSRs are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.
### Table 2-1. PSR Mode Bit Values

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Mode</th>
<th>Visible THUMB state registers</th>
<th>Visible ARM state registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>User</td>
<td>R7..R0, LR, SP, PC, CPSR</td>
<td>R14..R0, PC, CPSR</td>
</tr>
<tr>
<td>10001</td>
<td>FIQ</td>
<td>R7..R0, LR_fiq, SP_fiq, PC, CPSR, SPSR_fiq</td>
<td>R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq</td>
</tr>
<tr>
<td>10010</td>
<td>IRQ</td>
<td>R7..R0, LR_irq, SP_irq, PC, CPSR, SPSR_irq</td>
<td>R12..R0, R14_irq, R13_irq, PC, CPSR, SPSR_irq</td>
</tr>
<tr>
<td>10011</td>
<td>Supervisor</td>
<td>R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc</td>
<td>R12..R0, R14_svc, R13_svc, PC, CPSR, SPSR_svc</td>
</tr>
<tr>
<td>10111</td>
<td>Abort</td>
<td>R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt</td>
<td>R12..R0, R14_abt, R13_abt, PC, CPSR, SPSR_abt</td>
</tr>
<tr>
<td>11011</td>
<td>Undefined</td>
<td>R7..R0, LR_und, SP_und, PC, CPSR, SPSR_und</td>
<td>R12..R0, R14_und, R13_und, PC, CPSR</td>
</tr>
<tr>
<td>11111</td>
<td>System</td>
<td>R7..R0, LR, SP, PC, CPSR</td>
<td>R14..R0, PC, CPSR</td>
</tr>
</tbody>
</table>

**Reserved bits**

The remaining bits in the PSR’s are reserved. When changing a PSR’s flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.
EXCEPTIONS

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order. See Exception Priorities on page 2-14.

Action on Entering an Exception

When handling an exception, the ARM920T:

1. Preserves the address of the next instruction in the appropriate Link Register. If the exception has been entered from ARM state, then the address of the next instruction is copied into the Link Register (that is, current PC + 4 or PC + 8 depending on the exception. See Table 2-2 on for details). If the exception has been entered from THUMB state, then the value written into the Link Register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, MOVPS PC, R14_svc will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector

It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.

If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

Action on Leaving an Exception

On completion, the exception handler:

1. Moves the Link Register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR
3. Clears the interrupt disable flags, if they were set on entry

NOTES

An explicit switch back to THUMB state is never needed, since restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.
Exception Entry/Exit Summary

Table 2-2 summarizes the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

<table>
<thead>
<tr>
<th>Return Instruction</th>
<th>Previous State</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARM R14_x</td>
<td>THUMB R14_x</td>
</tr>
<tr>
<td>BL</td>
<td>PC + 4</td>
<td>PC + 2</td>
</tr>
<tr>
<td>SWI</td>
<td>PC + 4</td>
<td>PC + 2</td>
</tr>
<tr>
<td>UDEF</td>
<td>PC + 4</td>
<td>PC + 2</td>
</tr>
<tr>
<td>FIQ</td>
<td>PC + 4</td>
<td>PC + 4</td>
</tr>
<tr>
<td>IRQ</td>
<td>PC + 4</td>
<td>PC + 4</td>
</tr>
<tr>
<td>PABT</td>
<td>PC + 4</td>
<td>PC + 4</td>
</tr>
<tr>
<td>DABT</td>
<td>PC + 8</td>
<td>PC + 8</td>
</tr>
<tr>
<td>RESET</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

NOTES:
1. Where PC is the address of the BL/SWI/Undefined Instruction fetch which had the prefetch abort.
2. Where PC is the address of the instruction which did not get executed since the FIQ or IRQ took priority.
3. Where PC is the address of the Load or Store instruction which generated the data abort.
4. The value saved in R14_svc upon reset is unpredictable.

FIQ

The FIQ (Fast Interrupt Request) exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimizing the overhead of context switching).

FIQ is externally generated by taking the nFIQ input LOW. This input can except either synchronous or asynchronous transitions, depending on the state of the ISYNC input signal. When ISYNC is LOW, nFIQ and nIRQ are considered asynchronous, and a cycle delay for synchronization is incurred before the interrupt can affect the processor flow.

Irrespective of whether the exception was entered from ARM or Thumb state, a FIQ handler should leave the interrupt by executing

```
SUBS PC,R14_fiq,#4
```

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from User mode). If the F flag is clear, ARM920T checks for a LOW level on the output of the FIQ synchronizer at the end of each instruction.
IRQ
The IRQ (Interrupt Request) exception is a normal interrupt caused by a LOW level on the nIRQ input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS PC,R14_irq,#4
```

Abort
An abort indicates that the current memory access cannot be completed. It can be signaled by the external ABORT input. ARM920T checks for the abort exception during memory access cycles.

There are two types of abort:

- **Prefetch abort**: occurs during an instruction prefetch.
- **Data abort**: occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed - for example because a branch occurs while it is in the pipeline - the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

- Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
- The swap instruction (SWP) is aborted as though it had not been executed.
- Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (ie it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

```
SUBS PC,R14_abt,#4 ; for a prefetch abort, or
SUBS PC,R14_abt,#8 ; for a data abort
```

This restores both the PC and the CPSR, and retries the aborted instruction.
Software Interrupt
The software interrupt instruction (SWI) is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or Thumb):

```
MOV PC,R14_svc
```

This restores the PC and CPSR, and returns to the instruction following the SWI.

**NOTES**

nFIQ, nIRQ, ISYNC, LOCK, BIGEND, and ABORT pins exist only in the ARM920T CPU core.

Undefined Instruction
When ARM920T comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

```
MOVS PC,R14_und
```

This restores the CPSR and returns to the instruction following the undefined instruction.

Exception Vectors
The following table shows the exception vector addresses.

<table>
<thead>
<tr>
<th>Address</th>
<th>Exception</th>
<th>Mode in Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>Reset</td>
<td>Supervisor</td>
</tr>
<tr>
<td>0x00000004</td>
<td>Undefined instruction</td>
<td>Undefined</td>
</tr>
<tr>
<td>0x00000008</td>
<td>Software Interrupt</td>
<td>Supervisor</td>
</tr>
<tr>
<td>0x0000000C</td>
<td>Abort (prefetch)</td>
<td>Abort</td>
</tr>
<tr>
<td>0x00000010</td>
<td>Abort (data)</td>
<td>Abort</td>
</tr>
<tr>
<td>0x00000014</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x00000018</td>
<td>IRQ</td>
<td>IRQ</td>
</tr>
<tr>
<td>0x0000001C</td>
<td>FIQ</td>
<td>FIQ</td>
</tr>
</tbody>
</table>
Exception Priorities
When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:
1. Reset
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort

Lowest priority:
6. Undefined Instruction, Software interrupt.

Not All Exceptions Can Occur at Once:
Undefined Instruction and Software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (ie the CPSR's F flag is clear), ARM920T enters the data abort handler and then immediately proceeds to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection. The time for this exception entry should be added to worst-case FIQ latency calculations.
INTERRUPT LATENCIES

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchronizer (Tsyncmax if asynchronous), plus the time for the longest instruction to complete (Tldm, the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (Texc), plus the time for FIQ entry (Tfiq). At the end of this time ARM920T will be executing the instruction at 0x1C.

Tsyncmax is 3 processor cycles, Tldm is 20 cycles, Texc is 3 cycles, and Tfiq is 2 cycles. The total time is therefore 28 processor cycles. This is just over 1.4 microseconds in a system which uses a continuous 20 MHz processor clock. The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time. The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchronizer (Tsyncmin) plus Tfiq. This is 4 processor cycles.

RESET

When the nRESET signal goes LOW, ARM920T abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When nRESET goes HIGH again, ARM920T:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
2. Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.
3

ARM INSTRUCTION SET

INSTRUCTION SET SUMMARY

This chapter describes the ARM instruction set in the ARM920T core.

FORMAT SUMMARY

The ARM instruction set formats are shown below.

![Figure 3-1. ARM Instruction Set Format](image)
NOTES

Some instruction codes are not defined but do not cause the Undefined instruction trap to be taken, for instance a Multiply instruction with bit 6 changed to a 1. These instructions should not be used, as their action may change in future ARM implementations.

INSTRUCTION SUMMARY

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Add with carry</td>
<td>Rd: = Rn + Op2 + Carry</td>
</tr>
<tr>
<td>ADD</td>
<td>Add</td>
<td>Rd: = Rn + Op2</td>
</tr>
<tr>
<td>AND</td>
<td>AND</td>
<td>Rd: = Rn AND Op2</td>
</tr>
<tr>
<td>B</td>
<td>Branch</td>
<td>R15: = address</td>
</tr>
<tr>
<td>BIC</td>
<td>Bit Clear</td>
<td>Rd: = Rn AND NOT Op2</td>
</tr>
<tr>
<td>BL</td>
<td>Branch with Link</td>
<td>R14: = R15, R15: = address</td>
</tr>
<tr>
<td>BX</td>
<td>Branch and Exchange</td>
<td>R15: = Rn, T bit: = Rn[0]</td>
</tr>
<tr>
<td>CDP</td>
<td>Coprocessor Data Processing</td>
<td>(Coprocessor-specific)</td>
</tr>
<tr>
<td>CMN</td>
<td>Compare Negative</td>
<td>CPSR flags: = Rn + Op2</td>
</tr>
<tr>
<td>CMP</td>
<td>Compare</td>
<td>CPSR flags: = Rn - Op2</td>
</tr>
<tr>
<td>EOR</td>
<td>Exclusive OR</td>
<td>Rd: = (Rn AND NOT Op2) OR (Op2 AND NOT Rn)</td>
</tr>
<tr>
<td>LDC</td>
<td>Load coprocessor from memory</td>
<td>Coprocessor load</td>
</tr>
<tr>
<td>LDM</td>
<td>Load multiple registers</td>
<td>Stack manipulation (Pop)</td>
</tr>
<tr>
<td>LDR</td>
<td>Load register from memory</td>
<td>Rd: = (address)</td>
</tr>
<tr>
<td>MCR</td>
<td>Move CPU register to coprocessor register</td>
<td>cRn: = rRn (&lt;op&gt;cRm)</td>
</tr>
<tr>
<td>MLA</td>
<td>Multiply Accumulate</td>
<td>Rd: = (Rm × Rs) + Rn</td>
</tr>
<tr>
<td>MOV</td>
<td>Move register or constant</td>
<td>Rd: = Op2</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Instruction</td>
<td>Action</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>MRC</td>
<td>Move from coprocessor register to CPU register</td>
<td>Rn: = cRn {&lt;op&gt;cRm}</td>
</tr>
<tr>
<td>MRS</td>
<td>Move PSR status/flags to register</td>
<td>Rn: = PSR</td>
</tr>
<tr>
<td>MSR</td>
<td>Move register to PSR status/flags</td>
<td>PSR: = Rm</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiply</td>
<td>Rd: = Rm × Rs</td>
</tr>
<tr>
<td>MVN</td>
<td>Move negative register</td>
<td>Rd: = 0 × FFFFFFFF EOR Op2</td>
</tr>
<tr>
<td>ORR</td>
<td>OR</td>
<td>Rd: = Rn OR Op2</td>
</tr>
<tr>
<td>RSB</td>
<td>Reverse Subtract</td>
<td>Rd: = Op2 - Rn</td>
</tr>
<tr>
<td>RSC</td>
<td>Reverse Subtract with Carry</td>
<td>Rd: = Op2 - Rn - 1 + Carry</td>
</tr>
<tr>
<td>SBC</td>
<td>Subtract with Carry</td>
<td>Rd: = Rn - Op2 - 1 + Carry</td>
</tr>
<tr>
<td>STC</td>
<td>Store coprocessor register to memory</td>
<td>address: = CRn</td>
</tr>
<tr>
<td>STM</td>
<td>Store Multiple</td>
<td>Stack manipulation (Push)</td>
</tr>
<tr>
<td>STR</td>
<td>Store register to memory</td>
<td>&lt;address&gt;: = Rd</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract</td>
<td>Rd: = Rn - Op2</td>
</tr>
<tr>
<td>SWI</td>
<td>Software Interrupt</td>
<td>OS call</td>
</tr>
<tr>
<td>SWP</td>
<td>Swap register with memory</td>
<td>Rd: = [Rn], [Rn] := Rm</td>
</tr>
<tr>
<td>TEQ</td>
<td>Test bitwise equality</td>
<td>CPSR flags: = Rn EOR Op2</td>
</tr>
<tr>
<td>TST</td>
<td>Test bits</td>
<td>CPSR flags: = Rn AND Op2</td>
</tr>
</tbody>
</table>
THE CONDITION FIELD

In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field. This field (bits 31:28) determines the circumstances under which an instruction is to be executed. If the state of the C, N, Z and V flags fulfills the conditions encoded by the field, the instruction is executed, otherwise it is ignored.

There are sixteen possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. For example, a Branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the Branch will only be taken if the Z flag is set.

In practice, fifteen different conditions may be used: these are listed in Table 3-2. The sixteenth (1111) is reserved, and must not be used.

In the absence of a suffix, the condition field of most instructions is set to "Always" (suffix AL). This means the instruction will always be executed regardless of the CPSR condition codes.

Table 3-2. Condition Code Summary

<table>
<thead>
<tr>
<th>Code</th>
<th>Suffix</th>
<th>Flags</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>EQ</td>
<td>Z set</td>
<td>equal</td>
</tr>
<tr>
<td>0001</td>
<td>NE</td>
<td>Z clear</td>
<td>not equal</td>
</tr>
<tr>
<td>0010</td>
<td>CS</td>
<td>C set</td>
<td>unsigned higher or same</td>
</tr>
<tr>
<td>0011</td>
<td>CC</td>
<td>C clear</td>
<td>unsigned lower</td>
</tr>
<tr>
<td>0100</td>
<td>MI</td>
<td>N set</td>
<td>negative</td>
</tr>
<tr>
<td>0101</td>
<td>PL</td>
<td>N clear</td>
<td>positive or zero</td>
</tr>
<tr>
<td>0110</td>
<td>VS</td>
<td>V set</td>
<td>overflow</td>
</tr>
<tr>
<td>0111</td>
<td>VC</td>
<td>V clear</td>
<td>no overflow</td>
</tr>
<tr>
<td>1000</td>
<td>HI</td>
<td>C set and Z clear</td>
<td>unsigned higher</td>
</tr>
<tr>
<td>1001</td>
<td>LS</td>
<td>C clear or Z set</td>
<td>unsigned lower or same</td>
</tr>
<tr>
<td>1010</td>
<td>GE</td>
<td>N equals V</td>
<td>greater or equal</td>
</tr>
<tr>
<td>1011</td>
<td>LT</td>
<td>N not equal to V</td>
<td>less than</td>
</tr>
<tr>
<td>1100</td>
<td>GT</td>
<td>Z clear AND (N equals V)</td>
<td>greater than</td>
</tr>
<tr>
<td>1101</td>
<td>LE</td>
<td>Z set OR (N not equal to V)</td>
<td>less than or equal</td>
</tr>
<tr>
<td>1110</td>
<td>AL</td>
<td>(ignored)</td>
<td>always</td>
</tr>
</tbody>
</table>
BRANCH AND EXCHANGE (BX)

This instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

This instruction performs a branch by copying the contents of a general register, Rn, into the program counter, PC. The branch causes a pipeline flush and refill from the address specified by Rn. This instruction also permits the instruction set to be exchanged. When the instruction is executed, the value of Rn[0] determines whether the instruction stream will be decoded as ARM or THUMB instructions.

![Figure 3-2. Branch and Exchange Instructions](image)

### [3:0] Operand Register
- If bit0 of Rn = 1, subsequent instructions decoded as THUMB instructions
- If bit0 of Rn = 0, subsequent instructions decoded as ARM instructions

### [31:28] Condition Field

INSTRUCTION CYCLE TIMES

The BX instruction takes $2S + 1N$ cycles to execute, where $S$ and $N$ are defined as sequential (S-cycle) and non-sequential (N-cycle), respectively.

ASSEMBLER SYNTAX

BX - branch and exchange.

BX {cond} Rn

{cond} Two character condition mnemonic. See Table 3-2.

Rn is an expression evaluating to a valid register number.

USING R15 AS AN OPERAND

If R15 is used as an operand, the behavior is undefined.
Examples

ADR R0, Into_THUMB + 1 ; Generate branch target address
; and set bit 0 high - hence
; arrive in THUMB state.
BX R0 ; Branch and change to THUMB
; state.
CODE16 Into_THUMB ; Assemble subsequent code as
; THUMB instructions

ADR R5, Back_to_ARM ; Generate branch target to word aligned address
; - hence bit 0 is low and so change back to ARM state.
BX R5 ; Branch and change back to ARM state.
ALIGN ; Word align
CODE32 Back_to_ARM ; Assemble subsequent code as ARM instructions
BRANCH AND BRANCH WITH LINK (B, BL)

The instruction is only executed if the condition is true. The various conditions are defined Table 3-2. The instruction encoding is shown in Figure 3-3, below.

<table>
<thead>
<tr>
<th>31 28 27 25 24 23</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cond</td>
<td>101</td>
</tr>
<tr>
<td>[24] Link bit</td>
<td>L</td>
</tr>
</tbody>
</table>

[24] Link bit
0 = Branch        1 = Branch with link

[31:28] Condition Field

Figure 3-3. Branch Instructions

Branch instructions contain a signed 2's complement 24 bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a Branch with Link type operation is required.

THE LINK BIT

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or LDM Rn!,{..PC} if the link register has been saved onto a stack pointed to by Rn.

INSTRUCTION CYCLE TIMES

Branch and Branch with Link instructions take 2S + 1N incremental cycles, where S and N are defined as sequential (S-cycle) and internal (I-cycle).
ASSEMBLER SYNTAX

Items in {} are optional. Items in <> must be present.

B{L}{cond} <expression>

{L}  Used to request the Branch with Link form of the instruction. If absent, R14 will not be affected by the instruction.

{cond}  A two-character mnemonic as shown in Table 3-2. If absent then AL (ALways) will be used.

<expression>  The destination. The assembler calculates the offset.

Examples

here  BAL  here ; Assembles to 0xEAFFFFFE (note effect of PC offset).
here  B  there ; Always condition used as default.
here  CMP  R1,#0 ; Compare R1 with zero and branch to fred ; if R1 was zero, otherwise continue.
here  BEQ  fred ; Continue to next instruction.
here  BL  sub+ROM ; Call subroutine at computed address.
here  ADDS  R1,#1 ; Add 1 to register 1, setting CPSR flags ; on the result then call subroutine if
here  BLCC  sub ; the C flag is clear, which will be the ; case unless R1 held 0xFFFFFFFF.
DATA PROCESSING

The data processing instruction is only executed if the condition is true. The conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-4.

<table>
<thead>
<tr>
<th>31</th>
<th>28 27 26 25 24</th>
<th>21 20 19</th>
<th>16 15</th>
<th>12 11</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cond</td>
<td>00</td>
<td>L</td>
<td>OpCode</td>
<td>S</td>
<td>Rn</td>
</tr>
</tbody>
</table>

[15:12] Destination register
0 = Branch 1 = Branch with link

[19:16] 1st operand register
0 = Branch 1 = Branch with link

[20] Set condition codes
0 = Do not after condition codes 1 = Set condition codes

[24:21] Operation codes
0000 = AND-Rd: = Op1 AND Op2
0001 = EOR-Rd: = Op1 EOR Op2
0010 = SUB-Rd: = Op1-Op2
0011 = RSB-Rd: = Op2-Op1
0100 = ADD-Rd: = Op1+Op2
0101 = ADC-Rd: = Op1+Op2+C
0110 = SBC-Rd: = Op1-Op2+C-1
0111 = RSC-Rd: = Op2-Op1+C-1
1000 = TST-set condition codes on Op1 AND Op2
1001 = TEO-set condition codes on Op1 EOR Op2
1010 = CMP-set condition codes on Op1-Op2
1011 = SMN-set condition codes on Op1+Op2
1100 = ORR-Rd: = Op1 OR Op2
1101 = MOV-Rd: = Op2
1110 = BIC-Rd: = Op1 AND NOT Op2
1111 = MVN-Rd: = NOT Op2

[25] Immediate operand
0 = Operand 2 is a register 1 = Operand 2 is an immediate value

[11:0] Operand 2 type selection


[7:0] Unsigned 8 bit immediate value [11:8] Shift applied to Imm

[31:28] Condition field

Figure 3-4. Data Processing Instructions
The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn).

The second operand may be a shifted register (Rm) or a rotated 8 bit immediate value (Imm) according to the value of the I bit in the instruction. The condition codes in the CPSR may be preserved or updated as a result of this instruction, according to the value of the S bit in the instruction.

Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set. The instructions and their effects are listed in Table 3-3.
CPSR FLAGS

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result. If the S bit is set (and Rd is not R15, see below) the V flag in the CPSR will be unaffected, the C flag will be set to the carry out from the barrel shifter (or preserved when the shift operation is LSL #0), the Z flag will be set if and only if the result is all zeros, and the N flag will be set to the logical value of bit 31 of the result.

Table 3-3. ARM Data Processing Instructions

<table>
<thead>
<tr>
<th>Assembler Mnemonic</th>
<th>OP Code</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>0000</td>
<td>Operand1 AND operand2</td>
</tr>
<tr>
<td>EOR</td>
<td>0001</td>
<td>Operand1 EOR operand2</td>
</tr>
<tr>
<td>WUB</td>
<td>0010</td>
<td>Operand1 - operand2</td>
</tr>
<tr>
<td>RSB</td>
<td>0011</td>
<td>Operand2 operand1</td>
</tr>
<tr>
<td>ADD</td>
<td>0100</td>
<td>Operand1 + operand2</td>
</tr>
<tr>
<td>ADC</td>
<td>0101</td>
<td>Operand1 + operand2 + carry</td>
</tr>
<tr>
<td>SBC</td>
<td>0110</td>
<td>Operand1 - operand2 + carry - 1</td>
</tr>
<tr>
<td>RSC</td>
<td>0111</td>
<td>Operand2 - operand1 + carry - 1</td>
</tr>
<tr>
<td>TST</td>
<td>1000</td>
<td>As AND, but result is not written</td>
</tr>
<tr>
<td>TEQ</td>
<td>1001</td>
<td>As EOR, but result is not written</td>
</tr>
<tr>
<td>CMP</td>
<td>1010</td>
<td>As SUB, but result is not written</td>
</tr>
<tr>
<td>CMN</td>
<td>1011</td>
<td>As ADD, but result is not written</td>
</tr>
<tr>
<td>ORR</td>
<td>1100</td>
<td>Operand1 OR operand2</td>
</tr>
<tr>
<td>MOV</td>
<td>1101</td>
<td>Operand2 (operand1 is ignored)</td>
</tr>
<tr>
<td>BIC</td>
<td>1110</td>
<td>Operand1 AND NOT operand2 (Bit clear)</td>
</tr>
<tr>
<td>MVN</td>
<td>1111</td>
<td>NOT operand2 (operand1 is ignored)</td>
</tr>
</tbody>
</table>

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32 bit integer (either unsigned or 2’s complement signed, the two are equivalent). If the S bit is set (and Rd is not R15) the V flag in the CPSR will be set if an overflow occurs into bit 31 of the result; this may be ignored if the operands were considered unsigned, but warns of a possible error if the operands were 2’s complement signed. The C flag will be set to the carry out of bit 31 of the ALU, the Z flag will be set if and only if the result was zero, and the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2’s complement signed).
SHIFTS

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in Figure 3-5.

![Figure 3-5. ARM Shift Operations](image)

<table>
<thead>
<tr>
<th>11</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>11</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>RS</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- **[6:5] Shift type**
  - 00 = logical left
  - 01 = logical right
  - 10 = arithmetic right
  - 11 = rotate right

- **[11:7] Shift amount**
  - 5 bit unsigned integer

- **[6:5] Shift type**
  - 00 = logical left
  - 01 = logical right
  - 10 = arithmetic right
  - 11 = rotate right

- **[11:8] Shift register**
  - Shift amount specified in bottom-byte of Rs

Instruction specified shift amount

When the shift amount is specified in the instruction, it is contained in a 5 bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded, except that the least significant discarded bit becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For example, the effect of LSL #5 is shown in Figure 3-6.

![Figure 3-6. Logical Shift Left](image)

NOTES

LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand. A logical shift right (LSR) is similar, but the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in Figure 3-7.
The form of the shift field which might be expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This preserves the sign in 2’s complement notation. For example, ASR #5 is shown in Figure 3-8.
Rotate right (ROR) operations reuse the bits which "overshoot" in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For example, ROR #5 is shown in Figure 3-9.

![Figure 3-9. Rotate Right](image)

The form of the shift field which might be expected to give ROR #0 is used to encode a special function of the barrel shifter, rotate right extended (RRX). This is a rotate right by one bit position of the 33 bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in Figure 3-10.

![Figure 3-10. Rotate Right Extended](image)
Register Specified Shift Amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.

If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

If the value in the byte is 32 or more, the result will be a logical extension of the shift described above:

1. LSL by 32 has result zero, carry out equal to bit 0 of Rm.
2. LSL by more than 32 has result zero, carry out zero.
3. LSR by 32 has result zero, carry out equal to bit 31 of Rm.
4. LSR by more than 32 has result zero, carry out zero.
5. ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.
6. ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.
7. ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

NOTES

The zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.
IMMEDIATE OPERAND ROTATES

The immediate operand rotate field is a 4 bit unsigned integer which specifies a shift operation on the 8 bit immediate value. This value is zero extended to 32 bits, and then subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example all powers of 2.

WRITING TO R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state changes which atomically restore both PC and CPSR. This form of instruction should not be used in User mode.

USING R15 AS AN OPERAND

If R15 (the PC) is used as an operand in a data processing instruction the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

TEQ, TST, CMP AND CMN OPCODES

NOTES

TEQ, TST, CMP and CMN do not write the result of their operation but do set flags in the CPSR. An assembler should always set the S flag for these instructions even if this is not specified in the mnemonic.

The TEQP form of the TEQ instruction used in earlier ARM processors must not be used: the PSR transfer operations should be used instead.

The action of TEQP in the ARM920T is to move SPSR_<mode> to the CPSR if the processor is in a privileged mode and to do nothing if in User mode.

INSTRUCTION CYCLE TIMES

Data Processing instructions vary in the number of incremental cycles taken as follows:

<table>
<thead>
<tr>
<th>Processing Type</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal data processing</td>
<td>1S</td>
</tr>
<tr>
<td>Data processing with register specified shift</td>
<td>1S + 1I</td>
</tr>
<tr>
<td>Data processing with PC written</td>
<td>2S + 1N</td>
</tr>
<tr>
<td>Data processing with register specified shift and PC written</td>
<td>2S + 1N + 1I</td>
</tr>
</tbody>
</table>

NOTE: S, N and I are as defined sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle) respectively.
ASSEMBLER SYNTAX

- MOV, MVN (single operand instructions).
  \[ <\text{opcode}\\{\text{cond}}{\{S\}} \text{Rd},<\text{Op2}> \]

- CMP, CMN, TEQ, TST (instructions which do not produce a result).
  \[ <\text{opcode}\\{\text{cond}} \text{Rn},<\text{Op2}> \]

- AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, ORR, BIC
  \[ <\text{opcode}\\{\text{cond}}{\{S\}} \text{Rd},\text{Rn},<\text{Op2}> \]

where:

\(<\text{Op2}>\) \hspace{1cm} \text{Rm}\{,<\text{shift}>\} \text{or,}\#\text{expression}\)

\(<\text{cond}>\) \hspace{1cm} \text{A two-character condition mnemonic. See Table 3-2.}

\(<\{S\}>\) \hspace{1cm} \text{Set condition codes if S present (implied for CMP, CMN, TEQ, TST).}

\(\text{Rd}, \text{Rn and Rm}\) \hspace{1cm} \text{Expressions evaluating to a register number.}

\(<\#\text{expression}>\) \hspace{1cm} \text{If this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.}

\(<\text{shift}>\) \hspace{1cm} \text{<Shiftname> <register> or <shiftname> #expression, or RRX (rotate right one bit with extend).}

\(<\text{shiftname}>\) \hspace{1cm} \text{ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL, they assemble to the same code.)}

EXAMPLES

<table>
<thead>
<tr>
<th>Instruction</th>
<th>operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDEQ</td>
<td>R2,R4,R5</td>
<td>If the Z flag is set make R2:=R4+R5</td>
</tr>
<tr>
<td>TEQS</td>
<td>R4,#3</td>
<td>Test R4 for equality with 3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(The S is in fact redundant as the assembler inserts it automatically.)</td>
</tr>
<tr>
<td>SUB</td>
<td>R4,R5,R7,LSR R2</td>
<td>Logical right shift R7 by the number in the bottom byte of R2, subtract result from R5, and put the answer into R4.</td>
</tr>
<tr>
<td>MOV</td>
<td>PC,R14</td>
<td>Return from subroutine.</td>
</tr>
<tr>
<td>MOVS</td>
<td>PC,R14</td>
<td>Return from exception and restore CPSR from SPSR_mode.</td>
</tr>
</tbody>
</table>
PSR TRANSFER (MRS, MSR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN and CMP instructions without the S flag set. The encoding is shown in Figure 3-11.

These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32 bit immediate value are written to the top four bits of the relevant PSR.

OPERAND RESTRICTIONS

- In user mode, the control bits of the CPSR are protected from change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes the entire CPSR can be changed.
- Note that the software must never change the state of the T bit in the CPSR. If this happens, the processor will enter an unpredictable state.
- The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.
- You must not specify R15 as the source or destination register.
- Also, do not attempt to access an SPSR in User mode, since no such register exists.
**MRS (transfer PSR contents to a register)**

31 28 27 23 22 21 16 15 12 11 0

<table>
<thead>
<tr>
<th>Cond</th>
<th>00010</th>
<th>Ps</th>
<th>001111</th>
<th>Rd</th>
<th>000000000000</th>
</tr>
</thead>
</table>

- **[15:12] Destination Register**
- **[22] Source PSR**
  - 0 = CPSR
  - 1 = SPSR_<current mode>
- **[31:28] Condition Field**

**MSR (transfer register contents to PSR)**

31 28 27 23 22 21 16 15 12 11 4 3 0

<table>
<thead>
<tr>
<th>Cond</th>
<th>00010</th>
<th>Pd</th>
<th>101001111</th>
<th>00000000</th>
<th>Rm</th>
</tr>
</thead>
</table>

- **[0] Source Register**
- **[22] Destination PSR**
  - 0 = CPSR
  - 1 = SPSR_<current mode>
- **[31:28] Condition Field**

**MSR (transfer register contents or immediate value to PSR flag bits only)**

31 28 27 26 25 24 23 22 21 12 11 0

<table>
<thead>
<tr>
<th>Cond</th>
<th>00 1 10</th>
<th>Pd</th>
<th>101001111</th>
<th>Source operand</th>
<th></th>
</tr>
</thead>
</table>

- **[22] Destination PSR**
  - 0 = CPSR
  - 1 = SPSR_<current mode>
- **[25] Immediate Operand**
  - 0 = Source operand is a register
  - 1 = SPSR_<current mode>
- **[11:0] Source Operand**
  - 11 8 7 0
  - 00000000 |
  - Rm
- **[3:0] Source Register**
- **[11:4] Source operand is an immediate value**
- **[7:0] Unsigned 8 bit immediate value**
- **[11:8] Shift applied to Imm**
- **[31:28] Condition Field**

---

Figure 3-11. PSR Transfer
RESERVED BITS

Only twelve bits of the PSR are defined in ARM920T (N,Z,C,V,I,F, T & M[4:0]); the remaining bits are reserved for use in future versions of the processor. Refer to Figure 2-6 for a full description of the PSR bits.

To ensure the maximum compatibility between ARM920T programs and future processors, the following rules should be observed:

- The reserved bits should be preserved when changing the value in a PSR.
- Programs should not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

EXAMPLES

The following sequence performs a mode change:

```
MRS    R0,CPSR ; Take a copy of the CPSR.
BIC    R0,R0,#0x1F ; Clear the mode bits.
ORR    R0,R0,#new_mode ; Select new mode
MSR    CPSR,R0 ; Write back the modified CPSR.
```

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. The following instruction sets the N,Z,C and V flags:

```
MSR    CPSR_flg,#0xF0000000 ; Set all the flags regardless of their previous state
; (does not affect any control bits).
```

No attempt should be made to write an 8 bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

INSTRUCTION CYCLE TIMES

PSR transfers take 1S incremental cycles, where S is defined as Sequential (S-cycle).
ASSEMBLY SYNTAX

- **MRS** - transfer PSR contents to a register
  
  MRS{cond} Rd,<psr>

- **MSR** - transfer register contents to PSR
  
  MSR{cond} <psr>,Rm

- **MSR** - transfer register contents to PSR flag bits only
  
  MSR{cond} <psrf>,Rm

The most significant four bits of the register contents are written to the N,Z,C & V flags respectively.

- **MSR** - transfer immediate value to PSR flag bits only
  
  MSR{cond} <psrf>,<#expression>

The expression should symbolise a 32 bit value of which the most significant four bits are written to the N,Z,C and V flags respectively.

**Key:**

- `{cond}` Two-character condition mnemonic. See Table 3-2.
- Rd and Rm Expressions evaluating to a register number other than R15
- <psr> CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms as are SPSR and SPSR_all)
- <psrf> CPSR_flg or SPSR_flg
- <#expression> Where this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

**EXAMPLES**

In User mode the instructions behave as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSR</td>
<td>CPSR_all,Rm ; CPSR[31:28] &lt;- Rm[31:28]</td>
</tr>
<tr>
<td>MSR</td>
<td>CPSR_flg,Rm ; CPSR[31:28] &lt;- Rm[31:28]</td>
</tr>
<tr>
<td>MSR</td>
<td>CPSR_flg,#0xA0000000 ; CPSR[31:28] &lt;- 0xA (set N,C; clear Z,V)</td>
</tr>
<tr>
<td>MRS</td>
<td>Rd,CPSR ; Rd[31:0] &lt;- CPSR[31:0]</td>
</tr>
</tbody>
</table>

In privileged modes the instructions behave as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSR</td>
<td>CPSR_all,Rm ; CPSR[31:0] &lt;- Rm[31:0]</td>
</tr>
<tr>
<td>MSR</td>
<td>CPSR_flg,Rm ; CPSR[31:28] &lt;- Rm[31:28]</td>
</tr>
<tr>
<td>MSR</td>
<td>CPSR_flg,#0x50000000 ; CPSR[31:28] &lt;- 0x5 (set Z,V; clear N,C)</td>
</tr>
<tr>
<td>MSR</td>
<td>SPSR_all,Rm ; SPSR_&lt;mode&gt;[31:0] &lt;- Rm[31:0]</td>
</tr>
<tr>
<td>MSR</td>
<td>SPSR_flg,Rm ; SPSR_&lt;mode&gt;[31:28] &lt;- Rm[31:28]</td>
</tr>
<tr>
<td>MSR</td>
<td>SPSR_flg,#0xC0000000 ; SPSR_&lt;mode&gt;[31:28] &lt;- 0xC (set N,Z; clear C,V)</td>
</tr>
<tr>
<td>MRS</td>
<td>Rd,SPSR ; Rd[31:0] &lt;- SPSR_&lt;mode&gt;[31:0]</td>
</tr>
</tbody>
</table>
MULTIPLY AND MULTIPLY-ACCUMULATE (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-12.

The multiply and multiply-accumulate instructions use an 8 bit Booth's algorithm to perform integer multiplication.

---

<table>
<thead>
<tr>
<th>31</th>
<th>28</th>
<th>27</th>
<th>22 21 20 19</th>
<th>16 15 12 11</th>
<th>8 7 4 3 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cond</td>
<td>0 0 0 0 0 0</td>
<td>A S</td>
<td>Rd</td>
<td>Rn</td>
<td>Rs</td>
</tr>
</tbody>
</table>

- **[15:12]**[11:8][3:0] Operand Registers
- **[19:16]** Destination Register

- **[20]** Set Condition Code
  - 0 = Do not set condition codes
  - 1 = Set condition codes

- **[21]** Accumulate
  - 0 = Multiply only
  - 1 = Multiply and accumulate

- **[31:28]** Condition Field

---

**Figure 3-12. Multiply Instructions**

The multiply form of the instruction gives Rd:=Rm*Rs. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set. The multiply-accumulate form gives Rd:=Rm*Rs+Rn, which can save an explicit ADD instruction in some circumstances. Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

The results of a signed multiply and of an unsigned multiply of 32 bit operands differ only in the upper 32 bits - the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

For example consider the multiplication of the operands:

<table>
<thead>
<tr>
<th>Operand A</th>
<th>Operand B</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFF6</td>
<td>0x0000001</td>
<td>0xFFFFFFFF38</td>
</tr>
</tbody>
</table>
If the Operands Are Interpreted as Signed
Operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFFFF38.

If the Operands Are Interpreted as Unsigned
Operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFFFF38.

Operand Restrictions
The destination register Rd must not be the same as the operand register Rm. R15 must not be used as an operand or as the destination register.

All other register combinations will give correct results, and Rd, Rn and Rs may use the same register when required.
CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

INSTRUCTION CYCLE TIMES

MUL takes 1S + mI and MLA 1S + (m+1)I cycles to execute, where S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

\[ m \text{ which is } \]

controlled by the value of the multiplier operand specified by Rs. Its possible values are as follows:

1. If bits [32:8] of the multiplier operand are all zero or all one.
2. If bits [32:16] of the multiplier operand are all zero or all one.
3. If bits [32:24] of the multiplier operand are all zero or all one.
4. In all other cases.

ASSEMBLER SYNTAX

MUL{cond}{S} Rd,Rm,Rs
MLA{cond}{S} Rd,Rm,Rs,Rn

{cond} Two-character condition mnemonic. See Table 3-2..
{S} Set condition codes if S present
Rd, Rm, Rs and Rn Expressions evaluating to a register number other than R15.

EXAMPLES

MUL R1,R2,R3 ; R1:=R2*R3
MLAEQS R1,R2,R3,R4 ; Conditionally R1:=R2*R3+R4, Setting condition codes.
MULTIPLY LONG AND MULTIPLY-ACCUMULATE LONG (MULL, MLAL)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-13.

The multiply long instructions perform integer multiplication on two 32 bit operands and produce 64 bit results. Signed and unsigned multiplication each with optional accumulate give rise to four variations.

```
<table>
<thead>
<tr>
<th>31</th>
<th>28 27</th>
<th>26 25</th>
<th>24 23</th>
<th>22 21</th>
<th>20 19</th>
<th>18 16</th>
<th>15 12</th>
<th>11 8</th>
<th>7 4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cond</td>
<td>0 1</td>
<td>U</td>
<td>A</td>
<td>S</td>
<td>RdHi</td>
<td>RdLo</td>
<td>Rs</td>
<td>1 0</td>
<td>1</td>
<td>Rm</td>
<td></td>
</tr>
</tbody>
</table>

[11:8][3:0] Operand Registers
[19:16][15:12] Source Destination Registers

[20] Set Condition Code
0 = Do not alter condition codes
1 = Set condition codes

[21] Accumulate
0 = Multiply only
1 = Multiply and accumulate

[22] Unsigned
0 = Unsigned
1 = Signed

[31:28] Condition Field
```

Figure 3-13. Multiply Long Instructions

The multiply forms (UMULL and SMULL) take two 32 bit numbers and multiply them to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs. The lower 32 bits of the 64 bit result are written to RdLo, the upper 32 bits of the result are written to RdHi.

The multiply-accumulate forms (UMLAL and SMLAL) take two 32 bit numbers, multiply them and add a 64 bit number to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs + RdHi,RdLo. The lower 32 bits of the 64 bit number to add is read from RdLo. The upper 32 bits of the 64 bit number to add is read from RdHi. The lower 32 bits of the 64 bit result are written to RdLo. The upper 32 bits of the 64 bit result are written to RdHi.

The UMULL and UMLAL instructions treat all of their operands as unsigned binary numbers and write an unsigned 64 bit result. The SMULL and SMLAL instructions treat all of their operands as two's-complement signed numbers and write a two's-complement signed 64 bit result.
OPERAND RESTRICTIONS

- R15 must not be used as an operand or as a destination register.
- RdHi, RdLo, and Rm must all specify different registers.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N and Z flags are set correctly on the result (N is equal to bit 63 of the result, Z is set if and only if all 64 bits of the result are zero). Both the C and V flags are set to meaningless values.

INSTRUCTION CYCLE TIMES

MULL takes $1S + (m+1)I$ and MLAL $1S + (m+2)I$ cycles to execute, where $m$ is the number of 8 bit multiplier array cycles required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs.

Its possible values are as follows:

For Signed INSTRUCTIONS SMULL, SMLAL:

- If bits [31:8] of the multiplier operand are all zero or all one.
- If bits [31:16] of the multiplier operand are all zero or all one.
- If bits [31:24] of the multiplier operand are all zero or all one.
- In all other cases.

For Unsigned Instructions UMULL, UMLAL:

- If bits [31:8] of the multiplier operand are all zero.
- If bits [31:16] of the multiplier operand are all zero.
  - If bits [31:24] of the multiplier operand are all zero.
- In all other cases.

S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.
ASSEMBLER SYNTAX

Table 3-5. Assembler Syntax Descriptions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMULL{cond}{S} RdLo,RdHi,Rm,Rs</td>
<td>Unsigned Multiply Long</td>
<td>32 × 32 = 64</td>
</tr>
<tr>
<td>UMLAL{cond}{S} RdLo,RdHi,Rm,Rs</td>
<td>Unsigned Multiply &amp; Accumulate Long</td>
<td>32 × 32 + 64 = 64</td>
</tr>
<tr>
<td>SMULL{cond}{S} RdLo,RdHi,Rm,Rs</td>
<td>Signed Multiply Long</td>
<td>32 × 32 = 64</td>
</tr>
<tr>
<td>SMLAL{cond}{S} RdLo,RdHi,Rm,Rs</td>
<td>Signed Multiply &amp; Accumulate Long</td>
<td>32 × 32 + 64 = 64</td>
</tr>
</tbody>
</table>

where:

{cond} Two-character condition mnemonic. See Table 3-2.
{S} Set condition codes if S present
RdLo, RdHi, Rm, Rs Expressions evaluating to a register number other than R15.

EXAMPLES

UMULL R1,R4,R2,R3 ; R4,R1:=R2*R3
UMLALS R1,R5,R2,R3 ; R5,R1:=R2*R3+R5,R1 also setting condition codes
SINGLE DATA TRANSFER (LDR, STR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-14.

The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register.

The result of this calculation may be written back into the base register if auto-indexing is required.

![Table 3-2: Single Data Transfer Instruction Encoding](image)

- **[15:12] Source/Destination Registers**
- **[19:16] Base Register**
- **[20] Load/Store Bit**
  - 0 = Store to memory
  - 1 = Load from memory
- **[21] Write-back Bit**
  - 0 = No write-back
  - 1 = Write address into base
- **[22] Byte/Word Bit**
  - 0 = Transfer word quantity
  - 1 = Transfer byte quantity
- **[23] Up/Down Bit**
  - 0 = Down: subtract offset from base
  - 1 = Up: add offset to base
- **[24] Pre/Post Indexing Bit**
  - 0 = Post: add offset after transfer
  - 1 = Pre: add offset before transfer
- **[25] Immediate Offset**
  - 0 = Offset is an immediate value
- **[11:0] Offset**
  - 11: Immediate offset
  - 11: Unsigned 12-bit immediate offset
  - 3: Shifted offset
  - 4: Offset register
  - 11: Shift applied to Rm

![Figure 3-14. Single Data Transfer Instructions](image)
OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 12 bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base value may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

SHIFTED REGISTER OFFSET

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See Figure 3-5.

BYTES AND WORDS

This instruction class may be used to transfer a byte (B=1) or a word (B=0) between an ARM920T register and memory.

The action of LDR(B) and STR(B) instructions is influenced by the BIGEND control signal of ARM920T core. The two possible configurations are described below.

Little-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. Please see Figure 2-2.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.
Big-Endian Configuration
A byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. Please see Figure 2-1.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) should generate a word aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.
USE OF R15

Write-back must not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

Restriction on the use of base register

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

After an abort, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

EXAMPLE:

```
LDR R0,[R1],R1
```

Therefore a post-indexed LDR or STR where Rm is the same register as Rn should not be used.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR instructions take 1S + 1N + 1I and LDR PC take 2S + 2N +1I incremental cycles, where S,N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STR instructions take 2N incremental cycles to execute.
**ASSEMBLER SYNTAX**

\(<\text{LDR}|\text{STR}>\{\text{cond}\}\{\text{B}\}\{\text{T}\} \ \text{Rd.<Address>}\)

where:

- **LDR** Load from memory into a register
- **STR** Store from a register into memory
- \{\text{cond}\} Two-character condition mnemonic. See Table 3-2.
- \{\text{B}\} If \text{B} is present then byte transfer, otherwise word transfer
- \{\text{T}\} If \text{T} is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. \text{T} is not allowed when a pre-indexed addressing mode is specified or implied.
- **Rd** An expression evaluating to a valid register number.
- **Rn** and **Rm** Expressions evaluating to a register number. If \text{Rn} is \text{R15} then the assembler will subtract 8 from the offset value to allow for ARM920T pipelining. In this case base write-back should not be specified.

- **<Address>** can be:
  1. An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.
  2. A pre-indexed addressing specification:
     - [\text{Rn}] offset of zero
     - [\text{Rn},<#expression>][!] offset of <expression> bytes
     - [\text{Rn},{+/-}\text{Rm},{<shift>}]][!] offset of +/- contents of index register, shifted by <shift>
  3. A post-indexed addressing specification:
     - [\text{Rn},<#expression>] offset of <expression> bytes
     - [\text{Rn},{+/-}\text{Rm},{<shift>}] offset of +/- contents of index register, shifted as by <shift>.

- **<shift>** the shift General shift operation (see data processing instructions) but you cannot specify amount by a register.

- \{\text{!}\} Writes back the base register (set the W bit) if \text{!} is present.
EXAMPLES

- **STR** R1,[R2,R4]! ; Store R1 at R2+R4 (both of which are registers) and write back address to R2.
- **STR** R1,[R2],R4 ; Store R1 at R2 and write back R2+R4 to R2.
- **LDR** R1,[R2,#16] ; Load R1 from contents of R2+16, but don't write back.
- **LDR** R1,[R2,R3,LSL#2] ; Load R1 from contents of R2+R3*4.
- **LDREQB** R1,[R6,#5] ; Conditionally load byte at R6+5 into R1 bits 0 to 7, filling bits 8 to 31 with zeros.
- **STR** R1,PLACE ; Generate PC relative offset to address PLACE.
HALFWORD AND SIGNED DATA TRANSFER (LDRH/STRH/LDRSB/LDRSH)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-16.

These instructions are used to load or store half-words of data and also load sign-extended bytes or half-words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register. The result of this calculation may be written back into the base register if auto-indexing is required.

![Figure 3-16. Halfword and Signed Data Transfer with Register Offset](image-url)
OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 8-bit unsigned binary immediate value in the instruction, or a second register. The 8-bit offset is formed by concatenating bits 11 to 8 and bits 3 to 0 of the instruction word, such that bit 11 becomes the MSB and bit 0 becomes the LSB. The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base register is used as the transfer address.

The W bit gives optional auto-increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained if necessary by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base.

The Write-back bit should not be set high (W=1) when post-indexed addressing is selected.
HALFWORD LOAD AND STORES

Setting S=0 and H=1 may be used to transfer unsigned Half-words between an ARM920T register and memory.

The action of LDRH and STRH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the section below.

Signed byte and halfword loads

The S bit controls the loading of sign-extended data. When S=1 the H bit selects between Bytes (H=0) and Half-words (H=1). The L bit should not be set low (Store) when Signed (S=1) operations have been selected.

The LDRSB instruction loads the selected Byte into bits 7 to 0 of the destination register and bits 31 to 8 of the destination register are set to the value of bit 7, the sign bit.

The LDRSH instruction loads the selected Half-word into bits 15 to 0 of the destination register and bits 31 to 16 of the destination register are set to the value of bit 15, the sign bit.

The action of the LDRSB and LDRSH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the following section.

Endianness and byte/halfword selection

Little-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 7 through to 0 if the supplied address is on a word boundary, on data bus inputs 15 through to 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-2.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 15 through to 0 if the supplied address is on a word boundary and on data bus inputs 31 through to 16 if it is a halfword boundary, (A[1]=1). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM920T will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.
Big-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 31 through to 24 if the supplied address is on a word boundary, on data bus inputs 23 through to 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-1.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 31 through to 16 if the supplied address is on a word boundary and on data bus inputs 15 through to 0 if it is a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM920T will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

USE OF R15

Write-back should not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 should not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a Half-word store (STRH) instruction, the stored address will be address of the instruction plus 12.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from the main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR(H,SH,SB) instructions take 1S + 1N + 1I. LDR(H,SH,SB) PC take 2S + 2N + 1I incremental cycles. S,N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STRH instructions take 2N incremental cycles to execute.
ASSEMBLER SYNTAX

\texttt{\langle LDR|STR\rangle\{cond\}\langle H|SH|SB\rangle \textit{Rd},\langle\textit{address}\rangle}

- **LDR**: Load from memory into a register
- **STR**: Store from a register into memory
- **(cond)**: Two-character condition mnemonic. See Table 3-2
- **H**: Transfer halfword quantity
- **SB**: Load sign extended byte (Only valid for LDR)
- **SH**: Load sign extended halfword (Only valid for LDR)
- **Rd**: An expression evaluating to a valid register number.

\textit{\langle address\rangle} can be:

1. An expression which generates an address:
   The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2. A pre-indexed addressing specification:
   - \texttt{[Rn]} offset of zero
   - \texttt{[Rn,\langle expression\rangle]!} offset of \langle expression\rangle bytes
   - \texttt{[Rn,\{+/-\}Rm]!} offset of +/- contents of index register

3. A post-indexed addressing specification:
   - \texttt{[Rn,\langle expression\rangle]} offset of \langle expression\rangle bytes
   - \texttt{[Rn,\{+/-\}Rm]} offset of +/- contents of index register.

4. \texttt{[Rn]} and \texttt{Rm} are expressions evaluating to a register number. If \texttt{Rn} is R15 then the assembler will subtract 8 from the offset value to allow for ARM920T pipelining. In this case base write-back should not be specified.

\{!\} Writes back the base register (set the W bit) if \{!\} is present.
EXAMPLES

LDRH R1,[R2,-R3]! ; Load R1 from the contents of the halfword address
; contained in R2-R3 (both of which are registers)
; and write back address to R2

STRH R3,[R4,#14] ; Store the halfword in R3 at R14+14 but don't write back.

LDRESB R8,[R2],#-223 ; Load R8 with the sign extended contents of the byte
; address contained in R2 and write back R2-223 to R2.

LDRNESH R11,[R0] ; Conditionally load R11 with the sign extended contents
; of the halfword address contained in R0.

HERE ; Generate PC relative offset to address FRED.

STRH R5, [PC,#(FRED-HERE-8)]; Store the halfword in R5 at address FRED

FRED
BLOCK DATA TRANSFER (LDM, STM)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-18.

Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

THE REGISTER LIST

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16 bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory the stored value is the address of the STM instruction plus 12.

<table>
<thead>
<tr>
<th>31</th>
<th>28</th>
<th>27</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cond 100 P U S W L Rn Register list

[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] PSR & Force User Bit
0 = Do not load PSR or user mode
1 = Load PSR or force user mode

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset before transfer

[31:28] Condition Field

Figure 3-18. Block Data Transfer Instructions
ADDRESSING MODES

The transfer addresses are determined by the contents of the base register (Rn), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R15 (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of R1, R5 and R7 in the case where Rn=0x1000 and write back of the modified base is required (W=1). Figure 3.19-22 show the sequence of register transfers, the addresses used, and the value of Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn would have retained its initial value of 0x1000 unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

ADDRESS ALIGNMENT

The address should normally be a word aligned quantity and non-word aligned addresses do not affect the instruction. However, the bottom 2 bits of the address will appear on A[1:0] and might be interpreted by the memory system.

---

**Figure 3-19. Post-Increment Addressing**
Figure 3-20. Pre-Increment Addressing

Figure 3-21. Post-Decrement Addressing
USE OF THE S BIT

When the S bit is set in a LDM/STM instruction its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in Transfer List and S Bit Set (Mode Changes)
If the instruction is a LDM then SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in Transfer List and S Bit Set (User Bank Transfer)
The registers transferred are taken from the User bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

R15 not in List and S Bit Set (User Bank Transfer)
For both LDM and STM instructions, the User bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

When the instruction is LDM, care must be taken not to read from a banked register during the following cycle (inserting a dummy instruction such as MOV R0, R0 after the LDM will ensure safety).

USE OF R15 AS THE BASE
R15 should not be used as the base register in any LDM or STM instruction.
INCLUSION OF THE BASE IN THE REGISTER LIST

When write-back is specified, the base is written back at the end of the second cycle of the instruction. During a STM, the first register is written out at the start of the second cycle. A STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. A LDM will always overwrite the updated base if the base is in the list.

DATA ABORTS

Some legal addresses may be unacceptable to a memory management system, and the memory manager can indicate a problem with an address by taking the ABORT signal HIGH. This can happen on any transfer during a multiple register load or store, and must be recoverable if ARM920T is to be used in a virtual memory system.

Abort during STM Instructions

If the abort occurs during a store multiple instruction, ARM920T takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if write-back was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM Instructions

When ARM920T detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

- Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
- The base register is restored, to its modified value if write-back was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

INSTRUCTION CYCLE TIMES

Normal LDM instructions take nS + 1N + 1I and LDM PC takes (n+1)S + 2N + 1I incremental cycles, where S,N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STM instructions take (n-1)S + 2N incremental cycles to execute, where n is the number of words transferred.
ASSEMBLER SYNTAX

\[ \text{LDM|STM}\{\text{cond}\}\text{<FD|ED|FA|EA|IA|IB|DA|DB}> \text{ Rn}\{!\},<\text{Rlist}\}\{^\}\]

where:

- \{\text{cond}\} Two character condition mnemonic. See Table 3-2.
- \text{Rn} An expression evaluating to a valid register number
- \text{<Rlist>} A list of registers and register ranges enclosed in {} (e.g. \{R0,R2-R7,R10\}).
- \{!\} If present requests write-back (\(W=1\)), otherwise \(W=0\).
- \{^\} If present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode.

Addressing Mode Names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalence between the names and the values of the bits in the instruction are shown in the following table 3-6.

<table>
<thead>
<tr>
<th>Name</th>
<th>Stack</th>
<th>Other</th>
<th>L bit</th>
<th>P bit</th>
<th>U bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Increment Load</td>
<td>LDMED</td>
<td>LDMIB</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Post-Increment Load</td>
<td>LDMFD</td>
<td>LDMIA</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Pre-Decrement Load</td>
<td>LDMEA</td>
<td>LDMDB</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Post-Decrement Load</td>
<td>LDMFA</td>
<td>LMDMA</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pre-Increment Store</td>
<td>STMFA</td>
<td>STMIB</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Post-Increment Store</td>
<td>STMEA</td>
<td>STMIA</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Pre-Decrement Store</td>
<td>STMFD</td>
<td>STMDB</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Post-Decrement Store</td>
<td>STMED</td>
<td>STMDA</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

FD, ED, FA, EA define pre/post indexing and the up/down bit by reference to the form of stack required. The F and E refer to a "full" or "empty" stack, i.e. whether a pre-index has to be done (full) before storing to the stack. The A and D refer to whether the stack is ascending or descending. If ascending, a STM will go up and LDM down, if descending, vice-versa.

IA, IB, DA, DB allow control when LDM/STM are not being used for stacks and simply mean Increment After, Increment Before, Decrement After, Decrement Before.
EXAMPLES

LDMFD SP!,{R0,R1,R2} ; Unstack 3 registers.
STMIA R0,{R0-R15} ; Save all registers.
LDMFD SP!,{R15} ; R15 ← (SP), CPSR unchanged.
LDMFD SP!,{R15}^ ; R15 ← (SP), CPSR ← SPSR_mode
; (allowed only in privileged modes).
STMFD R13,{R0-R14}^ ; Save user mode regs on stack
; (allowed only in privileged modes).

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

STMED SP!,{R0-R3,R14} ; Save R0 to R3 to use as workspace
; and R14 for returning.
BL somewhere ; This nested call will overwrite R14
LDMED SP!,{R0-R3,R15} ; Restore workspace and return.
SINGLE DATA SWAP (SWP)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-23.

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together (the processor cannot be interrupted until both operations have completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

The swap address is determined by the contents of the base register (Rn). The processor first reads the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register may be specified as both the source and destination.

The LOCK output goes HIGH for the duration of the read and write operations to signal to the external memory manager that they are locked together, and should be allowed to complete without interruption. This is important in multi-processor systems where the swap instruction is the only indivisible instruction which may be used to implement semaphores; control of the memory must not be removed from a processor while it is performing a locked operation.

BYTES AND WORDS

This instruction class may be used to swap a byte (B=1) or a word (B=0) between an ARM920T register and memory. The SWP instruction is implemented as a LDR followed by a STR and the action of these is as described in the section on single data transfers. In particular, the description of Big and Little Endian configuration applies to the SWP instruction.
USE OF R15
Do not use R15 as an operand (Rd, Rn or Rs) in a SWP instruction.

DATA ABORTS
If the address used for the swap is unacceptable to a memory management system, the memory manager can flag the problem by driving ABORT HIGH. This can happen on either the read or the write cycle (or both), and in either case, the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES
Swap instructions take 1S + 2N +1I incremental cycles to execute, where S,N and I are defined as sequential (S-cycle), non-sequential, and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

```
<SWP>{cond}{B} Rd,Rm,[Rn]
```

- `{cond}` Two-character condition mnemonic. See Table 3-2.
- `{B}` If B is present then byte transfer, otherwise word transfer
- Rd,Rm,Rn Expressions evaluating to valid register numbers

Examples

```
SWP R0,R1,[R2] ; Load R0 with the word addressed by R2, and
                ; store R1 at R2.
SWPB R2,R3,[R4] ; Load R2 with the byte addressed by R4, and
                ; store bits 0 to 7 of R3 at R4.
SWPEQ R0,R0,[R1] ; Conditionally swap the contents of the
                ; word addressed by R1 with R0.
```
SOFTWARE INTERRUPT (SWI)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-24, below.

![Figure 3-24. Software Interrupt Instruction](image)

The software interrupt instruction is used to enter Supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

RETURN FROM THE SUPERVISOR

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVPC,R14_svc will return to the calling program and restore the CPSR.

Note that the link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself it must first save a copy of the return address and SPSR.

COMMENT FIELD

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

INSTRUCTION CYCLE TIMES

Software interrupt instructions take 2S + 1N incremental cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequential (N-cycle).
ASSEMBLER SYNTAX

SWI{cond} <expression>

{cond} Two character condition mnemonic, Table 3-2.

<expression> Evaluated and placed in the comment field (which is ignored by ARM920T).

Examples

SWI ReadC ; Get next character from read stream.
SWI WriteI+"k" ; Output a "k" to the write stream.
SWINE 0 ; Conditionally call supervisor with 0 in comment field.

Supervisor code

The previous examples assume that suitable supervisor code exists, for instance:

0x08 B Supervisor ; SWI entry point
EntryTable ; Addresses of supervisor routines
DCD ZeroRtn
DCD ReadCRtn
DCD WriteIRtn

Zero EQU 0

ReadC EQU 256
WriteI EQU 512

Supervisor ; SWI has routine required in bits 8-23 and data (if any) in
bits 0-7. Assumes R13_svc points to a suitable stack
STMFD R13,(R0-R2,R14) ; Save work registers and return address.
LDR R0,[R14,#-4] ; Get SWI instruction.
BIC R0,R0,#0xFF000000 ; Clear top 8 bits.
MOV R1,R0,LSR#8 ; Get routine offset.
ADR R2,EntryTable ; Get start address of entry table.
LDR R15,[R2,R1,LSL#2] ; Branch to appropriate routine.
WriteIRtn ; Enter with character in R0 bits 0-7.

* * *

LDMFD R13,(R0-R2,R15)^ ; Restore workspace and return,
; restoring processor mode and flags.
COPROCESSOR DATA OPERATIONS (CDP)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-25.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to ARM920T, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity, allowing the coprocessor and ARM920T to perform independent tasks in parallel.

COPROCESSOR INSTRUCTIONS

The S3C2410X, unlike some other ARM-based processors, does not have an external coprocessor interface. It does not have a on-chip coprocessor also.

So then all coprocessor instructions will cause the undefined instruction trap to be taken on the S3C2410X. These coprocessor instructions can be emulated by the undefined trap handler. Even though external coprocessor can not be connected to the S3C2410X, the coprocessor instructions are still described here in full for completeness. (Remember that any external coprocessor described in this section is a software emulation.)

<table>
<thead>
<tr>
<th>31</th>
<th>28 27 24 23</th>
<th>20 19</th>
<th>16 15</th>
<th>12 11</th>
<th>8 7</th>
<th>5 4 3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cond</td>
<td>CP Opc</td>
<td>CRn</td>
<td>CRd</td>
<td>Cpf</td>
<td>Cp</td>
<td>0</td>
<td>CRm</td>
</tr>
</tbody>
</table>

[3:0] Coprocessor operand register

[7:5] Coprocessor information

[11:8] Coprocessor number

[15:12] Coprocessor destination register

[19:16] Coprocessor operand register

[23:20] Coprocessor operation code

[31:28] Condition Field

Figure 3-25. Coprocessor Data Operation Instruction

Only bit 4 and bits 24 to 31 are significant to ARM920T. The remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields except CP# as appropriate. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.
INSTRUCTION CYCLE TIMES

Coprocessor data operations take 1S + bI incremental cycles to execute, where b is the number of cycles spent in the coprocessor busy-wait loop.

S and I are defined as sequential (S-cycle) and internal (I-cycle).

Assembler syntax

CDP{cond} p#,{<expression1>},cd,cn,cm{,<expression2>}

{cond} Two character condition mnemonic. See Table 3-2.
p# The unique number of the required coprocessor
<expression1> Evaluated to a constant and placed in the CP Opc field
cd, cn and cm Evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively
<expression2> Where present is evaluated to a constant and placed in the CP field

EXAMPLES

CDP       p1,10,c1,c2,c3 ; Request coproc 1 to do operation 10
       ; on CR2 and CR3, and put the result in CR1.
CDPEQ     p2,5,c1,c2,c3,2 ; If Z flag is set request coproc 2 to do operation 5 (type 2)
       ; on CR2 and CR3, and put the result in CR1.
COPROCESSOR DATA TRANSFERS (LDC, STC)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-26.

This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessor's registers directly to memory. ARM920T is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.

| 31 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 16 | 15 | 12 | 11 | 8  | 7  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Cond | 110 | P | U | N | W | L | Rn | CRd | CP# | Offset |

- [7:0] Unsigned 8 Bit Immediate Offset
- [11:8] Coprocessor Number
- [15:12] Coprocessor Source/Destination Register
- [19:16] Base Register
- [20] Load/Store Bit
  - 0 = Store to memory
  - 1 = Load from memory
- [21] Write-back Bit
  - 0 = No write-back
  - 1 = Write address into base
- [22] Transfer Length
- [23] Up/Down Bit
  - 0 = Down: subtract offset from base
  - 1 = Up: add offset to base
- [24] Pre/Post Indexing Bit
  - 0 = Post: add offset after transfer
  - 1 = Pre: add offset before transfer
- [31:28] Condition Field

Figure 3-26. Coprocessor Data Transfer Instructions
THE COPROCESSOR FIELDS

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will only respond if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be transferred), and the N bit is used to choose one of two transfer length options. For instance N=0 could select the transfer of a single register, and N=1 could select the transfer of all the registers for context switching.

ADDRESSING MODES

ARM920T is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that the immediate offsets are 8 bits wide and specify word offsets for coprocessor data transfers, whereas they are 12 bits wide and specify byte offsets for single data transfers.

The 8 bit unsigned immediate offset is shifted left 2 bits and either added to (U=1) or subtracted from (U=0) the base register (Rn); this calculation may be performed either before (P=1) or after (P=0) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if W=1), or the old value of the base may be preserved (W=0). Note that post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR which always write-back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

ADDRESS ALIGNMENT

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on A[1:0] and might be interpreted by the memory system.

Use of R15

If Rn is R15, the value used will be the address of the instruction plus 8 bytes. Base write-back to R15 must not be specified.

DATA ABORTS

If the address is legal but the memory manager generates an abort, the data trap will be taken. The write-back of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort has been resolved, and must ensure that any subsequent actions it undertakes can be repeated when the instruction is retried.

Instruction cycle times

Coprocessor data transfer instructions take \((n-1)S + 2N + bI\) incremental cycles to execute, where:

- **n** The number of words transferred.
- **b** The number of cycles spent in the coprocessor busy-wait loop.

S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively.
ASSEMBLER SYNTAX

<LDC|STC>{cond}{L} p#,cd,<Address>

LDC          Load from memory to coprocessor
STC          Store from coprocessor to memory
{L}          When present perform long transfer (N=1), otherwise perform short transfer (N=0)
{cond}       Two character condition mnemonic. See Table 3-2..
p#           The unique number of the required coprocessor
cd           An expression evaluating to a valid coprocessor register number that is placed in the CRd field

<Address> can be:

1. An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2. A pre-indexed addressing specification:
   [Rn]          offset of zero
   [Rn,#expression]{!} offset of <expression> bytes

3. A post-indexed addressing specification:
   [Rn],<#expression> offset of <expression> bytes
   {!}           write back the base register (set the W bit) if! is present
   Rn            is an expression evaluating to a valid ARM920T register number.

NOTES

If Rn is R15, the assembler will subtract 8 from the offset value to allow for ARM920T pipelining.

EXAMPLES

<table>
<thead>
<tr>
<th>_instruction</th>
<th></th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDC</td>
<td>p1,c2,table</td>
<td>Load c2 of coproc 1 from address table, using a PC relative address.</td>
</tr>
<tr>
<td>STCEQL</td>
<td>p2,c3,[R5,#24]!</td>
<td>Conditionally store c3 of coproc 2 into an address 24 bytes up from R5, write this address back to R5, and use long transfer option (probably to store multiple words).</td>
</tr>
</tbody>
</table>

NOTES

Although the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.
COPROCESSOR REGISTER TRANSFERS (MRC, MCR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.. The instruction encoding is shown in Figure 3-27.

This class of instruction is used to communicate information directly between ARM920T and a coprocessor. An example of a coprocessor to ARM920T register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32 bit integer within the coprocessor, and the result is then transferred to ARM920T register. A FLOAT of a 32 bit value in ARM920T register into a floating point value within the coprocessor illustrates the use of ARM920T register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the ARM920T CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

### Figure 3-27. Coprocessor Register Transfer Instructions

<table>
<thead>
<tr>
<th>31</th>
<th>28 27</th>
<th>24 23</th>
<th>21 20 19</th>
<th>16 15</th>
<th>12 11</th>
<th>8 7</th>
<th>5 4 3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cond</td>
<td>1110</td>
<td>CP Opc</td>
<td>L</td>
<td>CRn</td>
<td>Rd</td>
<td>CP#</td>
<td>CP</td>
<td>1</td>
</tr>
</tbody>
</table>

- **[3:0]** Coprocessor Operand Register
- **[7:5]** Coprocessor Information
- **[11:8]** Coprocessor Number
- **[15:12]** ARM Source/Destination Register
- **[19:16]** Coprocessor Source/Destination Register
- **[20]** Load/Store Bit
  - 0 = Store to coprocessor
  - 1 = Load from coprocessor
- **[21]** Coprocessor Operation Mode
- **[31:28]** Condition Field

THE COPROCESSOR FIELDS

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in some way which depends on the particular operation specified.
TRANSFERS TO R15

When a coprocessor register transfer to ARM920T has R15 as the destination, bits 31, 30, 29 and 28 of the transferred word are copied into the N, Z, C and V flags respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

TRANSFERS FROM R15

A coprocessor register transfer from ARM920T with R15 as the source register will store the PC+12.

INSTRUCTION CYCLE TIMES

MRC instructions take 1S + (b+1)I +1C incremental cycles to execute, where S, I and C are defined as sequential (S-cycle), internal (I-cycle), and coprocessor register transfer (C-cycle), respectively. MCR instructions take 1S + bI +1C incremental cycles to execute, where b is the number of cycles spent in the coprocessor busy-wait loop.

ASSEMBLER SYNTAX

\(<\text{MCR}|\text{MRC}\rangle\{\text{cond}\} \ p\#,.<\text{expression1}>,Rd,cn,cm\{,.<\text{expression2}>)\)

- MRC: Move from coprocessor to ARM920T register (L=1)
- MCR: Move from ARM920T register to coprocessor (L=0)
- \{\text{cond}\}: Two character condition mnemonic. See Table 3-2
- \ p\#: The unique number of the required coprocessor
- \ <\text{expression1}>: Evaluated to a constant and placed in the CP Opc field
- \ Rd\: An expression evaluating to a valid ARM920T register number
- \ cn\ and \ cm\: Expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively
- \ <\text{expression2}>: Where present is evaluated to a constant and placed in the CP field

EXAMPLES

- \text{MRC} \ p2,5,R3,c5,c6 \; \text{; Request coproc 2 to perform operation 5}
  \; \text{on c5 and c6, and transfer the (single}
  \; \text{32-bit word) result back to R3.}
- \text{MCR} \ p6,0,R4,c5,c6 \; \text{; Request coproc 6 to perform operation 0}
  \; \text{on R4 and place the result in c6.}
- \text{MRCEQ} \ p3,9,R3,c5,c6,2 \; \text{; Conditionally request coproc 3 to}
  \; \text{perform operation 9 (type 2) on c5 and}
  \; \text{c6, and transfer the result back to R3.}
UNDEFINED INSTRUCTION

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction format is shown in Figure 3-28.

![Figure 3-28. Undefined Instruction](image)

If the condition is true, the undefined instruction trap will be taken.

Note that the undefined instruction mechanism involves offering this instruction to any coprocessors which may be present, and all coprocessors must refuse to accept it by driving CPA and CPB HIGH.

INSTRUCTION CYCLE TIMES

This instruction takes $2S + 1I + 1N$ cycles, where $S$, $N$ and $I$ are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle).

ASSEMBLER SYNTAX

The assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until such time, this instruction must not be used.
INSTRUCTION SET EXAMPLES

The following examples show ways in which the basic ARM920T instructions can combine to give efficient code. None of these methods saves a great deal of execution time (although they may save some), mostly they just save code.

USING THE CONDITIONAL INSTRUCTIONS

Using Conditionals for Logical OR

\[
\begin{align*}
\text{CMP} & \quad Rn,#p \quad ; \quad \text{If } Rn=p \text{ OR } Rm=q \text{ THEN GOTO Label.} \\
\text{BEQ} & \quad \text{Label} \\
\text{CMP} & \quad Rm,#q \\
\text{BEQ} & \quad \text{Label}
\end{align*}
\]

This can be replaced by

\[
\begin{align*}
\text{CMP} & \quad Rn,#p \quad ; \quad \text{If condition not satisfied try other test.} \\
\text{CMPNE} & \quad Rm,#q \\
\text{BEQ} & \quad \text{Label}
\end{align*}
\]

Absolute Value

\[
\begin{align*}
\text{TEQ} & \quad Rn,#0 \quad ; \quad \text{Test sign} \\
\text{RSBMI} & \quad Rn,Rn,#0 \quad ; \quad \text{and 2's complement if necessary.}
\end{align*}
\]

Multiplication by 4, 5 or 6 (Run Time)

\[
\begin{align*}
\text{MOV} & \quad Rc,Ra,LSL#2 \quad ; \quad \text{Multiply by 4,} \\
\text{CMP} & \quad Rb,#5 \quad ; \quad \text{Test value,} \\
\text{ADDCS} & \quad Rc,Rc,Ra \quad ; \quad \text{Complete multiply by 5,} \\
\text{ADDHI} & \quad Rc,Rc,Ra \quad ; \quad \text{Complete multiply by 6.}
\end{align*}
\]

Combining Discrete and Range Tests

\[
\begin{align*}
\text{TEQ} & \quad Rc,#127 \quad ; \quad \text{Discrete test,} \\
\text{CMPNE} & \quad Rc,\"\",-1 \quad ; \quad \text{Range test} \\
\text{MOVLS} & \quad Rc,\"\" \quad ; \quad \text{IF } Rc=\"\" \text{ OR } Rc=\text{ASCII}(127) \\
& \quad \text{THEN } Rc=\"\" \\
\end{align*}
\]

Division and Remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM Cross Development Toolkit, available from your supplier. A short general purpose divide routine follows.

```asm
; Enter with numbers in Ra and Rb.
MOV Rcnt,#1 ; Bit to control the division.

Div1
CMP Rb,#0x80000000 ; Move Rb until greater than Ra.
CMPCC Rb,Ra
MOVCC Rb,Rb,ASL#1
MOVCC Rcnt,Rcnt,ASL#1
BCC Div1
MOV Rc,#0

Div2
CMP Ra,Rb ; Test for possible subtraction.
SUBCS Ra,Ra,Rb ; Subtract if ok,
ADDCS Rc,Rc,Rcnt ; Put relevant bit into result
MOVSS Rcnt,Rcnt,LSR#1 ; Shift control bit
MOVNE Rb,Rb,LSR#1 ; Halve unless finished.
BNE Div2 ; Divide result in Rc, remainder in Ra.
```

Overflow Detection in the ARM920T

1. Overflow in unsigned multiply with a 32-bit result

```asm
UMULL Rd,Rt,Rm,Rn ; 3 to 6 cycles
TEQ Rt,#0 ; +1 cycle and a register
BNE overflow
```

2. Overflow in signed multiply with a 32-bit result

```asm
SMULL Rd,Rt,Rm,Rn ; 3 to 6 cycles
TEQ Rt,Rd, ASR#31 ; +1 cycle and a register
BNE overflow
```

3. Overflow in unsigned multiply accumulate with a 32 bit result

```asm
UMLAL Rd,Rt,Rm,Rn ; 4 to 7 cycles
TEQ Rt,#0 ; +1 cycle and a register
BNE overflow
```

4. Overflow in signed multiply accumulate with a 32 bit result

```asm
SMLAL Rd,Rt,Rm,Rn ; 4 to 7 cycles
TEQ Rt,Rd, ASR#31 ; +1 cycle and a register
BNE overflow
```
5. Overflow in unsigned multiply accumulate with a 64 bit result

UMULL  RL,Rh,Rm,Rn  ; 3 to 6 cycles
ADDs   RL,Ri,Ra1    ; Lower accumulate
ADC    Rh,Rh,Ra2    ; Upper accumulate
BCS    overflow     ; 1 cycle and 2 registers

6. Overflow in signed multiply accumulate with a 64 bit result

SMULL RL,Rh,Rm,Rn  ; 3 to 6 cycles
ADDs   RL,Ri,Ra1    ; Lower accumulate
ADC    Rh,Rh,Ra2    ; Upper accumulate
BVS    overflow     ; 1 cycle and 2 registers

NOTES

Overflow checking is not applicable to unsigned and signed multiplies with a 64-bit result, since overflow does not occur in such calculations.

PSEUDO-RANDOM BINARY SEQUENCE GENERATOR

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32 bit generator needs more than one feedback tap to be maximal length (i.e. $2^{32} - 1$ cycles before repetition), so this example uses a 33 bit register with taps at bits 33 and 20. The basic algorithm is

newbit := bit 33 eor bit 20, shift left the 33 bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (i.e. 32 bits). The entire operation can be done in 5 S cycles:

; Enter with seed in Ra (32 bits),
; Rb (1 bit in Rb lsb), uses Rc.
TST   Rb,Rb,LSR#1   ; Top bit into carry
MOVs  Rc,Ra,RRX    ; 33 bit rotate right
ADC   Rb,Rb,Rb     ; Carry into lsb of Rb
EOR   Rc,Rc,Ra,LSR#12 ; (involved!)
EOR   Ra,Rc,Rc,LSR#20 ; (similarly involved!) new seed in Ra, Rb as before

MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER

Multiplication by $2^n$ ($1,2,4,8,16,32,..$)

MOV   Ra, Rb, LSL #n

Multiplication by $2^n+1$ ($3,5,9,17,..$)

ADD   Ra,Ra,Ra,LSL #n

Multiplication by $2^n-1$ ($3,7,15,..$)

RSB   Ra,Ra,Ra,LSL #n
Multiplication by 6

- \text{ADD} \quad \text{Ra,Ra,Ra,LSL } #1 \quad ; \quad \text{Multiply by 3}
- \text{MOV} \quad \text{Ra,Ra,LSL} #1 \quad ; \quad \text{and then by 2}

Multiply by 10 and add in extra number

- \text{ADD} \quad \text{Ra,Ra,Ra,LSL} #2 \quad ; \quad \text{Multiply by 5}
- \text{ADD} \quad \text{Ra,Rc,Ra,LSL} #1 \quad ; \quad \text{Multiply by 2 and add in next digit}

General recursive method for Rb := Ra*C, C a constant:

1. If C even, say C = 2^n*D, D odd:
   - D=1: \quad \text{MOV} \quad \text{Rb,Ra,LSL } #n
   - D<>1: \{Rb := Ra*D\}
   - \text{MOV} \quad \text{Rb,Rb,LSL } #n

2. If C MOD 4 = 1, say C = 2^n*D+1, D odd, n>1:
   - D=1: \quad \text{ADD} \quad \text{Rb,Ra,Ra,LSL } #n
   - D<>1: \{Rb := Ra*D\}
   - \text{ADD} \quad \text{Rb,Ra,Rb,LSL } #n

3. If C MOD 4 = 3, say C = 2^n*D-1, D odd, n>1:
   - D=1: \quad \text{RSB} \quad \text{Rb,Ra,Ra,LSL } #n
   - D<>1: \{Rb := Ra*D\}
   - \text{RSB} \quad \text{Rb,Ra,Rb,LSL } #n

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

- \text{RSB} \quad \text{Rb,Ra,Ra,LSL} #2 \quad ; \quad \text{Multiply by 3}
- \text{RSB} \quad \text{Rb,Ra,Rb,LSL} #2 \quad ; \quad \text{Multiply by 4*3-1 = 11}
- \text{ADD} \quad \text{Rb,Ra,Rb,LSL# 2} \quad ; \quad \text{Multiply by 4*11+1 = 45}

rather than by:

- \text{ADD} \quad \text{Rb,Ra,Ra,LSL} #3 \quad ; \quad \text{Multiply by 9}
- \text{ADD} \quad \text{Rb,Rb,Rb,LSL} #2 \quad ; \quad \text{Multiply by 5*9 = 45}
LOADING A WORD FROM AN UNKNOWN ALIGNMENT

; Enter with address in Ra (32 bits) uses
; Rb, Rc result in Rd. Note d must be less than c e.g. 0,1

BIC Rb,Ra,#3 ; Get word aligned address
LDMIA Rb,{Rd,Rc} ; Get 64 bits containing answer
AND Rb,Ra,#3 ; Correction factor in bytes
MOVNE Rd,Rb,LSL#3 ; ...now in bits and test if aligned
MOVNE Rd,Rd,LSR Rb ; Produce bottom of result word (if not aligned)
RSBNE Rd,Rd,#32 ; Get other shift amount
ORRNE Rd,Rd,Rc,LSL Rb ; Combine two halves to get result
THUMB INSTRUCTION SET FORMAT

The thumb instruction sets are 16-bit versions of ARM instruction sets (32-bit format). The ARM instructions are reduced to 16-bit versions, Thumb instructions, at the cost of versatile functions of the ARM instruction sets. The thumb instructions are decompressed to the ARM instructions by the Thumb decompressor inside the ARM920T core.

As the Thumb instructions are compressed ARM instructions, the Thumb instructions have the 16-bit format instructions and have some restrictions. The restrictions by 16-bit format is fully notified for using the Thumb instructions.
FORMAT SUMMARY

The THUMB instruction set formats are shown in the following figure.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 Op Offset5 Rs Rd</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 1 1 I Op Rn/offset3 Rs Rd</td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 Op Rd Offset8</td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 0 0 0 Op Rs Rd</td>
</tr>
<tr>
<td>5</td>
<td>0 1 0 0 0 1 Op H1 H2 Rs/Hs Rd/Hd</td>
</tr>
<tr>
<td>6</td>
<td>0 1 0 0 1 Rd Word8</td>
</tr>
<tr>
<td>7</td>
<td>0 1 0 1 L B 0 Ro Rb Rd</td>
</tr>
<tr>
<td>8</td>
<td>0 1 0 1 H S 1 Ro Rb Rd</td>
</tr>
<tr>
<td>9</td>
<td>0 1 1 B L Offset5 Rb Rd</td>
</tr>
<tr>
<td>10</td>
<td>1 0 0 0 L Offset5 Rb Rd</td>
</tr>
<tr>
<td>11</td>
<td>1 0 0 1 L Rd Word8</td>
</tr>
<tr>
<td>12</td>
<td>1 0 1 0 SP Rd Word8</td>
</tr>
<tr>
<td>13</td>
<td>1 0 1 1 0 0 0 0 S SWord7</td>
</tr>
<tr>
<td>14</td>
<td>1 0 1 1 L 1 0 R Rlist</td>
</tr>
<tr>
<td>15</td>
<td>1 1 0 0 L Rb Rlist</td>
</tr>
<tr>
<td>16</td>
<td>1 1 0 1 Cond Softset8</td>
</tr>
<tr>
<td>17</td>
<td>1 1 0 1 1 1 1 1 Value8</td>
</tr>
<tr>
<td>18</td>
<td>1 1 1 0 0 Offset11</td>
</tr>
<tr>
<td>19</td>
<td>1 1 1 1 H Offset</td>
</tr>
</tbody>
</table>

Move Shifted register
Add/subtract
Move/compare/add/subtract immediate
ALU operations
Hi register operations /branch exchange
PC-relative load
Load/store with register offset
Load/store sign-extended byte/halfword
Load/store with immediate offset
Load/store halfword
SP-relative load/store
Load address
Add offset to stack pointer
Push/pop register
Multiple load/store
Conditional branch
Software interrupt
Unconditional branch
Long branch with link

Figure 4-1. THUMB Instruction Set Formats
**OPCODE SUMMARY**

The following table summarizes the THUMB instruction set. For further information about a particular instruction please refer to the sections listed in the right-most column.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Lo-Register Operand</th>
<th>Hi-Register Operand</th>
<th>Condition Codes Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Add with Carry</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>ADD</td>
<td>Add</td>
<td>Y</td>
<td>–</td>
<td>Y (1)</td>
</tr>
<tr>
<td>AND</td>
<td>AND</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>ASR</td>
<td>Arithmetic Shift Right</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>Unconditional branch</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Bxx</td>
<td>Conditional branch</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>BIC</td>
<td>Bit Clear</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>BL</td>
<td>Branch and Link</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>BX</td>
<td>Branch and Exchange</td>
<td>Y</td>
<td>Y</td>
<td>–</td>
</tr>
<tr>
<td>CMN</td>
<td>Compare Negative</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>CMP</td>
<td>Compare</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>EOR</td>
<td>EOR</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>LDMIA</td>
<td>Load multiple</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LDR</td>
<td>Load word</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LDRB</td>
<td>Load byte</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LDRH</td>
<td>Load halfword</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LSL</td>
<td>Logical Shift Left</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>LDSB</td>
<td>Load sign-extended byte</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LDSH</td>
<td>Load sign-extended halfword</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LSR</td>
<td>Logical Shift Right</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>MOV</td>
<td>Move register</td>
<td>Y</td>
<td>Y</td>
<td>Y (2)</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiply</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>MVN</td>
<td>Move Negative register</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
</tbody>
</table>
### Table 4-1. THUMB Instruction Set Opcodes (Continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Lo-Register Operand</th>
<th>Hi-Register Operand</th>
<th>Condition Codes Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG</td>
<td>Negate</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>ORR</td>
<td>OR</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>POP</td>
<td>Pop register</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PUSH</td>
<td>Push register</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ROR</td>
<td>Rotate Right</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>SBC</td>
<td>Subtract with Carry</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>STMIA</td>
<td>Store Multiple</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>STR</td>
<td>Store word</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>STRB</td>
<td>Store byte</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>STRH</td>
<td>Store halfword</td>
<td>Y</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SWI</td>
<td>Software Interrupt</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
<tr>
<td>TST</td>
<td>Test bits</td>
<td>Y</td>
<td>–</td>
<td>Y</td>
</tr>
</tbody>
</table>

**NOTES:**
1. The condition codes are unaffected by the format 5, 12 and 13 versions of this instruction.
2. The condition codes are unaffected by the format 5 version of this instruction.
FORMAT 1: MOVE SHIFTED REGISTER

[2:0] Destination Register
[5:3] Source Register
[10:6] Immediate Value
[12:11] Opcode
  0 = LSL
  1 = LSR
  2 = ASR

Figure 4-2. Format 1

OPERATION
These instructions move a shifted value between Lo registers. The THUMB assembler syntax is shown in Table 4-2.

NOTE
All instructions in this group set the CPSR condition codes.

Table 4-2. Summary of Format 1 Instructions

<table>
<thead>
<tr>
<th>OP</th>
<th>THUMB Assembler</th>
<th>ARM Equipment</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>LSL Rd, Rs, #Offset5</td>
<td>MOVs Rd, Rs, LSL #Offset5</td>
<td>Shift Rs left by a 5-bit immediate value and store the result in Rd.</td>
</tr>
<tr>
<td>01</td>
<td>LSR Rd, Rs, #Offset5</td>
<td>MOVs Rd, Rs, LSR #Offset5</td>
<td>Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.</td>
</tr>
<tr>
<td>10</td>
<td>ASR Rd, Rs, #Offset5</td>
<td>MOVs Rd, Rs, ASR #Offset5</td>
<td>Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.</td>
</tr>
</tbody>
</table>
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-2. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

LSR R2, R5, #27 ; Logical shift right the contents
; of R5 by 27 and store the result in R2.
; Set condition codes on the result.
FORMAT 2: ADD/SUBTRACT

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Op</td>
<td>Rn/Offset3</td>
<td>Rs</td>
<td>Rd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[2:0] Destination Register
[5:3] Source Register
[8:6] Register/Immediate Value

[9] Opcode
0 = ADD
1 = SUB

[10] Immediate Flag
0 = Register operand
1 = Immediate operand

Figure 4-3. Format 2

OPERATION

These instructions allow the contents of a Lo register or a 3-bit immediate value to be added to or subtracted from a Lo register. The THUMB assembler syntax is shown in Table 4-3.

NOTE

All instructions in this group set the CPSR condition codes.

Table 4-3. Summary of Format 2 Instructions

<table>
<thead>
<tr>
<th>OP</th>
<th>I</th>
<th>THUMB Assembler</th>
<th>ARM Equipment</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ADD Rd, Rs, Rn</td>
<td>ADDS Rd, Rs, Rn</td>
<td>Add contents of Rn to contents of Rs. Place result in Rd.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADD Rd, Rs, #Offset3</td>
<td>ADDS Rd, Rs, #Offset3</td>
<td>Add 3-bit immediate value to contents of Rs. Place result in Rd.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>SUB Rd, Rs, Rn</td>
<td>SUBS Rd, Rs, Rn</td>
<td>Subtract contents of Rn from contents of Rs. Place result in Rd.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>SUB Rd, Rs, #Offset3</td>
<td>SUBS Rd, Rs, #Offset3</td>
<td>Subtract 3-bit immediate value from contents of Rs. Place result in Rd.</td>
</tr>
</tbody>
</table>
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-3. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

<table>
<thead>
<tr>
<th>Command</th>
<th>Registers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>R0, R3, R4</td>
<td>( R0 := R3 + R4 ) and set condition codes on the result.</td>
</tr>
<tr>
<td>SUB</td>
<td>R6, R2, #6</td>
<td>( R6 := R2 - 6 ) and set condition codes.</td>
</tr>
</tbody>
</table>
FORMAT 3: MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE

![Figure 4-4. Format 3](image)

O P E R A T I O N S

The instructions in this group perform operations between a Lo register and an 8-bit immediate value. The THUMB assembler syntax is shown in Table 4-4.

**NOTE**

All instructions in this group set the CPSR condition codes.

### Table 4-4. Summary of Format 3 Instructions

<table>
<thead>
<tr>
<th>OP</th>
<th>THUMB Assembler</th>
<th>ARM Equipment</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>MOV Rd, #Offset8</td>
<td>MOVS Rd, #Offset8</td>
<td>Move 8-bit immediate value into Rd.</td>
</tr>
<tr>
<td>01</td>
<td>CMP Rd, #Offset8</td>
<td>CMP Rd, #Offset8</td>
<td>Compare contents of Rd with 8-bit immediate value.</td>
</tr>
<tr>
<td>10</td>
<td>ADD Rd, #Offset8</td>
<td>ADDS Rd, Rd, #Offset8</td>
<td>Add 8-bit immediate value to contents of Rd and place the result in Rd.</td>
</tr>
<tr>
<td>11</td>
<td>SUB Rd, #Offset8</td>
<td>SUBS Rd, Rd, #Offset8</td>
<td>Subtract 8-bit immediate value from contents of Rd and place the result in Rd.</td>
</tr>
</tbody>
</table>
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-4. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

| MOV     | R0, #128          | R0 := 128 and set condition codes |
| CMP     | R2, #62           | Set condition codes on R2 - 62    |
| ADD     | R1, #255          | R1 := R1 + 255 and set condition codes |
| SUB     | R6, #145          | R6 := R6 - 145 and set condition codes |
FORMAT 4: ALU OPERATIONS

![Figure 4-5. Format 4](image)

OPERATION

The following instructions perform ALU operations on a Lo register pair.

NOTE

All instructions in this group set the CPSR condition codes.

Table 4-5. Summary of Format 4 Instructions

<table>
<thead>
<tr>
<th>OP</th>
<th>THUMB Assembler</th>
<th>ARM Equipment</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>AND Rd, Rs</td>
<td>ANDS Rd, Rd, Rs</td>
<td>Rd := Rd AND Rs</td>
</tr>
<tr>
<td>0001</td>
<td>EOR Rd, Rs</td>
<td>EORS Rd, Rd, Rs</td>
<td>Rd := Rd EOR Rs</td>
</tr>
<tr>
<td>0010</td>
<td>LSL Rd, Rs</td>
<td>MOVSrds Rd, Rd, LSL Rs</td>
<td>Rd := Rd &lt;&lt; Rs</td>
</tr>
<tr>
<td>0011</td>
<td>LSR Rd, Rs</td>
<td>MOVs Rd, Rd, LSR Rs</td>
<td>Rd := Rd &gt;&gt; Rs</td>
</tr>
<tr>
<td>0100</td>
<td>ASR Rd, Rs</td>
<td>MOVAs Rd, Rd, ASR Rs</td>
<td>Rd := Rd ASR Rs</td>
</tr>
<tr>
<td>0101</td>
<td>ADC Rd, Rs</td>
<td>ADCSrds Rd, Rd, Rs</td>
<td>Rd := Rd + Rs + C-bit</td>
</tr>
<tr>
<td>0110</td>
<td>SBC Rd, Rs</td>
<td>SBCSrds Rd, Rd, Rs</td>
<td>Rd := Rd - Rs - NOT C-bit</td>
</tr>
<tr>
<td>0111</td>
<td>ROR Rd, Rs</td>
<td>MOVS Rd, Rd, ROR Rs</td>
<td>Rd := Rd ROR Rs</td>
</tr>
<tr>
<td>1000</td>
<td>TST Rd, Rs</td>
<td>TST Rd, Rs</td>
<td>Set condition codes on Rd AND Rs</td>
</tr>
<tr>
<td>1001</td>
<td>NEG Rd, Rs</td>
<td>RSBS Rd, Rs, #0</td>
<td>Rd = - Rs</td>
</tr>
<tr>
<td>1010</td>
<td>CMP Rd, Rs</td>
<td>CMPrd, Rs</td>
<td>Set condition codes on Rd - Rs</td>
</tr>
<tr>
<td>1011</td>
<td>CMN Rd, Rs</td>
<td>CMN Rd, Rs</td>
<td>Set condition codes on Rd + Rs</td>
</tr>
<tr>
<td>1100</td>
<td>ORR Rd, Rs</td>
<td>ORRsrds Rd, Rd, Rs</td>
<td>Rd := Rd OR Rs</td>
</tr>
<tr>
<td>1101</td>
<td>MUL Rd, Rs</td>
<td>MULsrds Rd, Rs, Rd</td>
<td>Rd := Rs * Rd</td>
</tr>
<tr>
<td>1110</td>
<td>BIC Rd, Rs</td>
<td>BICS Rd, Rd, Rs</td>
<td>Rd := Rd AND NOT Rs</td>
</tr>
<tr>
<td>1111</td>
<td>MVN Rd, Rs</td>
<td>MVNS Rd, Rs</td>
<td>Rd := NOT Rs</td>
</tr>
</tbody>
</table>
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-5. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Registers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOR</td>
<td>R3, R4</td>
<td>R3 := R3 EOR R4 and set condition codes</td>
</tr>
<tr>
<td>ROR</td>
<td>R1, R0</td>
<td>Rotate Right R1 by the value in R0, store the result in R1 and set condition codes</td>
</tr>
<tr>
<td>NEG</td>
<td>R5, R3</td>
<td>Subtract the contents of R3 from zero, store the result in R5. Set condition codes i.e. R5 = - R3</td>
</tr>
<tr>
<td>CMP</td>
<td>R2, R6</td>
<td>Set the condition codes on the result of R2 - R6</td>
</tr>
<tr>
<td>MUL</td>
<td>R0, R7</td>
<td>R0 := R7 * R0 and set condition codes</td>
</tr>
</tbody>
</table>
FORMAT 5: HI-REGISTER OPERATIONS/BRANCH EXCHANGE

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Op</td>
<td>H1</td>
<td>H2</td>
<td>Rs/Hs</td>
<td>Rd/Hd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[2:0] Destination Register
[5:3] Source Register
[6] Hi Operand Flag 2
[7] Hi Operand Flag 1
[9:8] Opcode

Figure 4-6. Format 5

OPERATION

There are four sets of instructions in this group. The first three allow ADD, CMP and MOV operations to be performed between Lo and Hi registers, or a pair of Hi registers. The fourth, BX, allows a Branch to be performed which may also be used to switch processor state. The THUMB assembler syntax is shown in Table 4-6.

NOTES

In this group only CMP (Op = 01) sets the CPSR condition codes.

The action of H1 = 0, H2 = 0 for Op = 00 (ADD), Op = 01 (CMP) and Op = 10 (MOV) is undefined, and should not be used.

Table 4-6. Summary of Format 5 Instructions

<table>
<thead>
<tr>
<th>Op</th>
<th>H1</th>
<th>H2</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>ADD Rd, Rs</td>
<td>ADD Rd, Rd, Rs</td>
<td>Add a register in the range 8-15 to a register in the range 0-7.</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>ADD Hd, Rs</td>
<td>ADD Hd, Hd, Rs</td>
<td>Add a register in the range 0-7 to a register in the range 8-15.</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>1</td>
<td>ADD Hd, Rs</td>
<td>ADD Hd, Hd, Rs</td>
<td>Add two registers in the range 8-15</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>CMP Rd, Rs</td>
<td>CMP Rd, Rs</td>
<td>Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>CMP Hd, Rs</td>
<td>CMP Hd, Rs</td>
<td>Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.</td>
</tr>
</tbody>
</table>
Table 4-6. Summary of Format 5 Instructions (Continued)

<table>
<thead>
<tr>
<th>Op</th>
<th>H1</th>
<th>H2</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>CMP Hd, Hs</td>
<td>CMP Hd, Hs</td>
<td>Compare two registers in the range 8-15. Set the condition code flags on the result.</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>MOV Rd, Hs</td>
<td>MOV Rd, Hs</td>
<td>Move a value from a register in the range 8-15 to a register in the range 0-7.</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>MOV Hd, Rs</td>
<td>MOV Hd, Rs</td>
<td>Move a value from a register in the range 0-7 to a register in the range 8-15.</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>MOV Hd, Hs</td>
<td>MOV Hd, Hs</td>
<td>Move a value between two registers in the range 8-15.</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>BX Rs</td>
<td>BX Rs</td>
<td>Perform branch (plus optional state change) to address in a register in the range 0-7.</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>BX Hs</td>
<td>BX Hs</td>
<td>Perform branch (plus optional state change) to address in a register in the range 8-15.</td>
</tr>
</tbody>
</table>

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-6. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

THE BX INSTRUCTION

BX performs a Branch to a routine whose start address is specified in a Lo or Hi register.

Bit 0 of the address determines the processor state on entry to the routine:

- Bit 0 = 0 Causes the processor to enter ARM state.
- Bit 0 = 1 Causes the processor to enter THUMB state.

NOTE

The action of H1 = 1 for this instruction is undefined, and should not be used.
EXAMPLES

Hi-Register Operations

```
ADD    PC, R5          ; PC := PC + R5 but don't set the condition codes.
CMP    R4, R12         ; Set the condition codes on the result of R4 - R12.
MOV    R15, R14        ; Move R14 (LR) into R15 (PC)
                   ; but don't set the condition codes,
                   ; eg. return from subroutine.
```

Branch and Exchange

```
ADR    R1, outofTHUMB  ; Switch from THUMB to ARM state.
MOV    R11, R1         ; Load address of outofTHUMB into R1.
BX     R11             ; Transfer the contents of R11 into the PC.
                   ; Bit 0 of R11 determines whether
                   ; ARM or THUMB state is entered, ie. ARM state here.
```

```
ALIGN
CODE32
outofTHUMB  ; Now processing ARM instructions...
```

USING R15 AS AN OPERAND

If R15 is used as an operand, the value will be the address of the instruction + 4 with bit 0 cleared. Executing a BX PC in THUMB state from a non-word aligned address will result in unpredictable execution.
FORMAT 6: PC-RELATIVE LOAD

![Format 6 Diagram]

[7:0] Immediate Value
[10:8] Destination Register

Figure 4-7. Format 6

OPERATION

This instruction loads a word from an address specified as a 10-bit immediate offset from the PC. The THUMB assembler syntax is shown below.

Table 4-7. Summary of PC-Relative Load Instruction

<table>
<thead>
<tr>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR Rd, [PC, #Imm]</td>
<td>LDR Rd, [R15, #Imm]</td>
<td>Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.</td>
</tr>
</tbody>
</table>

NOTE: The value specified by #Imm is a full 10-bit address, but must always be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in field Word 8. The value of the PC will be 4 bytes greater than the address of this instruction, but bit 1 of the PC is forced to 0 to ensure it is word aligned.
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

LDR R3,[PC,#844] ; Load into R3 the word found at the address formed by adding 844 to PC. ; bit[1] of PC is forced to zero. ; Note that the THUMB opcode will contain 211 as the Word8 value.
FORMAT 7: LOAD/STORE WITH REGISTER OFFSET

Figure 4-8. Format 7
OPERATION

These instructions transfer byte or word values between registers and memory. Memory addresses are pre-indexed using an offset register in the range 0-7. The THUMB assembler syntax is shown in Table 4-8.

Table 4-8. Summary of Format 7 Instructions

<table>
<thead>
<tr>
<th>L</th>
<th>B</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>STR Rd, [Rb, Ro]</td>
<td>STR Rd, [Rb, Ro]</td>
<td>Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>STRB Rd, [Rb, Ro]</td>
<td>STRB Rd, [Rb, Ro]</td>
<td>Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>LDR Rd, [Rb, Ro]</td>
<td>LDR Rd, [Rb, Ro]</td>
<td>Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>LDRB Rd, [Rb, Ro]</td>
<td>LDRB Rd, [Rb, Ro]</td>
<td>Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.</td>
</tr>
</tbody>
</table>

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-8. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STR R3, [R2,R6] ; Store word in R3 at the address formed by adding R6 to R2.
LDRB R2, [R0,R7] ; Load into R2 the byte found at the address formed by adding R7 to R0.
```
FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD

![Format 8 Diagram](image)

**[2:0] Destination Register**

**[5:3] Base Register**

**[8:6] Offset Register**

**[10] Sign-Extended Flag**

0 = Operand not sing-extended
1 = Operand sing-extended

**[11] H Flag**

Figure 4-9. Format 8

**OPERATION**

These instructions load optionally sign-extended bytes or halfwords, and store halfwords. The THUMB assembler syntax is shown below.

<table>
<thead>
<tr>
<th>L</th>
<th>B</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>STRH Rd, [Rb, Ro]</td>
<td>STRH Rd, [Rb, Ro]</td>
<td>Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>LDRH Rd, [Rb, Ro]</td>
<td>LDRH Rd, [Rb, Ro]</td>
<td>Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>LDSB Rd, [Rb, Ro]</td>
<td>LDRSB Rd, [Rb, Ro]</td>
<td>Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>LDSH Rd, [Rb, Ro]</td>
<td>LDRSH Rd, [Rb, Ro]</td>
<td>Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.</td>
</tr>
</tbody>
</table>

Table 4-9. Summary of format 8 instructions
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-9. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

<table>
<thead>
<tr>
<th>THUMB</th>
<th>ARM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRH</td>
<td>R4, [R3, R0]</td>
<td>Store the lower 16 bits of R4 at the address formed by adding R0 to R3.</td>
</tr>
<tr>
<td>LDSB</td>
<td>R2, [R7, R1]</td>
<td>Load into R2 the sign extended byte found at the address formed by adding R1 to R7.</td>
</tr>
<tr>
<td>LDSH</td>
<td>R3, [R4, R2]</td>
<td>Load into R3 the sign extended halfword found at the address formed by adding R2 to R4.</td>
</tr>
</tbody>
</table>
**FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET**

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>B</td>
<td>L</td>
<td>Offset5</td>
<td>Rb</td>
<td>Rd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **[2:0] Source/Destination Register**
- **[5:3] Base Register**
- **[10:6] Offset Register**
- **[11] Load/Store Flag**
  - 0 = Store to memory
  - 1 = Load from memory
- **[12] Byte/Word Flag**
  - 0 = Transfer word quantity
  - 1 = Transfer byte quantity

*Figure 4-10. Format 9*
OPERATION

These instructions transfer byte or word values between registers and memory using an immediate 5 or 7-bit offset. The THUMB assembler syntax is shown in Table 4-10.

### Table 4-10. Summary of Format 9 Instructions

<table>
<thead>
<tr>
<th>L</th>
<th>B</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>STR Rd, [Rb, #Imm]</td>
<td>STR Rd, [Rb, #Imm]</td>
<td>Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>LDR Rd, [Rb, #Imm]</td>
<td>LDR Rd, [Rb, #Imm]</td>
<td>Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>STRB Rd, [Rb, #Imm]</td>
<td>STRB Rd, [Rb, #Imm]</td>
<td>Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>LDRB Rd, [Rb, #Imm]</td>
<td>LDRB Rd, [Rb, #Imm]</td>
<td>Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.</td>
</tr>
</tbody>
</table>

**NOTE:** For word accesses (B = 0), the value specified by #Imm is a full 7-bit address, but must be word-aligned (i.e., with bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Offset5 field.

### INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-10. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

### EXAMPLES

```
LDR R2, [R5,#116] ; Load into R2 the word found at the address formed by adding 116 to R5. Note that the THUMB opcode will contain 29 as the Offset5 value.

STRB R1, [R0,#13] ; Store the lower 8 bits of R1 at the address formed by adding 13 to R0. Note that the THUMB opcode will contain 13 as the Offset5 value.
```
FORMAT 10: LOAD/STORE HALFWORD

Figure 4-11. Format 10

OPERATION
These instructions transfer halfword values between a Lo register and memory. Addresses are pre-indexed, using a 6-bit immediate value. The THUMB assembler syntax is shown in Table 4-11.

Table 4-11. Halfword Data Transfer Instructions

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STRH Rd, [Rb, #Imm]</td>
<td>STRH Rd, [Rb, #Imm]</td>
<td>Add #Imm to base address in Rb and store bits 0 - 15 of Rd at the resulting address.</td>
</tr>
<tr>
<td>1</td>
<td>LDRH Rd, [Rb, #Imm]</td>
<td>LDRH Rd, [Rb, #Imm]</td>
<td>Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.</td>
</tr>
</tbody>
</table>

NOTE: #Imm is a full 6-bit address but must be halfword-aligned (ie with bit 0 set to 0) since the assembler places #Imm >> 1 in the Offset5 field.
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-11. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Destination, Source [Offset]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRH</td>
<td>R6, [R1, #56]</td>
<td>Store the lower 16 bits of R4 at the address formed by adding 56 R1. Note that the THUMB opcode will contain 28 as the Offset value.</td>
</tr>
<tr>
<td>LDRH</td>
<td>R4, [R7, #4]</td>
<td>Load into R4 the halfword found at the address formed by adding 4 to R7. Note that the THUMB opcode will contain 2 as the Offset value.</td>
</tr>
</tbody>
</table>
FORMAT 11: SP-RELATIVE LOAD/STORE

![Figure 4-12. Format 11](image)

**[7:0] Immediate Value**

**[10:8] Destination Register**

**[11] Load/Store Bit**
0 = Store to memory
1 = Load from memory

**OPERATION**

The instructions in this group perform an SP-relative load or store. The THUMB assembler syntax is shown in the following table.

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STR Rd, [SP, #Imm]</td>
<td>STR Rd, [R13 #Imm]</td>
<td>Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.</td>
</tr>
<tr>
<td>1</td>
<td>LDR Rd, [SP, #Imm]</td>
<td>LDR Rd, [R13 #Imm]</td>
<td>Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.</td>
</tr>
</tbody>
</table>

**NOTE:** The offset supplied in #Imm is a full 10-bit address, but must always be word-aligned (ie bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Word8 field.
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-12. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STR R4, [SP,#492] ; Store the contents of R4 at the address
                  ; formed by adding 492 to SP (R13).
                  ; Note that the THUMB opcode will contain
                  ; 123 as the Word8 value.
```
FORMAT 12: LOAD ADDRESS

![Figure 4-13. Format 12](image)

**OPERATION**

These instructions calculate an address by adding an 10-bit constant to either the PC or the SP, and load the resulting address into a register. The THUMB assembler syntax is shown in the following table.

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADD Rd, PC, #Imm</td>
<td>ADD Rd, R15, #Imm</td>
<td>Add #Imm to the current value of the program counter (PC) and load the result into Rd.</td>
</tr>
<tr>
<td>1</td>
<td>ADD Rd, SP, #Imm</td>
<td>ADD Rd, R13, #Imm</td>
<td>Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.</td>
</tr>
</tbody>
</table>

**NOTE:** The value specified by #Imm is a full 10-bit value, but this must be word-aligned (ie with bits 1:0 set to 0) since the assembler places #Imm >> 2 in field Word 8.

Where the PC is used as the source register (SP = 0), bit 1 of the PC is always read as 0. The value of the PC will be 4 bytes greater than the address of the instruction before bit 1 is forced to 0.

The CPSR condition codes are unaffected by these instructions.
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-13. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

ADD R2, PC, #572 ; R2 := PC + 572, but don't set the condition codes. bit[1] of PC is forced to zero.
; Note that the THUMB opcode will contain 143 as the Word8 value.

ADD R6, SP, #212 ; R6 := SP (R13) + 212, but don't set the condition codes.
; Note that the THUMB opcode will contain 53 as the Word8 value.
FORMAT 13: ADD OFFSET TO STACK POINTER

![Figure 4-14. Format 13](image)

[6:0] 7-bit Immediate Value
[7] Sign Flag
0 = Offset is positive
1 = Offset is negative

OPERATION

This instruction adds a 9-bit signed constant to the stack pointer. The following table shows the THUMB assembler syntax.

Table 4-14. The ADD SP Instruction

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADD SP, #Imm</td>
<td>ADD R13, R13, #Imm</td>
<td>Add #Imm to the stack pointer (SP).</td>
</tr>
<tr>
<td>1</td>
<td>ADD SP, # -Imm</td>
<td>SUB R13, R13, #Imm</td>
<td>Add #-Imm to the stack pointer (SP).</td>
</tr>
</tbody>
</table>

NOTE: The offset specified by #Imm can be up to +/- 508, but must be word-aligned (ie with bits 1:0 set to 0) since the assembler converts #Imm to an 8-bit sign + magnitude number before placing it in field SWord7. The condition codes are not set by this instruction.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-14. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
ADD SP, #268 ; SP (R13) := SP + 268, but don't set the condition codes.
             ; Note that the THUMB opcode will contain 67 as the Word7 value and S=0.
ADD SP, #-104 ; SP (R13) := SP - 104, but don't set the condition codes.
               ; Note that the THUMB opcode will contain 26 as the Word7 value and S=1.
```
FORMAT 14: PUSH/POP REGISTERS

Figure 4-15. Format 14

OPERATION

The instructions in this group allow registers 0-7 and optionally LR to be pushed onto the stack, and registers 0-7 and optionally PC to be popped off the stack. The THUMB assembler syntax is shown in Table 4-15.

NOTE

The stack is always assumed to be Full Descending.

Table 4-15. PUSH and POP Instructions

<table>
<thead>
<tr>
<th>L</th>
<th>B</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>PUSH { Rlist }</td>
<td>STMDB R13!, { Rlist }</td>
<td>Push the registers specified by Rlist onto the stack. Update the stack pointer.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>PUSH { Rlist, LR }</td>
<td>STMDB R13!, { Rlist, R14 }</td>
<td>Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>POP { Rlist }</td>
<td>LDMIA R13!, { Rlist }</td>
<td>Pop values off the stack into the registers specified by Rlist. Update the stack pointer.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>POP { Rlist, PC }</td>
<td>LDMIA R13!, {Rlist, R15}</td>
<td>Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.</td>
</tr>
</tbody>
</table>
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-15. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

PUSH {R0-R4,LR} ; Store R0,R1,R2,R3,R4 and R14 (LR) at
; the stack pointed to by R13 (SP) and update R13.
; Useful at start of a sub-routine to
; save workspace and return address.

POP {R2,R6,PC} ; Load R2,R6 and R15 (PC) from the stack
; pointed to by R13 (SP) and update R13.
; Useful to restore workspace and return from sub-routine.
FORMAT 15: MULTIPLE LOAD/STORE

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>L</td>
<td>Rb</td>
<td></td>
<td>Rlist</td>
<td></td>
</tr>
</tbody>
</table>

[7:0] Register List

[10:8] Base Register

[11] Load/Store Bit
0 = Store to memory
1 = Load from memory

Figure 4-16. Format 15

OPERATION

These instructions allow multiple loading and storing of Lo registers. The THUMB assembler syntax is shown in the following table.

Table 4-16. The Multiple Load/Store Instructions

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STMIA Rb!, { Rlist }</td>
<td>STMIA Rb!, { Rlist }</td>
<td>Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.</td>
</tr>
<tr>
<td>1</td>
<td>LDMIA Rb!, { Rlist }</td>
<td>LDMIA Rb!, { Rlist }</td>
<td>Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.</td>
</tr>
</tbody>
</table>

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 4-16. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

STMIA R0!, {R3-R7} ; Store the contents of registers R3-R7
; starting at the address specified in
; R0, incrementing the addresses for each word.
; Write back the updated value of R0.
FORMAT 16: CONDITIONAL BRANCH

![Diagram of Format 16]

Figure 4-17. Format 16

OPERATION

The instructions in this group all perform a conditional Branch depending on the state of the CPSR condition codes. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

The THUMB assembler syntax is shown in the following table.

Table 4-17. The Conditional Branch Instructions

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>BEQ label</td>
<td>BEQ label</td>
<td>Branch if Z set (equal)</td>
</tr>
<tr>
<td>0001</td>
<td>BNE label</td>
<td>BNE label</td>
<td>Branch if Z clear (not equal)</td>
</tr>
<tr>
<td>0010</td>
<td>BCS label</td>
<td>BCS label</td>
<td>Branch if C set (unsigned higher or same)</td>
</tr>
<tr>
<td>0011</td>
<td>BCC label</td>
<td>BCC label</td>
<td>Branch if C clear (unsigned lower)</td>
</tr>
<tr>
<td>0100</td>
<td>BMI label</td>
<td>BMI label</td>
<td>Branch if N set (negative)</td>
</tr>
<tr>
<td>0101</td>
<td>BPL label</td>
<td>BPL label</td>
<td>Branch if N clear (positive or zero)</td>
</tr>
<tr>
<td>0110</td>
<td>BVS label</td>
<td>BVS label</td>
<td>Branch if V set (overflow)</td>
</tr>
<tr>
<td>0111</td>
<td>BVC label</td>
<td>BVC label</td>
<td>Branch if V clear (no overflow)</td>
</tr>
<tr>
<td>1000</td>
<td>BHI label</td>
<td>BHI label</td>
<td>Branch if C set and Z clear (unsigned higher)</td>
</tr>
</tbody>
</table>
### Table 4-17. The Conditional Branch Instructions (Continued)

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>BLS label</td>
<td>BLS label</td>
<td>Branch if C clear or Z set (unsigned lower or same)</td>
</tr>
<tr>
<td>1010</td>
<td>BGE label</td>
<td>BGE label</td>
<td>Branch if N set and V set, or N clear and V clear (greater or equal)</td>
</tr>
<tr>
<td>1011</td>
<td>BLT label</td>
<td>BLT label</td>
<td>Branch if N set and V clear, or N clear and V clear (less than)</td>
</tr>
<tr>
<td>1100</td>
<td>BGT label</td>
<td>BGT label</td>
<td>Branch if Z clear, and either N set and V set or N clear and V clear (greater than)</td>
</tr>
<tr>
<td>1101</td>
<td>BLE label</td>
<td>BLE label</td>
<td>Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)</td>
</tr>
</tbody>
</table>

**NOTES:**

1. While label specifies a full 9-bit two’s complement address, this must always be halfword-aligned (i.e., with bit 0 set to 0) since the assembler actually places `label >> 1` in field `SOffset8`.
2. Cond = 1110 is undefined, and should not be used. Cond = 1111 creates the SWI instruction: see...

**INSTRUCTION CYCLE TIMES**

All instructions in this format have an equivalent ARM instruction as shown in Table 3-1. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

**EXAMPLES**

```plaintext
CMP R0, #45 ; Branch to over-if R0 > 45.
BGT over  ; Note that the THUMB opcode will contain
• ; the number of halfwords to offset.
• ; Must be halfword aligned.
over  •
```
FORMAT 17: SOFTWARE INTERRUPT

Figure 4-18. Format 17

OPERATION
The SWI instruction performs a software interrupt. On taking the SWI, the processor switches into ARM state and enters Supervisor (SVC) mode.

The THUMB assembler syntax for this instruction is shown below.

Table 4-18. The SWI Instruction

<table>
<thead>
<tr>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWI Value 8</td>
<td>SWI Value 8</td>
<td>Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.</td>
</tr>
</tbody>
</table>

NOTE: Value8 is used solely by the SWI handler; it is ignored by the processor.

INSTRUCTION CYCLE TIMES
All instructions in this format have an equivalent ARM instruction as shown in Table 4-18. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

SWI 18 ; Take the software interrupt exception.
; Enter Supervisor mode with 18 as the ; requested SWI number.
FORMAT 18: UNCONDITIONAL BRANCH

[10:0] Immediate Value

Figure 4-19. Format 18

OPERATION

This instruction performs a PC-relative Branch. The THUMB assembler syntax is shown below. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

<table>
<thead>
<tr>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>B label</td>
<td>BAL label (halfword offset)</td>
<td>Branch PC relative +/- Offset11 &lt;&lt; 1, where label is PC +/- 2048 bytes.</td>
</tr>
</tbody>
</table>

NOTE: The address specified by label is a full 12-bit two's complement address, but must always be halfword aligned (ie bit 0 set to 0), since the assembler places label >> 1 in the Offset11 field.

EXAMPLES

here   B here ; Branch onto itself. Assembles to 0xE7FE.

B jimmy ; (Note effect of PC offset).

• ; Branch to 'jimmy'.

• ; Note that the THUMB opcode will contain the number of halfwords to offset.

• ; Must be halfword aligned.
FORMAT 19: LONG BRANCH WITH LINK

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Offset

Long Branch and Link Offset High/Low

Low/High Offset Bit
0 = Offset high
1 = Offset low

Figure 4-20. Format 19

OPERATION

This format specifies a long branch with link.

The assembler splits the 23-bit two's complement half-word offset specified by the label into two 11-bit halves, ignoring bit 0 (which must be 0), and creates two THUMB instructions.

Instruction 1 (H = 0)
In the first instruction the Offset field contains the upper 11 bits of the target address. This is shifted left by 12 bits and added to the current PC address. The resulting address is placed in LR.

Instruction 2 (H =1)
In the second instruction the Offset field contains an 11-bit representation lower half of the target address. This is shifted left by 1 bit and added to LR. LR, which now contains the full 23-bit address, is placed in PC, the address of the instruction following the BL is placed in LR and bit 0 of LR is set.

The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.
INSTRUCTION CYCLE TIMES

This instruction format does not have an equivalent ARM instruction.

Table 4-20. The BL Instruction

<table>
<thead>
<tr>
<th>L</th>
<th>THUMB assembler</th>
<th>ARM equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BL label</td>
<td>none</td>
<td>LR := PC + OffsetHigh &lt;&lt; 12</td>
</tr>
<tr>
<td>1</td>
<td>temp := next instruction address PC := LR + OffsetLow &lt;&lt; 1</td>
<td>LR := temp</td>
<td>1</td>
</tr>
</tbody>
</table>

EXAMPLES

BL faraway ; Unconditionally Branch to ‘faraway’
next ; and place following instruction
  • ; address, ie "next", in R14, the Link
  • ; register and set bit 0 of LR high.
  • ; Note that the THUMB opcodes will
  • ; contain the number of halfwords to offset.
faraway ; Must be Half-word aligned.
  •
**INSTRUCTION SET EXAMPLES**

The following examples show ways in which the THUMB instructions may be used to generate small and efficient code. Each example also shows the ARM equivalent so these may be compared.

**MULTIPLICATION BY A CONSTANT USING SHIFTS AND ADDS**

The following shows code to multiply by various constants using 1, 2 or 3 Thumb instructions alongside the ARM equivalents. For other constants it is generally better to use the built-in MUL instruction rather than using a sequence of 4 or more instructions.

<table>
<thead>
<tr>
<th>Thumb</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>LSL</td>
<td>Ra, Rb, LSL #n ; MOV Ra, Rb, LSL #n</td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>LSL</td>
<td>Rt, Rb, #n ; ADD Ra, Rb, Rb, LSL #n</td>
</tr>
<tr>
<td>ADD</td>
<td>Ra, Rt, Rb</td>
</tr>
<tr>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>LSL</td>
<td>Rt, Rb, #n ; RSB Ra, Rb, Rb, LSL #n</td>
</tr>
<tr>
<td>SUB</td>
<td>Ra, Rt, Rb</td>
</tr>
<tr>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>LSL</td>
<td>Ra, Rb, #n ; MOV Ra, Rb, LSL #n</td>
</tr>
<tr>
<td>MVN</td>
<td>Ra, Ra      ; RSB Ra, Ra, #0</td>
</tr>
<tr>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>LSL</td>
<td>Rt, Rb, #n ; SUB Ra, Rb, Rb, LSL #n</td>
</tr>
<tr>
<td>SUB</td>
<td>Ra, Rb, Rt</td>
</tr>
</tbody>
</table>

Multiplication by any \( C = (2^n+1, 2^n-1, -2^n or -2^n-1) \times 2^n \)

Effectively this is any of the multiplications in 2 to 5 followed by a final shift. This allows the following additional constants to be multiplied: 6, 10, 12, 14, 18, 20, 24, 28, 30, 34, 36, 40, 48, 56, 60, 62 ......

<table>
<thead>
<tr>
<th>Thumb</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2..5)</td>
<td></td>
</tr>
<tr>
<td>LSL</td>
<td>Ra, Ra, #n ; MOV Ra, Ra, LSL #n</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GENERAL PURPOSE SIGNED DIVIDE

This example shows a general purpose signed divide and remainder routine in both Thumb and ARM code.

**Thumb code**

```assembly
;signed_divide ; Signed divide of R1 by R0: returns quotient in R0, ; remainder in R1

;Get abs value of R0 into R3
    ASR R2, R0, #31 ; Get 0 or -1 in R2 depending on sign of R0
    EOR R0, R2 ; EOR with -1 (0xFFFFFFFF) if negative
    SUB R3, R0, R2 ; and ADD 1 (SUB -1) to get abs value

;SUB always sets flag so go & report division by 0 if necessary
    BEQ divide_by_zero

;Get abs value of R1 by xoring with 0xFFFFFFFF and adding 1 if negative
    ASR R0, R1, #31 ; Get 0 or -1 in R3 depending on sign of R1
    EOR R1, R0 ; EOR with -1 (0xFFFFFFFF) if negative
    SUB R1, R0 ; and ADD 1 (SUB -1) to get abs value

;Save signs (0 or -1 in R0 & R2) for later use in determining ; sign of quotient & remainder.
    PUSH  {R0, R2}

;Justification, shift 1 bit at a time until divisor (R0 value) ; is just <= than dividend (R1 value). To do this shift ; dividend ; right by 1 and stop as soon as shifted value becomes >.
    LSR R0, R1, #1
    MOV R2, R3
    B  %FT0
    just_l LSL R2, #1
    CMP R2, R0
    BLS just_l
    MOV R0, #0 ; Set accumulator to 0
    B  %FT0 ; Branch into division loop

div_l LSR R2, #1
0    CMP R1, R2
    BCC %FT0
    SUB R1, R2 ; If successful do a real subtract
    0 ADC R0, R0 ; Shift result and add 1 if subtract succeeded

    CMP R2, R3 ; Terminate when R2 == R3 (ie we have just
    BNE div_l ; tested subtracting the 'ones' value).
```

---

SAMSUNG  
ELECTRONICS

4-41
Now fix up the signs of the quotient (R0) and remainder (R1)

```assembly
POP    {R2, R3} ; Get dividend/divisor signs back
EOR    R3, R2   ; Result sign
EOR    R0, R3   ; Negate if result sign = -1
SUB    R0, R3   ; Negate remainder if dividend sign = -1
EOR    R1, R2
SUB    R1, R2
MOV    pc, lr
```

**ARM Code**

```assembly
signed_divide ; Effectively zero a4 as top bit will be shifted out later
    ANDS   a4, a1, #&80000000
    RSBMI  a1, a1, #0
    EORS   ip, a4, a2, ASR #32
;ip bit 31 = sign of result
;ip bit 30 = sign of a2
    RSBCS  a2, a2, #0
;Central part is identical code to udiv (without MOV a4, #0 which comes for free as part of signed entry sequence)
    MOVVS  a3, a1
    BEQ    divide_by_zero
just_l
    CMP    a3, a2, LSR #1
    MOVLS  a3, a3, LSL #1 ; NB: LSL #1 is always OK if LS succeeds
    BLO    s_loop
div_l
    CMP    a2, a3
    ADC    a4, a4, a4
    SUBCS  a2, a2, a3
    TEQ    a3, a1
    MOVNE  a3, a3, LSR #1
    BNE    s_loop2
    MOV    a1, a4
    MOVVS  ip, ip, ASL #1
    RSBCS  a1, a1, #0
    RSBMI  a2, a2, #0
    MOV    pc, lr
```

---

4-42
DIVISION BY A CONSTANT

Division by a constant can often be performed by a short fixed sequence of shifts, adds and subtracts.

Here is an example of a divide by 10 routine based on the algorithm in the ARM Cookbook in both Thumb and ARM code.

**Thumb Code**

```
udiv10 ; Take argument in a1 returns quotient in a1, 
         ; remainder in a2
   MOV     a2, a1
   LSR     a3, a1, #2
   SUB     a1, a3
   LSR     a3, a1, #4
   ADD     a1, a3
   LSR     a3, a1, #8
   ADD     a1, a3
   LSR     a3, a1, #16
   ADD     a1, a3
   LSR     a1, #3
   ASL     a3, a1, #2
   ADD    a3, a1
   ASL     a3, #1
   SUB    a2, a3
   CMP     a2, #10
   BLT     %FT0
   ADD     a1, #1
   SUB    a2, #10
0
   MOV    pc, lr
```

**ARM Code**

```
udiv10 ; Take argument in a1 returns quotient in a1, 
         ; remainder in a2
 SUB     a2, a1, #10
 SUB     a1, a1, a1, lsr #2
 ADD     a1, a1, a1, lsr #4
 ADD     a1, a1, a1, lsr #8
 ADD     a1, a1, a1, lsr #16
 MOV     a1, a1, lsr #3
 ADD     a3, a1, a1, asl #2
 SUBS    a2, a2, a3, asl #1
 ADDPL   a1, a1, #1
 ADDDMI  a2, a2, #10
 MOV     pc, lr
```
NOTES
MEMORY CONTROLLER

OVERVIEW

The S3C2410X memory controller provides memory control signals required for external memory access.

The S3C2410X has the following features:

— Little/Big endian (selectable by a software)
— Address space: 128Mbytes per bank (total 1GB/8 banks)
— Programmable access size (8/16/32-bit) for all banks except bank0 (16/32-bit)
— Total 8 memory banks
  Six memory banks for ROM, SRAM, etc.
  Remaining two memory banks for ROM, SRAM, SDRAM, etc.
— Seven fixed memory bank start address
— One flexible memory bank start address and programmable bank size
— Programmable access cycles for all memory banks
— External wait to extend the bus cycles
— Supporting self-refresh and power down mode in SDRAM
NOTES:
1. SROM means ROM or SRAM type memory.
2. SFR means Special Function Register.

Figure 5-1. S3C2410X Memory Map after Reset

Table 5-1. Bank 6/7 Addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>2MB</th>
<th>4MB</th>
<th>8MB</th>
<th>16MB</th>
<th>32MB</th>
<th>64MB</th>
<th>128MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start address</td>
<td>0x3000_0000</td>
<td>0x3000_0000</td>
<td>0x3000_0000</td>
<td>0x3000_0000</td>
<td>0x3000_0000</td>
<td>0x3000_0000</td>
<td>0x3000_0000</td>
</tr>
<tr>
<td>End address</td>
<td>0x301f_ffff</td>
<td>0x303f_ffff</td>
<td>0x307f_ffff</td>
<td>0x30ff_ffff</td>
<td>0x31ff_ffff</td>
<td>0x33ff_ffff</td>
<td>0x37ff_ffff</td>
</tr>
<tr>
<td>Bank 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start address</td>
<td>0x3020_0000</td>
<td>0x3040_0000</td>
<td>0x3080_0000</td>
<td>0x3100_0000</td>
<td>0x3200_0000</td>
<td>0x3400_0000</td>
<td>0x3800_0000</td>
</tr>
<tr>
<td>End address</td>
<td>0x303f_ffff</td>
<td>0x307f_ffff</td>
<td>0x30ff_ffff</td>
<td>0x31ff_ffff</td>
<td>0x33ff_ffff</td>
<td>0x37ff_ffff</td>
<td>0x3fff_ffff</td>
</tr>
</tbody>
</table>

NOTE: Bank 6 and 7 must have the same memory size.
FUNCTION DESCRIPTION

BANK0 BUS WIDTH

The data bus of BANK0 (nGCS0) should be configured in width as one of 16-bit and 32-bit ones. Because the BANK0 works as the booting ROM bank (map to 0x0000_0000), the bus width of BANK0 should be determined before the first ROM access, which will depend on the logic level of OM[1:0] at Reset.

<table>
<thead>
<tr>
<th>OM1 (Operating Mode 1)</th>
<th>OM0 (Operating Mode 0)</th>
<th>Booting ROM Data width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Nand Flash Mode</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>16-bit</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>32-bit</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Test Mode</td>
</tr>
</tbody>
</table>

MEMORY (SROM/SDRAM) ADDRESS PIN CONNECTIONS

<table>
<thead>
<tr>
<th>MEMORY ADDR. PIN</th>
<th>S3C2410X ADDR. @ 8-bit DATA BUS</th>
<th>S3C2410X ADDR. @ 16-bit DATA BUS</th>
<th>S3C2410X ADDR. @ 32-bit DATA BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>A0</td>
<td>A1</td>
<td>A2</td>
</tr>
<tr>
<td>A1</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
</tr>
<tr>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
</tr>
</tbody>
</table>
### SDRAM Bank Address Pin Connection

#### Table 5-2. SDRAM Bank Address Configuration

<table>
<thead>
<tr>
<th>Bank Size</th>
<th>Bus Width</th>
<th>Base Component</th>
<th>Memory Configuration</th>
<th>Bank Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>2MB</td>
<td>x8</td>
<td>16Mb</td>
<td>(1M x 8 x 2banks) x 1 ea</td>
<td>A20</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(512K x 16 x 2banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td>4MB</td>
<td>x8</td>
<td>16Mb</td>
<td>(2M x 4 x 2banks) x 2 ea</td>
<td>A21</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(1M x 8 x 2banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(512K x 16 x 2banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td>8MB</td>
<td>x16</td>
<td>16Mb</td>
<td>(2M x 4 x 2banks) x 4 ea</td>
<td>A22</td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(1M x 8 x 2banks) x 4 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td>64Mb</td>
<td>(4M x 8 x 2banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td></td>
<td>(2M x 8 x 4banks) x 1 ea</td>
<td>A[22:21]</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(2M x 16 x 2banks) x 1 ea</td>
<td>A22</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(1M x 16 x 4banks) x 1 ea</td>
<td>A[22:21]</td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(512K x 32 x 4banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td>16MB</td>
<td>x32</td>
<td>16Mb</td>
<td>(2M x 4 x 2banks) x 8 ea</td>
<td>A23</td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td>64Mb</td>
<td>(8M x 4 x 2banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td></td>
<td>(4M x 4 x 4banks) x 2 ea</td>
<td>A[23:22]</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(4M x 8 x 2banks) x 2 ea</td>
<td>A23</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(2M x 8 x 4banks) x 2 ea</td>
<td>A[23:22]</td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(2M x 16 x 2banks) x 2 ea</td>
<td>A23</td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(1M x 16 x 4banks) x 2 ea</td>
<td>A[23:22]</td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td>128Mb</td>
<td>(4M x 8 x 4banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(2M x 16 x 4banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td>32MB</td>
<td>x16</td>
<td>64Mb</td>
<td>(8M x 4 x 2banks) x 4 ea</td>
<td>A24</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(4M x 4 x 4banks) x 4 ea</td>
<td>A[24:23]</td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(4M x 8 x 2banks) x 4 ea</td>
<td>A24</td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(2M x 8 x 4banks) x 4 ea</td>
<td>A[24:23]</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td>128Mb</td>
<td>(4M x 8 x 4banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(2M x 16 x 4banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td>256Mb</td>
<td>(8M x 8 x 4banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(4M x 16 x 4banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td>64MB</td>
<td>x32</td>
<td>128Mb</td>
<td>(4M x 8 x 4banks) x 4 ea</td>
<td>A[25:24]</td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td>256Mb</td>
<td>(8M x 8 x 4banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(4M x 16 x 4banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td>512Mb</td>
<td>(16M x 8 x 4banks) x 1 ea</td>
<td></td>
</tr>
<tr>
<td>128MB</td>
<td>x32</td>
<td>256Mb</td>
<td>(8M x 8 x 4banks) x 4 ea</td>
<td>A[26:25]</td>
</tr>
<tr>
<td></td>
<td>x8</td>
<td>512Mb</td>
<td>(32M x 4 x 4banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x16</td>
<td></td>
<td>(16M x 8 x 4banks) x 2 ea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x32</td>
<td></td>
<td>(8M x 16 x 4banks) x 2 ea</td>
<td></td>
</tr>
</tbody>
</table>
nWAIT PIN OPERATION

If the WAIT corresponding to each memory bank is enabled, the nOE duration should be prolonged by the external nWAIT pin while the memory bank is active. nWAIT is checked from tacc=1. nOE will be deasserted at the next clock after sampling nWAIT is high. The nWE signal have the same relation with nOE.

![S3C2410X External nWAIT Timing Diagram (Tacc=4)](image)

*Figure 5-2. S3C2410X External nWAIT Timing Diagram (Tacc=4)*
nXBREQ/nXBACK Pin Operation

If nXBREQ is asserted, the S3C2410X will respond by lowering nXBACK. If nXBACK=L, the address/data bus and memory control signals are in Hi-Z state as shown in Table 1-1. When nXBREQ is de-asserted, the nXBACK will also be de-asserted.

Figure 5-3. S3C2410X nXBREQ/nXBACK Timing Diagram
ROM Memory Interface Examples

Figure 5-4. Memory Interface with 8-bit ROM

Figure 5-5. Memory Interface with 8-bit ROM × 2
Figure 5-6. Memory Interface with 8-bit ROM × 4

Figure 5-7. Memory Interface with 16-bit ROM
**SRAM Memory Interface Examples**

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>DQ0</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>A1</td>
<td>DQ1</td>
<td>D1</td>
</tr>
<tr>
<td>A3</td>
<td>A2</td>
<td>DQ2</td>
<td>D2</td>
</tr>
<tr>
<td>A4</td>
<td>A3</td>
<td>DQ3</td>
<td>D3</td>
</tr>
<tr>
<td>A5</td>
<td>A4</td>
<td>DQ4</td>
<td>D4</td>
</tr>
<tr>
<td>A6</td>
<td>A5</td>
<td>DQ5</td>
<td>D5</td>
</tr>
<tr>
<td>A7</td>
<td>A6</td>
<td>DQ6</td>
<td>D6</td>
</tr>
<tr>
<td>A8</td>
<td>A7</td>
<td>DQ7</td>
<td>D7</td>
</tr>
<tr>
<td>A9</td>
<td>A8</td>
<td>DQ8</td>
<td>D8</td>
</tr>
<tr>
<td>A10</td>
<td>A9</td>
<td>DQ9</td>
<td>D9</td>
</tr>
<tr>
<td>A11</td>
<td>A10</td>
<td>DQ10</td>
<td>D10</td>
</tr>
<tr>
<td>A12</td>
<td>A11</td>
<td>DQ11</td>
<td>D11</td>
</tr>
<tr>
<td>A13</td>
<td>A12</td>
<td>DQ12</td>
<td>D12</td>
</tr>
<tr>
<td>A14</td>
<td>A13</td>
<td>DQ13</td>
<td>D13</td>
</tr>
<tr>
<td>A15</td>
<td>A14</td>
<td>DQ14</td>
<td>D14</td>
</tr>
<tr>
<td>A16</td>
<td>A15</td>
<td>DQ15</td>
<td>D15</td>
</tr>
</tbody>
</table>

nWE  nWE  
nOE  nOE  
nCS  nGCSn  
nUB  nBE1  
nLB  nBE0

**Figure 5-8. Memory Interface with 16-bit SRAM**

<table>
<thead>
<tr>
<th>A2</th>
<th>A0</th>
<th>DQ0</th>
<th>D0</th>
<th>A2</th>
<th>A0</th>
<th>DQ0</th>
<th>D16</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3</td>
<td>A1</td>
<td>DQ1</td>
<td>D1</td>
<td>A3</td>
<td>A1</td>
<td>DQ1</td>
<td>D17</td>
</tr>
<tr>
<td>A4</td>
<td>A2</td>
<td>DQ2</td>
<td>D2</td>
<td>A4</td>
<td>A2</td>
<td>DQ2</td>
<td>D18</td>
</tr>
<tr>
<td>A5</td>
<td>A3</td>
<td>DQ3</td>
<td>D3</td>
<td>A5</td>
<td>A3</td>
<td>DQ3</td>
<td>D19</td>
</tr>
<tr>
<td>A6</td>
<td>A4</td>
<td>DQ4</td>
<td>D4</td>
<td>A6</td>
<td>A4</td>
<td>DQ4</td>
<td>D20</td>
</tr>
<tr>
<td>A7</td>
<td>A5</td>
<td>DQ5</td>
<td>D5</td>
<td>A7</td>
<td>A5</td>
<td>DQ5</td>
<td>D21</td>
</tr>
<tr>
<td>A8</td>
<td>A6</td>
<td>DQ6</td>
<td>D6</td>
<td>A8</td>
<td>A6</td>
<td>DQ6</td>
<td>D22</td>
</tr>
<tr>
<td>A9</td>
<td>A7</td>
<td>DQ7</td>
<td>D7</td>
<td>A9</td>
<td>A7</td>
<td>DQ7</td>
<td>D23</td>
</tr>
<tr>
<td>A10</td>
<td>A8</td>
<td>DQ8</td>
<td>D8</td>
<td>A10</td>
<td>A8</td>
<td>DQ8</td>
<td>D24</td>
</tr>
<tr>
<td>A11</td>
<td>A9</td>
<td>DQ9</td>
<td>D9</td>
<td>A11</td>
<td>A9</td>
<td>DQ9</td>
<td>D25</td>
</tr>
<tr>
<td>A12</td>
<td>A10</td>
<td>DQ10</td>
<td>D10</td>
<td>A12</td>
<td>A10</td>
<td>DQ10</td>
<td>D26</td>
</tr>
<tr>
<td>A13</td>
<td>A11</td>
<td>DQ11</td>
<td>D11</td>
<td>A13</td>
<td>A11</td>
<td>DQ11</td>
<td>D27</td>
</tr>
<tr>
<td>A14</td>
<td>A12</td>
<td>DQ12</td>
<td>D12</td>
<td>A14</td>
<td>A12</td>
<td>DQ12</td>
<td>D28</td>
</tr>
<tr>
<td>A15</td>
<td>A13</td>
<td>DQ13</td>
<td>D13</td>
<td>A15</td>
<td>A13</td>
<td>DQ13</td>
<td>D29</td>
</tr>
<tr>
<td>A16</td>
<td>A14</td>
<td>DQ14</td>
<td>D14</td>
<td>A16</td>
<td>A14</td>
<td>DQ14</td>
<td>D30</td>
</tr>
<tr>
<td>A17</td>
<td>A15</td>
<td>DQ15</td>
<td>D15</td>
<td>A17</td>
<td>A15</td>
<td>DQ15</td>
<td>D31</td>
</tr>
</tbody>
</table>

nWE  nWE  
nOE  nOE  
nCS  nGCSn  
nUB  nBE1  
nLB  nBE0

**Figure 5-9. Memory Interface with 16-bit SRAM × 2**
SDRAM Memory Interface Examples

![SDRAM Memory Interface Diagram](image)

Figure 5-10. Memory Interface with 16-bit SDRAM (8MB: 1Mb × 16 × 4banks)

![SDRAM Memory Interface Diagram](image)

Figure 5-11. Memory Interface with 16-bit SDRAM (16MB: 1Mb × 16 × 4banks × 2ea)

NOTE: Refer to Table 5-2 for the Bank Address configurations of SDRAM.
PROGRAMMABLE ACCESS CYCLE

Figure 5-12. S3C2410X nGCS Timing Diagram

Tacs = 1 cycle    Tacc = 3 cycles
Tcos = 1 cycle    Tacp = 2 cycles
Tcoh = 1 cycle    Tcah = 2 cycles
Figure 5-13. S3C2410X SDRAM Timing Diagram

Data (CL2)

Data (CL3)

nWE

DQM

Bank Precharge

Row Active

Write

Read (CL = 2, CL = 3, BL = 1)

Trp = 2 cycle

Tcas = 2 cycle

Trcd = 2 cycle

Tcp = 2 cycle
BUS WIDTH & WAIT CONTROL REGISTER (BWSCON)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWSCON</td>
<td>0x48000000</td>
<td>R/W</td>
<td>Bus width &amp; wait status control register</td>
<td>0x000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BWSCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
</table>
|        |      | 0 = Not using UB/LB (The pins are dedicated nWBE[3:0])  
|        |      | 1 = Using UB/LB (The pins are dedicated nBE[3:0]) | 0           |
|        |      | 0 = WAIT disable, 1 = WAIT enable | 0           |
|        |      | 00 = 8-bit, 01 = 16-bit, 10 = 32-bit, 11 = reserved | 0           |
|        |      | 0 = Not using UB/LB (The pins are dedicated nWBE[3:0])  
|        |      | 1 = Using UB/LB (The pins are dedicated nBE[3:0]) | 0           |
|        |      | 0 = WAIT disable, 1 = WAIT enable | 0           |
|        |      | 00 = 8-bit, 01 = 16-bit, 10 = 32-bit, 11 = reserved | 0           |
|        |      | 0 = Not using UB/LB (The pins are dedicated nWBE[3:0])  
|        |      | 1 = Using UB/LB (The pins are dedicated nBE[3:0]) | 0           |
| WS5    | [22] | Determine WAIT status for bank 5.  
|        |      | 0 = WAIT disable, 1 = WAIT enable | 0           |
|        |      | 00 = 8-bit, 01 = 16-bit, 10 = 32-bit, 11 = reserved | 0           |
|        |      | 0 = Not using UB/LB (The pins are dedicated nWBE[3:0])  
|        |      | 1 = Using UB/LB (The pins are dedicated nBE[3:0]) | 0           |
|        |      | 0 = WAIT disable, 1 = WAIT enable | 0           |
|        |      | 00 = 8-bit, 01 = 16-bit, 10 = 32-bit, 11 = reserved | 0           |
|        |      | 0 = Not using UB/LB (The pins are dedicated nWBE[3:0])  
|        |      | 1 = Using UB/LB (The pins are dedicated nBE[3:0]) | 0           |
|        |      | 0 = WAIT disable, 1 = WAIT enable | 0           |
|        |      | 00 = 8-bit, 01 = 16-bit, 10 = 32-bit, 11 = reserved | 0           |
|        |      | 0 = Not using UB/LB (The pins are dedicated nWBE[3:0])  
|        |      | 1 = Using UB/LB (The pins are dedicated nBE[3:0]) | 0           |
### BUS WIDTH & WAIT CONTROL REGISTER (BWSCON) (Continued)

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit(s)</th>
<th>Description</th>
<th>Values</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS2</td>
<td>[10]</td>
<td>Determine WAIT status for bank 2.</td>
<td>0 = WAIT disable, 1 = WAIT enable</td>
<td>0</td>
</tr>
<tr>
<td>DW2</td>
<td>[9:8]</td>
<td>Determine data bus width for bank 2.</td>
<td>00 = 8-bit, 01 = 16-bit, 10 = 32-bit, 11 = reserved</td>
<td>0</td>
</tr>
<tr>
<td>ST1</td>
<td>[7]</td>
<td>Determine SRAM for using UB/LB for bank 1.</td>
<td>0 = Not using UB/LB (The pins are dedicated nWBE[3:0]), 1 = Using UB/LB (The pins are dedicated nBE[3:0])</td>
<td>0</td>
</tr>
<tr>
<td>WS1</td>
<td>[6]</td>
<td>Determine WAIT status for bank 1.</td>
<td>0 = WAIT disable, 1 = WAIT enable</td>
<td>0</td>
</tr>
<tr>
<td>DW1</td>
<td>[5:4]</td>
<td>Determine data bus width for bank 1.</td>
<td>00 = 8-bit, 01 = 16-bit, 10 = 32-bit, 11 = reserved</td>
<td>0</td>
</tr>
<tr>
<td>DW0</td>
<td>[2:1]</td>
<td>Indicate data bus width for bank 0 (read only).</td>
<td>01 = 16-bit, 10 = 32-bit, The states are selected by OM[1:0] pins</td>
<td>-</td>
</tr>
<tr>
<td>Reserved</td>
<td>[0]</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**NOTES:**

1. All types of master clock in this memory controller correspond to the bus clock.
   For example, HCLK in SRAM is the same as the bus clock, and SCLK in SDRAM is also the same as the bus clock. In this chapter (Memory Controller), one clock means one bus clock.
2. nBE[3:0] is the 'AND' signal nWBE[3:0] and nOE.
### BANK CONTROL REGISTER (BANKCONN: nGCS0-nGCS5)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANKCON0</td>
<td>0x48000004</td>
<td>R/W</td>
<td>Bank 0 control register</td>
<td>0x0700</td>
</tr>
<tr>
<td>BANKCON1</td>
<td>0x48000008</td>
<td>R/W</td>
<td>Bank 1 control register</td>
<td>0x0700</td>
</tr>
<tr>
<td>BANKCON2</td>
<td>0x4800000C</td>
<td>R/W</td>
<td>Bank 2 control register</td>
<td>0x0700</td>
</tr>
<tr>
<td>BANKCON3</td>
<td>0x48000010</td>
<td>R/W</td>
<td>Bank 3 control register</td>
<td>0x0700</td>
</tr>
<tr>
<td>BANKCON4</td>
<td>0x48000014</td>
<td>R/W</td>
<td>Bank 4 control register</td>
<td>0x0700</td>
</tr>
<tr>
<td>BANKCON5</td>
<td>0x48000018</td>
<td>R/W</td>
<td>Bank 5 control register</td>
<td>0x0700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BANKCONn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tacs</td>
<td>[14:13]</td>
<td>Address set-up time before nGCSn</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 1 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 2 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tcos</td>
<td>[12:11]</td>
<td>Chip selection set-up time before nOE</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 1 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 2 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tacc</td>
<td>[10:8]</td>
<td>Access cycle</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000 = 1 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>010 = 3 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>011 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>100 = 6 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>101 = 8 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>110 = 10 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>111 = 14 clocks</td>
<td></td>
</tr>
<tr>
<td>Tcoh</td>
<td>[7:6]</td>
<td>Chip selection hold time after nOE</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 1 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 2 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tcah</td>
<td>[5:4]</td>
<td>Address hold time after nGCSn</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 1 clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 2 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tcph</td>
<td>[3:2]</td>
<td>Page mode access cycle @ Page mode</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 2 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 3 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 6 clocks</td>
<td></td>
</tr>
<tr>
<td>PMC</td>
<td>[1:0]</td>
<td>Page mode configuration</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = normal (1 data)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 4 data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 8 data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 16 data</td>
<td></td>
</tr>
</tbody>
</table>
### BANK CONTROL REGISTER (BANKCONn: nGCS6-nGCS7)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANKCON6</td>
<td>0x48000001C</td>
<td>R/W</td>
<td>Bank 6 control register</td>
<td>0x18008</td>
</tr>
<tr>
<td>BANKCON7</td>
<td>0x480000020</td>
<td>R/W</td>
<td>Bank 7 control register</td>
<td>0x18008</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BANKCONn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT</td>
<td>[16:15]</td>
<td>Determine the memory type for bank6 and bank7.</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = ROM or SRAM 01 = Reserved (Do not use) 10 = Reserved (Do not use) 11 = Sync. DRAM</td>
<td></td>
</tr>
<tr>
<td>Memory Type = ROM or SRAM [MT=00] (15-bit)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tacs</td>
<td>[14:13]</td>
<td>Address set-up time before nGCS</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tcos</td>
<td>[12:11]</td>
<td>Chip selection set-up time before nOE</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tacc</td>
<td>[10:8]</td>
<td>Access cycle</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000 = 1 clock 001 = 2 clocks 010 = 3 clocks 011 = 4 clocks 100 = 6 clocks 101 = 8 clocks 110 = 10 clocks 111 = 14 clocks</td>
<td></td>
</tr>
<tr>
<td>Tcoh</td>
<td>[7:6]</td>
<td>Chip selection hold time after nOE</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tcah</td>
<td>[5:4]</td>
<td>Address hold time after nGCSn</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>Tacp</td>
<td>[3:2]</td>
<td>Page mode access cycle @ Page mode</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 2 clocks 01 = 3 clocks 10 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>PMC</td>
<td>[1:0]</td>
<td>Page mode configuration</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = normal (1 data) 01 = 4 consecutive accesses 10 = 8 consecutive accesses 11 = 16 consecutive accesses</td>
<td></td>
</tr>
<tr>
<td>Memory Type = SDRAM [MT=11] (4-bit)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trcd</td>
<td>[3:2]</td>
<td>RAS to CAS delay</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 2 clocks 01 = 3 clocks 10 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td>SCAN</td>
<td>[1:0]</td>
<td>Column address number</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 8-bit 01 = 9-bit 10= 10-bit</td>
<td></td>
</tr>
</tbody>
</table>
# REFRESH CONTROL REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFRESH</td>
<td>0x48000024</td>
<td>R/W</td>
<td>SDRAM refresh control register</td>
<td>0xac0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFRESH</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFEN</td>
<td>[23]</td>
<td>SDRAM Refresh Enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Enable (self/auto refresh)</td>
<td></td>
</tr>
<tr>
<td>TREFMD</td>
<td>[22]</td>
<td>SDRAM Refresh Mode</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Auto Refresh</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Self Refresh</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In self-refresh time, the SDRAM control signals are driven to the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>appropriate level.</td>
<td></td>
</tr>
<tr>
<td>Trp</td>
<td>[21:20]</td>
<td>SDRAM RAS pre-charge Time</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 2 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 3 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = Not support</td>
<td></td>
</tr>
<tr>
<td>Tsrc</td>
<td>[19:18]</td>
<td>SDRAM Semi Row Cycle Time</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 4 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 5 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 6 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 7 clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SDRAM’s Row_Cycle time (Trc) = Tsrc + Trp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If Trp = 3 clocks &amp; Tsrc = 7 clocks, Trc = 3 + 7 = 10 clocks</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>[17:16]</td>
<td>Not used</td>
<td>00</td>
</tr>
<tr>
<td>Reserved</td>
<td>[15:11]</td>
<td>Not used</td>
<td>0000</td>
</tr>
<tr>
<td>Refresh</td>
<td>[10:0]</td>
<td>SDRAM refresh count value.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refresh period = (2^{11} - refresh_count + 1)/HCLK</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ex) If refresh period is 15.6 us and HCLK is 60 MHz,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>the refresh count is as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refresh count = 2^{11} + 1 - 60x15.6 = 1113</td>
<td></td>
</tr>
</tbody>
</table>
## BANKSIZE REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANKSIZE</td>
<td>0x48000028</td>
<td>R/W</td>
<td>Flexible bank size register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BANKSIZE Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>BURST_EN [7]</td>
<td>ARM core burst operation enable. 0 = Disable burst operation. 1 = Enable burst operation.</td>
<td>0</td>
</tr>
<tr>
<td>Reserved [6]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>SCKE_EN [5]</td>
<td>SDRAM power down mode enable control by SCKE 0 = SDRAM power down mode disable 1 = SDRAM power down mode enable</td>
<td>0</td>
</tr>
<tr>
<td>SCLK_EN [4]</td>
<td>SCLK is enabled only during SDRAM access cycle for reducing power consumption. When SDRAM is not accessed, SCLK becomes 'L' level. 0 = SCLK is always active. 1 = SCLK is active only during the access (recommended).</td>
<td>0</td>
</tr>
<tr>
<td>Reserved [3]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>BK76MAP [2:0]</td>
<td>BANK6/7 memory map 010 = 128MB/128MB 001 = 64MB/64MB 000 = 32M/32M 111 = 16M/16M 110 = 8M/8M 101 = 4M/4M 100 = 2M/2M</td>
<td>010</td>
</tr>
</tbody>
</table>
## SDRAM MODE REGISTER SET REGISTER (MRSR)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRSRB6</td>
<td>0x4800002C</td>
<td>R/W</td>
<td>Mode register set register bank6</td>
<td>xxx</td>
</tr>
<tr>
<td>MRSRB7</td>
<td>0x48000030</td>
<td>R/W</td>
<td>Mode register set register bank7</td>
<td>xxx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MRSR</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[11:10]</td>
<td>Not used</td>
<td>–</td>
</tr>
<tr>
<td>WBL</td>
<td>[9]</td>
<td>Write burst length</td>
<td>x</td>
</tr>
<tr>
<td>TM</td>
<td>[8:7]</td>
<td>Test mode</td>
<td>xx</td>
</tr>
<tr>
<td>CAS latency</td>
<td>000 = 1 clock, 010 = 2 clocks, 011=3 clocks</td>
<td>Others: reserved</td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>[3]</td>
<td>Burst type</td>
<td>x</td>
</tr>
<tr>
<td>BL</td>
<td>[2:0]</td>
<td>Burst length</td>
<td>xxx</td>
</tr>
<tr>
<td>000: 1 (Fixed)</td>
<td>1: Reserved</td>
<td>Others: Reserved</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** MRSR register must not be reconfigured while the code is running on SDRAM.

**IMPORTANT NOTES**

In Power_OFF mode, SDRAM has to enter SDRAM self-refresh mode.
OVERVIEW

Recently, a NOR flash memory gets high in price while an SDRAM and a NAND flash memory get moderate, motivating some users to execute the boot code on a NAND flash and execute the main code on an SDRAM.

S3C2410X boot code can be executed on an external NAND flash memory. In order to support NAND flash boot loader, the S3C2410X is equipped with an internal SRAM buffer called ‘Steppingstone’. When booting, the first 4 KBytes of the NAND flash memory will be loaded into Steppingstone and the boot code loaded into Steppingstone will be executed.

Generally, the boot code will copy NAND flash content to SDRAM. Using hardware ECC, the NAND flash data validity will be checked. Upon the completion of the copy, the main program will be executed on the SDRAM.

FEATURES

- NAND Flash mode: Support read/erase/program NAND flash memory
- Auto boot mode : The boot code is transferred into Steppingstone after reset. After the transfer, the boot code will be executed on the Steppingstone.
- Hardware ECC detecting block (for hardware detecting and software correcting)
- The Steppingstone 4-KB internal SRAM buffer can be used for another purpose after NAND flash booting.
BLOCK DIAGRAM

![Figure 6-1. NAND Flash Controller Block Diagram](image)

OPERATION SCHEME

![Figure 6-2. NAND Flash Operation Scheme](image)
AUTO BOOT MODE SEQUENCE

1. Reset is completed.
2. When the auto boot mode is enabled, the first 4 KBytes of NAND flash memory is copied onto Steppingstone 4-KB internal buffer.
3. The Steppingstone is mapped to nGCS0.
4. CPU starts to execute the boot code on the Steppingstone 4-KB internal buffer.

**NOTE**

In the auto boot mode, ECC is not checked. So, The first 4 KBytes of NAND flash should have no bit error.

NAND FLASH MODE CONFIGURATION

1. Set NAND flash configuration by NFCONF register.
2. Write NAND flash command onto NFCMD register.
3. Write NAND flash address onto NFADDR register.
4. Read/Write data while checking NAND flash status by NFSTAT register. R/nB signal should be checked before read operation or after program operation.

NAND FLASH MEMORY TIMING

![NAND Flash Memory Timing Diagram]

Figure 6-3. TACLS=0, TWRPH0=1, TWRPH1=0
PIN CONFIGURATION

D[7:0]: Data/Command/Address In/Out Port (shared with the data bus)
CLE: Command Latch Enable (Output)
ALE: Address Latch Enable (Output)
nFCE: NAND Flash Chip Enable (Output)
nFRE: NAND Flash Read Enable (Output)
nFWE: NAND Flash Write Enable (Output)
nWAIT: NAND Flash Ready/nBusy (Input)

BOOT AND NAND FLASH CONFIGURATIONS

1. OM[1:0] = 00b: Enable NAND Flash controller auto boot mode
2. NAND Flash memory page size should be 512Bytes.
3. NCON: NAND Flash memory address step selection
   0: 3 Step addressing
   1: 4 Step addressing

512-BYTE ECC PARITY CODE ASSIGNMENT TABLE

<table>
<thead>
<tr>
<th>ECC</th>
<th>DATA7</th>
<th>DATA6</th>
<th>DATA5</th>
<th>DATA4</th>
<th>DATA3</th>
<th>DATA2</th>
<th>DATA1</th>
<th>DATA0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC0</td>
<td>P64</td>
<td>P64'</td>
<td>P32</td>
<td>P32'</td>
<td>P16</td>
<td>P16'</td>
<td>P8</td>
<td>P8'</td>
</tr>
<tr>
<td>ECC1</td>
<td>P1024</td>
<td>P1024'</td>
<td>P512</td>
<td>P512'</td>
<td>P256</td>
<td>P256'</td>
<td>P128</td>
<td>P128'</td>
</tr>
<tr>
<td>ECC2</td>
<td>P4</td>
<td>P4'</td>
<td>P2</td>
<td>P2'</td>
<td>P1</td>
<td>P1'</td>
<td>P2048</td>
<td>P2048'</td>
</tr>
</tbody>
</table>

S3C2410 generates 512-Byte ECC Parity Code during Write/Read operation. ECC Parity Code consists of 3 Bytes per 512-Byte data.

24-bit ECC Parity Code = 18-bit Line parity + 6-bit Column Parity

ECC generator block executes the followings:

1. When MCU writes data to NAND, the ECC generator block generates ECC code.
2. When MCU reads data from NAND, the ECC generator block generates ECC code and compares it with pre-written ECC code.
## NAND FLASH MEMORY MAPPING

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF_FFFF</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>0x6000_0000</td>
<td>SFR Area</td>
<td></td>
</tr>
<tr>
<td>0x4800_0000</td>
<td>BootSRAM (4KBytes)</td>
<td></td>
</tr>
<tr>
<td>0x4000_0000</td>
<td>SDRAM (BANK7, nGCS7)</td>
<td></td>
</tr>
<tr>
<td>0x3800_0000</td>
<td>SDRAM (BANK6, nGCS6)</td>
<td></td>
</tr>
<tr>
<td>0x3000_0000</td>
<td>SROM (BANK5, nGCS5)</td>
<td></td>
</tr>
<tr>
<td>0x2800_0000</td>
<td>SROM (BANK4, nGCS4)</td>
<td></td>
</tr>
<tr>
<td>0x2000_0000</td>
<td>SROM (BANK3, nGCS3)</td>
<td></td>
</tr>
<tr>
<td>0x1800_0000</td>
<td>SROM (BANK2, nGCS2)</td>
<td></td>
</tr>
<tr>
<td>0x1000_0000</td>
<td>SROM (BANK1, nGCS1)</td>
<td></td>
</tr>
<tr>
<td>0x0800_0000</td>
<td>SROM (BANK0, nGCS0)</td>
<td></td>
</tr>
<tr>
<td>0x0000_0000</td>
<td>BootSRAM (4KBytes)</td>
<td></td>
</tr>
</tbody>
</table>

OM[1:0] == 01, 10  

- a) Not using NAND flash for booting ROM

OM[1:0] == 00  

- a) Using NAND flash for booting ROM

**NOTE:** SFR means Special Function Register.

---

**Figure 6-4. NAND Flash Memory Mapping**
SPECIAL FUNCTION REGISTERS

NAND FLASH CONFIGURATION (NFCONF) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFCONF</td>
<td>0x4E000000</td>
<td>R/W</td>
<td>NAND Flash configuration</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NFCONF</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
</table>
| Enable/Disable | [15] | NAND Flash controller enable/disable  
|             |          | 0 = Disable NAND Flash Controller  
|             |          | 1 = Enable NAND Flash Controller  
|             |          | After auto-boot, this bit is cleared to 0 automatically. For the access to the NAND flash memory, this bit must be set. | 0             |
| Reserved   | [14:13]  | Reserved                                                                       | –             |
| Initialize ECC | [12] | Initialize ECC decoder/encoder  
|             |          | 0 : Not initialize ECC  
|             |          | 1 : Initialize ECC  
|             |          | (S3C2410 supports only 512-Byte ECC checking, so it is required to set ECC initialized per 512 Bytes.) | 0             |
| NAND Flash Memory chip enable | [11] | NAND Flash Memory nFCE control  
|             |          | 0 : NAND flash nFCE = L (active)  
|             |          | 1 : NAND flash nFCE = H (inactive)  
|             |          | (After auto-boot, nFCE will be inactive.) | –             |
| TACLS      | [10:8]   | CLE & ALE duration setting value (0~7)  
|             |          | Duration = HCLK * (TACLS + 1) | –             |
| TWRPH0     | [6:4]    | TWRPH0 duration setting value (0~7)  
|             |          | Duration = HCLK * (TWRPH0 + 1) | 0             |
| TWRPH1     | [2:0]    | TWRPH1 duration setting value (0~7)  
|             |          | Duration = HCLK * (TWRPH1 + 1) | 0             |
### NAND FLASH COMMAND SET (NFCMD) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFCMD</td>
<td>0x4E000004</td>
<td>R/W</td>
<td>NAND flash command set register</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NFCMD</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[15:8]</td>
<td>Reserved</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>[7:0]</td>
<td>NAND Flash memory command value</td>
<td>0x00</td>
</tr>
</tbody>
</table>

### NAND FLASH ADDRESS SET (NFADDR) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFADDR</td>
<td>0x4E000008</td>
<td>R/W</td>
<td>NAND flash address set register</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NFADDR</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[15:8]</td>
<td>Reserved</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>[7:0]</td>
<td>NAND flash memory address value</td>
<td>0x00</td>
</tr>
</tbody>
</table>

### NAND FLASH DATA (NFDATA) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFDATA</td>
<td>0x4E00000C</td>
<td>R/W</td>
<td>NAND flash data register</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NFDATA</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[15:8]</td>
<td>Reserved</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>[7:0]</td>
<td>NAND Flash read/program data value</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In case of write: Programming data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In case of read: Read data.</td>
<td></td>
</tr>
</tbody>
</table>
### NAND FLASH OPERATION STATUS (NFSTAT) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFSTAT</td>
<td>0x4E000010</td>
<td>R</td>
<td>NAND Flash operation status</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NFSTAT</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[16:1]</td>
<td>Reserved</td>
<td>–</td>
</tr>
<tr>
<td>RnB</td>
<td>[0]</td>
<td>NAND Flash memory ready/busy status. (This signal is checked through R/nB pin.)&lt;br&gt;0 = NAND Flash memory busy&lt;br&gt;1 = NAND Flash memory ready to operate</td>
<td>–</td>
</tr>
</tbody>
</table>

### NAND FLASH ECC (NFECC) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFECC</td>
<td>0x4E000014</td>
<td>R</td>
<td>NAND Flash ECC (Error Correction Code) register</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NFECC</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC2</td>
<td>[23:16]</td>
<td>Error Correction Code #2</td>
<td>–</td>
</tr>
<tr>
<td>ECC1</td>
<td>[15:8]</td>
<td>Error Correction Code #1</td>
<td>–</td>
</tr>
<tr>
<td>ECC0</td>
<td>[7:0]</td>
<td>Error Correction Code #0</td>
<td>–</td>
</tr>
</tbody>
</table>

**Known Problems**

- Problem: NAND flash controller can’t be accessed by DMA.
- Solution: Instead of DMA, use LDM/STM Instructions like our boot loader example code.
OVERVIEW

The clock & power management block consists of three parts: clock control, USB control, and power control.

The clock control logic in S3C2410X can generate the required clock signals including FCLK for CPU, HCLK for the AHB bus peripherals, and PCLK for the APB bus peripherals. The S3C2410X has two Phase Locked Loops (PLLs): one for FCLK, HCLK, and PCLK, and the other dedicated for USB block (48Mhz). The clock control logic can make slow clocks without PLL and connect/disconnect the clock to each peripheral block by software, which will reduce the power consumption.

For the power control logic, the S3C2410X has various power management schemes to keep optimal power consumption for a given task. The power management block in the S3C2410X can activate four modes: NORMAL mode, SLOW mode, IDLE mode, and Power-OFF mode.

NORMAL mode: the block supplies clocks to CPU as well as all peripherals in the S3C2410X. In this mode, the power consumption will be maximized when all peripherals are turned on. It allows the user to control the operation of peripherals by software. For example, if a timer is not needed, the user can disconnect the clock to the timer to reduce power consumption.

SLOW mode: Non-PLL mode. Unlike the Normal mode, the Slow mode uses an external clock (XTIpll or EXTCLK) directly as FCLK in the S3C2410X without PLL. In this mode, the power consumption depends on the frequency of the external clock only. The power consumption due to PLL is excluded.

IDLE mode: the block disconnects clocks (FCLK) only to the CPU core while it supplies clocks to all other peripherals. The IDLE mode results in reduced power consumption due to CPU core. Any interrupt request to CPU can be woken up from the Idle mode.

Power-OFF mode: the block disconnects the internal power. So, there occurs no power consumption due to CPU and the internal logic except the wake-up logic in this mode. Activating the Power-OFF mode requires two independent power sources. One of the two power sources supplies the power for the wake-up logic. The other one supplies other internal logics including CPU, and should be controlled for power on/off. In the Power-OFF mode, the second power supply source for the CPU and internal logics will be turned off. The wakeup from Power-OFF mode can be issued by the EINT[15:0] or by RTC alarm interrupt.
FUNCTIONAL DESCRIPTION

CLOCK ARCHITECTURE

Figure 7-1 shows a block diagram of the clock architecture. The main clock source comes from an external crystal (XTipll) or an external clock (EXTCLK). The clock generator includes an oscillator (Oscillation Amplifier), which is connected to an external crystal, and also has two PLLs (Phase-Locked-Loop), which generate the high frequency clock required in the S3C2410X.

CLOCK SOURCE SELECTION

Table 7-1 shows the relationship between the combination of mode control pins (OM3 and OM2) and the selection of source clock for the S3C2410X. The OM[3:2] status is latched internally by referring the OM3 and OM2 pins at the rising edge of nRESET.

<table>
<thead>
<tr>
<th>Mode OM[3:2]</th>
<th>MP LL State</th>
<th>UPLL State</th>
<th>Main Clock source</th>
<th>USB Clock Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>On</td>
<td>On</td>
<td>Crystal</td>
<td>Crystal</td>
</tr>
<tr>
<td>01</td>
<td>On</td>
<td>On</td>
<td>Crystal</td>
<td>EXTCLK</td>
</tr>
<tr>
<td>10</td>
<td>On</td>
<td>On</td>
<td>EXTCLK</td>
<td>Crystal</td>
</tr>
<tr>
<td>11</td>
<td>On</td>
<td>On</td>
<td>EXTCLK</td>
<td>EXTCLK</td>
</tr>
</tbody>
</table>

NOTES:
1. Although the MP LL starts just after a reset, the MP LL output (MPLL) is not used as the system clock until the software writes valid settings to the MP LLCON register. Before this valid setting, the clock from external crystal or EXTCLK source will be used as the system clock directly. Even if the user does not want to change the default value of MP LLCON register, the user should write the same value into MP LLCON register.
2. OM[3:2] is used to determine a test mode when OM[1:0] is 11.
Figure 7-1. Clock Generator Block Diagram
PHASE LOCKED LOOP (PLL)

The MPLL within the clock generator, as a circuit, synchronizes an output signal with a reference input signal in frequency and phase. In this application, it includes the following basic blocks as shown in Figure 7-2: the Voltage Controlled Oscillator (VCO) to generate the output frequency proportional to input DC voltage, the divider P to divide the input frequency (Fin) by p, the divider M to divide the VCO output frequency by m which is input to Phase Frequency Detector (PFD), the divider S to divide the VCO output frequency by s which is Mpll (the output frequency from MPLL block), the phase difference detector, the charge pump, and the loop filter. The output clock frequency M pll is related to the reference input clock frequency Fin by the following equation:

\[ M_{\text{pll}} = \frac{m \times \text{Fin}}{(p \times 2^s)} \]

\[ m = M \text{(the value for divider M)} + 8, \quad p = P \text{(the value for divider P)} + 2 \]

The UPLL within the clock generator is the same as the MPLL in every aspect.

The following sections describe the operation of the PLL, including the phase difference detector, the charge pump, the Voltage controlled oscillator (VCO), and the loop filter.

Phase Difference Detector (PFD)

The PFD monitors the phase difference between Fref and Fvco, and generates a control signal (tracking signal) when it detects a difference. The Fref means the reference frequency as shown in the Figure 7-2.

Charge Pump (PUMP)

The charge pump converts PFD control signals into a proportional charge in voltage across the external filter that drives the VCO.

Loop Filter

The control signal, which the PFD generates for the charge pump, may generate large excursions (ripples) each time the Fvco is compared to the Fref. To avoid overloading the VCO, a low pass filter samples and filters the high-frequency components out of the control signal. The filter is typically a single-pole RC filter with a resistor and a capacitor.

Voltage Controlled Oscillator (VCO)

The output voltage from the loop filter drives the VCO, causing its oscillation frequency to increase or decrease linearly as a function of variations in average voltage. When the Fvco matches Fref in terms of frequency as well as phase, the PFD stops sending control signals to the charge pump, which in turn stabilizes the input voltage to the loop filter. The VCO frequency then remains constant, and the PLL remains fixed onto the system clock.

Usual Conditions for PLL & Clock Generator

PLL & Clock Generator generally uses the following conditions.

<table>
<thead>
<tr>
<th>Loop filter capacitance</th>
<th>5 pF</th>
</tr>
</thead>
<tbody>
<tr>
<td>External X-tal frequency</td>
<td>10 – 20 MHz (note)</td>
</tr>
<tr>
<td>External capacitance used for X-tal</td>
<td>15 – 22 pF</td>
</tr>
</tbody>
</table>

NOTES:
1. The value could be changed.
2. FCLK must be more than three times X-tal or EXTCLK (FCLK ≥ 3X-tal or 3EXTCLK)
Figure 7-2. PLL (Phase-Locked Loop) Block Diagram

Figure 7-3. Main Oscillator Circuit Examples
CLOCK CONTROL LOGIC

The clock control logic determines the clock source to be used, i.e., the PLL clock (Mpll) or the direct external clock (XTIpll or EXTCLK). When PLL is configured to a new frequency value, the clock control logic disables the FCLK until the PLL output is stabilized using the PLL locking time. The clock control logic is also activated at power-on reset and wakeup from power-down mode.

Power-On Reset (XTIpll)

Figure 7-4 shows the clock behavior during the power-on reset sequence. The crystal oscillator begins oscillation within several milliseconds. When nRESET is released after the stabilization of OSC (XTIpll) clock, the PLL starts to operate according to the default PLL configuration. However, PLL is commonly known to be unstable after power-on reset, so Fin is fed directly to FCLK instead of the Mpll (PLL output) before the software newly configures the PLLCON. Even if the user does not want to change the default value of PLLCON register after reset, the user should write the same value into PLLCON register by software.

The PLL restarts the lockup sequence toward the new frequency only after the software configures the PLL with a new frequency. FCLK can be configured as PLL output (Mpll) immediately after lock time.

---

Figure 7-4. Power-On Reset Sequence (when the external clock source is a crystal oscillator)
Change PLL Settings In Normal Operation Mode

During the operation of the S3C2410X in NORMAL mode, the user can change the frequency by writing the PMS value and the PLL lock time will be automatically inserted. During the lock time, the clock is not supplied to the internal blocks in the S3C2410X. Figure 7-5 shows the timing diagram.

![Figure 7-5. Changing Slow Clock by Setting PMS Value](image)

USB Clock Control

USB host interface and USB device interface needs 48Mhz clock. In the S3C2410X, the USB dedicated PLL (UPLL) generates 48Mhz for USB. UCLK does not fed until the PLL (UPLL) is configured.

<table>
<thead>
<tr>
<th>Condition</th>
<th>UCLK State</th>
<th>UPLL State</th>
</tr>
</thead>
<tbody>
<tr>
<td>After reset</td>
<td>XTlpll or EXTCLK</td>
<td>On</td>
</tr>
<tr>
<td>After configuring UPLL</td>
<td>L during PLL lock time</td>
<td>On</td>
</tr>
<tr>
<td></td>
<td>48MHz after PLL lock time</td>
<td></td>
</tr>
<tr>
<td>UPLL is turned off by CLKSLOW register</td>
<td>XTlpll or EXTCLK</td>
<td>Off</td>
</tr>
<tr>
<td>UPLL is turned on by CLKSLOW register</td>
<td>48MHz</td>
<td>On</td>
</tr>
</tbody>
</table>
FCLK, HCLK, and PCLK

FCLK is used by ARM920T.
HCLK is used for AHB bus, which is used by the ARM920T, the memory controller, the interrupt controller, the LCD controller, the DMA and the USB host block.
PCLK is used for APB bus, which is used by the peripherals such as WDT, IIS, I2C, PWM timer, MMC interface, ADC, UART, GPIO, RTC and SPI.

The S3C2410X supports selection of Dividing Ratio between FCLK, HCLK and PCLK. This ratio is determined by HDIVN and PDIVN of CLKDIVN control register.

<table>
<thead>
<tr>
<th>HDIVN</th>
<th>PDIVN</th>
<th>FCLK</th>
<th>HCLK</th>
<th>PCLK</th>
<th>Divide Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>FCLK</td>
<td>FCLK</td>
<td>FCLK</td>
<td>1 : 1 : 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Default)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>FCLK</td>
<td>FCLK</td>
<td>FCLK / 2</td>
<td>1 : 1 : 2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>FCLK</td>
<td>FCLK / 2</td>
<td>FCLK / 2</td>
<td>1 : 2 : 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>FCLK</td>
<td>FCLK / 2</td>
<td>FCLK / 4</td>
<td>1 : 2 : 4</td>
</tr>
</tbody>
</table>

After setting PMS value, it is required to set CLKDIVN register. The setting value of CLKDIVN will be valid after PLL lock time. The value is also available for reset and changing Power Management Mode.
The setting value can also be valid after 1.5 HCLK. Only, 1HCLK can validate the value of CLKDIVN register changed from Default (1:1:1) to other Divide Ratio (1:1:2, 1:2:2 and 1:2:4)

Figure 7-6. Changing CLKDIVN Register Value

NOTES
1. CLKDIVN should be set carefully not to exceed the limit of HCLK and PCLK.
2. If HDIVN=1, the CPU bus mode has to be changed from the fast bus mode to the asynchronous bus mode using following instructions.
   ```
   MMU_SetAsyncBusMode
   mrc p15,0,r0,c1,c0,0
   orr r0,r0,#R1_nF:OR:R1_iA
   mcr p15,0,r0,c1,c0,0
   ```
   If HDIVN=1 and the CPU bus mode is the fast bus mode, the CPU will operate by the HCLK. This feature can be used to change the CPU frequency as a half without affecting the HCLK and PCLK.
POWER MANAGEMENT

The power management block controls the system clocks by software for the reduction of power consumption in the S3C2410X. These schemes are related to PLL, clock control logics (FCLK, HCLK, and PCLK) and wakeup signals. Figure 7-7 shows the clock distribution of the S3C2410X.

The S3C2410X has four power modes. The following section describes each power management mode. The transition between the modes is not allowed freely. For available transitions among the modes, see Figure 7-8.

![Figure 7-7. The Clock Distribution Block Diagram](image-url)
Figure 7-8. Power Management State Diagram

Table 7-2. Clock and Power State in Each Power Mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>ARM920T</th>
<th>AHB Modules /WDT (1)</th>
<th>Power Management</th>
<th>GPIO</th>
<th>32.768kHz RTC clock</th>
<th>APB Modules &amp; USBH/LCD/NAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>SEL</td>
<td>O</td>
<td>SEL</td>
</tr>
<tr>
<td>IDLE</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>SEL</td>
<td>O</td>
<td>SEL</td>
</tr>
<tr>
<td>SLOW</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>SEL</td>
<td>O</td>
<td>SEL</td>
</tr>
<tr>
<td>POWER_OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>Wait for wake-up event</td>
<td>Previous state</td>
<td>O</td>
<td>OFF</td>
</tr>
</tbody>
</table>

NOTES:
1. USB host, LCD, and NAND are excluded.
2. WDT is excluded. RTC interface for CPU access is included.
3. SEL : selectable (O, X), O : enable, X : disable, OFF: power is turned off
NORMAL Mode

In normal mode, all peripherals and the basic blocks including power management block, the CPU core, the bus controller, the memory controller, the interrupt controller, DMA, and the external master may operate fully. But, the clock to each peripheral, except the basic blocks, can be stopped selectively by software to reduce the power consumption.

IDLE Mode

In IDLE mode, the clock to the CPU core is stopped except the bus controller, the memory controller, the interrupt controller, and the power management block. To exit the IDLE mode, EINT[23:0], or RTC alarm interrupt, or the other interrupts should be activated. (EINT is not available until GPIO block is turned on).

SLOW Mode (Non-PLL Mode)

Power consumption can be reduced in the SLOW mode by applying a slow clock and excluding the power consumption from the PLL. The FCLK is the frequency of divide_by_n of the input clock (XTIpll or EXTCLK) without PLL. The divider ratio is determined by SLOW_VAL in the CLKSLOW control register and CLKDIVN control register.

<table>
<thead>
<tr>
<th>SLOW_VAL</th>
<th>FCLK</th>
<th>1/1 Option (HDIVN=0)</th>
<th>1/2 Option (HDIVN=1)</th>
<th>1/1 Option (PDIVN=0)</th>
<th>1/2 Option (PDIVN=1)</th>
<th>UCLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>EXTCLK or XTI_pll / 1</td>
<td>EXTCLK or XTI_pll / 1</td>
<td>EXTCLK or XTI_pll / 2</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
<tr>
<td>0 0 1</td>
<td>EXTCLK or XTI_pll / 2</td>
<td>EXTCLK or XTI_pll / 2</td>
<td>EXTCLK or XTI_pll / 4</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
<tr>
<td>0 1 0</td>
<td>EXTCLK or XTI_pll / 4</td>
<td>EXTCLK or XTI_pll / 4</td>
<td>EXTCLK or XTI_pll / 8</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
<tr>
<td>0 1 1</td>
<td>EXTCLK or XTI_pll / 6</td>
<td>EXTCLK or XTI_pll / 6</td>
<td>EXTCLK or XTI_pll / 12</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
<tr>
<td>1 0 0</td>
<td>EXTCLK or XTI_pll / 8</td>
<td>EXTCLK or XTI_pll / 8</td>
<td>EXTCLK or XTI_pll / 16</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
<tr>
<td>1 0 1</td>
<td>EXTCLK or XTI_pll / 10</td>
<td>EXTCLK or XTI_pll / 10</td>
<td>EXTCLK or XTI_pll / 20</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
<tr>
<td>1 1 0</td>
<td>EXTCLK or XTI_pll / 12</td>
<td>EXTCLK or XTI_pll / 12</td>
<td>EXTCLK or XTI_pll / 24</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
<tr>
<td>1 1 1</td>
<td>EXTCLK or XTI_pll / 14</td>
<td>EXTCLK or XTI_pll / 14</td>
<td>EXTCLK or XTI_pll / 28</td>
<td>HCLK</td>
<td>HCLK / 2</td>
<td>48 MHz</td>
</tr>
</tbody>
</table>

In SLOW mode, PLL will be turned off to reduce the PLL power consumption. When the PLL is turned off in the SLOW mode and the user changes power mode from SLOW mode to NORMAL mode, the PLL needs clock stabilization time (PLL lock time). This PLL stabilization time is automatically inserted by the internal logic with lock time count register. The PLL stability time will take 150us after the PLL is turned on. During PLL lock time, the FCLK becomes SLOW clock.
Users can change the frequency by enabling SLOW mode bit in CLKSLOW register in PLL on state. The SLOW clock is generated during the SLOW mode. Figure 7-11 shows the timing diagram.

If the user switches from SLOW mode to Normal mode by disabling the SLOW_BIT in the CLKSLOW register after PLL lock time, the frequency is changed just after SLOW mode is disabled. Figure 7-12 shows the timing diagram.

---

**Figure 7-9. Issuing Exit_from_Slow_mode Command in PLL on State**

**Figure 7-10. Issuing Exit_from_Slow_mode Command After Lock Time**
If the user switches from SLOW mode to Normal mode by disabling SLOW_BIT and MPLL_OFF bit simultaneously in the CLKSLOW register, the frequency is changed just after the PLL lock time. Figure 7-13 shows the timing diagram.

![Timing Diagram](image)

Figure 7-11. Issuing Exit_from_Slow_mode Command and the Instant PLL_on Command Simultaneously
Power_OFF Mode

The block disconnects the internal power. So, there occurs no power consumption due to CPU and the internal logic except the wake-up logic in this mode. Activating the Power-OFF mode requires two independent power sources. One of the two power sources supplies the power for the wake-up logic. The other one supplies other internal logics including CPU, and should be controlled for power on/off. In the Power-OFF mode, the second power supply source for the CPU and internal logics will be turned off. The wakeup from Power-OFF mode can be issued by the EINT[15:0] or by RTC alarm interrupt.

Procedure to Enter Power_OFF mode

1. Set the GPIO configuration adequate for Power_OFF mode.
2. Mask all interrupts in the INTMSK register.
3. Configure the wake-up sources properly including RTC alarm.
   The bit of EINTMASK corresponding to the wake-up source has not to be masked in order to let the corresponding bit of SRCPND or EINTPEND set. Although a wake-up source is issued and the corresponding bit of EINTMASK is masked, the wake-up will occur and the corresponding bit of SRCPND or EINTPEND will not be set.
4. Set USB pads as suspend mode. (MISCCR[13:12]=11b)
5. Save some meaning values into GSTATUS3,4 register. These register are preserved during Power_OFF mode.
6. Configure MISCCR[1:0] for the pull-up resistors on the data bus,D[31:0]. If there is an external BUS holder, such as 74LVCH162245, turn off the pull-up resistors. If not, turn on the pull-up resistors
7. Stop LCD by clearing LCDCON1.ENVID bit.
8. Read rREFRESH and rCLKCON registers in order to fill the TLB.
9. Let SDRAM enter the self-refresh mode by setting the REFRESH[22]=1b.
10. Wait until SDRAM self-refresh is effective.
11. Set MISCCR[19:17]=111b to make SDRAM signals(SCLK0,SCLK1 and SCKE) protected during Power_OFF mode
12. Set the Power_OFF mode bit in the CLKCON register.
Procedure to Wake-up from Power_OFF mode

1. The internal reset signal will be asserted if one of the wake-up sources is issued. This reset duration is determined by the internal 16-bit counter logic and the reset assertion time is calculated as \( t_{\text{RST}} = \frac{65535}{\text{XTAL\_frequency}} \).

2. Check GSTATUS2[2] in order to know whether or not the power-up is caused by the wake-up from Power_OFF mode.


4. Configure the SDRAM memory controller.

5. Wait until the SDRAM self-refresh is released. Mostly SDRAM needs the refresh cycle of all SDRAM row.

6. The information in GSTATUS3,4 can be used for user’s own purpose because the value in GSTATUS3,4 has been preserved during Power_OFF mode.

7. – For EINT[3:0], check the SRCPND register.
   – For EINT[15:4], check the EINTPEND instead of SRCPND (SRCPND will not be set although some bits of EINTPEND are set.).
   – For alarm wake-up, check the RTC time because the RTC bit of SRCPND isn’t set at the alarm wake-up.
   – If there was the nBATT_FLT assertion during POWER_OFF mode, the corresponding bit of SRCPND has been set.

Pin States in Power_OFF Mode

The pin state of the Power_OFF mode is as follows;

<table>
<thead>
<tr>
<th>Pin Type</th>
<th>Pin Example</th>
<th>Pin States in Power_OFF Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO output pin</td>
<td>GPB0: output</td>
<td>Output (GPIO data register value is used.)</td>
</tr>
<tr>
<td>GPIO input pin</td>
<td>GPB0: input</td>
<td>Input</td>
</tr>
<tr>
<td>GPIO bi-directional pin</td>
<td>GPG6:SPIMOSI</td>
<td>Input</td>
</tr>
<tr>
<td>Function output pin</td>
<td>nGCS0</td>
<td>Output (the last output level is held.)</td>
</tr>
<tr>
<td>Function input pin</td>
<td>nWAIT</td>
<td>Input</td>
</tr>
</tbody>
</table>
Power Control of VDDi and VDDiarm

In Power_OFF mode, only VDDi and VDDiarm will be turned off, which is controlled by PWREN pin.

If PWREN signal is active (H), VDDi and VDDiarm are supplied by an external voltage regulator. If PWREN pin is inactive (L), the VDDi and VDDiarm are turned off.

**NOTE**

Although VDDi, VDDiarm, VDDi_MPLL and VDDi_UPLL may be turned off, the other power pins have to be supplied.

---

**Figure 7-12. Power-OFF Mode**

---
Signaling EINT[15:0] for Wakeup

The S3C2410X can be woken up from Power_OFF mode only if the following conditions are met.

a) Level signals (H or L) or edge signals (rising or falling or both) are asserted on EINTn input pin.
b) The EINTn pin has to be configured as EINT in the GPIO control register.
c) nBATT_FLT pin has to be H level. It is important to configure the EINTn in the GPIO control register as an external interrupt pins, considering the condition a) above.

Just after the wake-up, the corresponding EINTn pin will not be used for wakeup. This means that the pin can be used as an external interrupt request pin again.

Entering IDLE Mode

If CLKCON[2] is set to 1 to enter the IDLE mode, the S3C2410X will enter IDLE mode after some delay (until the power control logic receives ACK signal from the CPU wrapper).

PLL On/Off

The PLL can only be turned off for low power consumption in slow mode. If the PLL is turned off in any other mode, MCU operation is not guaranteed.

When the processor is in SLOW mode and tries to change its state into other state with the PLL turned on, then SLOW_BIT should be clear to move to another state after PLL stabilization.

Pull-up Resistors on the Data Bus and Power_OFF Mode

In Power_OFF mode, the data bus (D[31:0] or D[15:0] ) is in Hi-z state.

But, because of the characteristics of I/O pad, the data bus pull-up resistors have to be turned on for low power consumption in Power_OFF mode. D[31:0] pin pull-up resistors can be controlled by the GPIO control register (MISCCR). However, if there is an external bus holder, such as 74LVCH162245, on the data bus, turning off the data bus pull-up resistors will be reduce power consumption.
Output Port State and Power_OFF Mode
The output port should have a proper logic level in power off mode, which makes the current consumption minimized. If there is no load on an output port pin, H level is preferred. If output is L, the current will be consumed through the internal parasitic resistance; if the output is H, the current will not be consumed. For an output port, the current consumption can be reduced if the output state is H.

Battery Fault Signal(nBATT_FLT)
There are two functions in nBATT_FLT pin as follows;

— When CPU is not in Power-OFF mode, nBATT_FLT pin will cause the interrupt request. The interrupt attribute of the nBATT_FLT is L-level triggered.

— While CPU is in Power-OFF mode, assertion of the nBATT_FLT will prohibit the wake up from the power-down mode. So, Any wake-up source will be masked if nBATT_FLT is asserted, which is protecting the system malfunction of the low battery capacity

ADC Power Down
The ADC has an additional power-down bit in ADCCON. If the S3C2410X enters the Power_OFF mode, the ADC should enter its own power-down mode.

S/W Work-Around
After 'wake-up' from the Power_OFF mode by RTC_ALARM, the RTC source pending bit of the SRCPND register is not set. So, the ALARM date has to be checked after the wake-up from Power_OFF mode.
CLOCK GENERATOR & POWER MANAGEMENT SPECIAL REGISTER

LOCK TIME COUNT REGISTER (LOCKTIME)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKTIME</td>
<td>0x4C000000</td>
<td>R/W</td>
<td>PLL lock time count register</td>
<td>0xFFFFFFFFFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOCKTIME</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_LTIME</td>
<td>[23:12]</td>
<td>UPLL lock time count value for UCLK. (U_LTIME &gt; 150uS)</td>
<td>0xFFF</td>
</tr>
<tr>
<td>M_LTIME</td>
<td>[11:0]</td>
<td>MPLL lock time count value for FCLK, HCLK, and PCLK (M_LTIME &gt; 150uS)</td>
<td>0xFFF</td>
</tr>
</tbody>
</table>

PLL Control Register (MPLLCON and UPLLCON)

\[
\text{MPLL} = \left(\frac{m \times \text{Fin}}{(p \times 2^s)}\right)
\]

\[
m = (\text{MDIV} + 8), \quad p = (\text{PDIV} + 2), \quad s = \text{SDIV}
\]

PLL Value Selection Guide

1. \(\frac{\text{Fout}}{25 \times p} < 28.449e6/m < \frac{\text{Fin}}{10 \times p}\)
2. \(0.7 < 3.373/\sqrt{m} < 1.8\)
3. \(160e6 \leq \text{Fvco} \leq 400e6\)
4. \(20e6 \leq \text{Fout} \leq 300e6\) (The max. Fout of the PLL itself is 300Mhz)
5. \(\text{FCLK} \geq 3\times\text{tal or 3EXTCLK}\)

NOTE: Although there is the rule for choosing PLL value, we recommend only the values in the PLL value recommendation table. If you have to use another value, please contact us.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPLLCON</td>
<td>0x4C000004</td>
<td>R/W</td>
<td>MPLL configuration register</td>
<td>0x0005C080</td>
</tr>
<tr>
<td>UPLLCON</td>
<td>0x4C000008</td>
<td>R/W</td>
<td>UPLL configuration register</td>
<td>0x00028080</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PLLCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDIV</td>
<td>[19:12]</td>
<td>Main divider control</td>
<td>0x5C / 0x28</td>
</tr>
<tr>
<td>PDIV</td>
<td>[9:4]</td>
<td>Pre-divider control</td>
<td>0x08 / 0x08</td>
</tr>
<tr>
<td>SDIV</td>
<td>[1:0]</td>
<td>Post divider control</td>
<td>0x0 / 0x0</td>
</tr>
</tbody>
</table>

NOTE: When you set MPLL&UPLL values simultaneously, set MPLL value first and then UPLL value.
### PLL VALUE SELECTION TABLE

It is not easy to find a proper PLL value. So, we recommend referring to the following PLL value recommendation table.

<table>
<thead>
<tr>
<th>Input Frequency</th>
<th>Output Frequency</th>
<th>MDIV</th>
<th>PDIV</th>
<th>SDIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.00MHz</td>
<td>11.289MHz</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>16.934MHz</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>22.50MHz</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>33.75MHz</td>
<td>82 (0x52)</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>45.00MHz</td>
<td>82 (0x52)</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>50.70MHz</td>
<td>161 (0xa1)</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>48.00MHz (note)</td>
<td>120 (0x78)</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>56.25MHz</td>
<td>142 (0x8e)</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>67.50MHz</td>
<td>82 (0x52)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>79.00MHz</td>
<td>71 (0x47)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>84.75MHz</td>
<td>105 (0x69)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>90.00MHz</td>
<td>112 (0x70)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>101.25MHz</td>
<td>127 (0x7f)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>113.00MHz</td>
<td>105 (0x69)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>118.50MHz</td>
<td>150 (0x96)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>124.00MHz</td>
<td>116 (0x74)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>135.00MHz</td>
<td>82 (0x52)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>147.00MHz</td>
<td>90 (0x5a)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>152.00MHz</td>
<td>68 (0x44)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>158.00MHz</td>
<td>71 (0x47)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>170.00MHz</td>
<td>77 (0x4d)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>180.00MHz</td>
<td>82 (0x52)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>186.00MHz</td>
<td>85 (0x55)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>192.00MHz</td>
<td>88 (0x58)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12.00MHz</td>
<td>202.80MHz</td>
<td>161 (0xa1)</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**NOTE:** The 48.00MHz output is used for UPLLCON register.
### CLOCK CONTROL REGISTER (CLKCON)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKCON</td>
<td>0x4C00000C</td>
<td>R/W</td>
<td>Clock generator control register</td>
<td>0x7FFF0</td>
</tr>
</tbody>
</table>

#### CLKCON Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI [18]</td>
<td>Control PCLK into SPI block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>IIS [17]</td>
<td>Control PCLK into IIS block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>IIC [16]</td>
<td>Control PCLK into IIC block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>ADC(&amp;Touch Screen) [15]</td>
<td>Control PCLK into ADC block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>RTC [14]</td>
<td>Control PCLK into RTC control block. Even if this bit is cleared to 0, RTC timer is alive. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>GPIO [13]</td>
<td>Control PCLK into GPIO block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>UART2 [12]</td>
<td>Control PCLK into UART2 block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>UART1 [11]</td>
<td>Control PCLK into UART1 block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>UART0 [10]</td>
<td>Control PCLK into UART0 block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>SDI [9]</td>
<td>Control PCLK into SDI interface block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>PWMTIMER [8]</td>
<td>Control PCLK into PWMTIMER block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>USB device [7]</td>
<td>Control PCLK into USB device block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>USB host [6]</td>
<td>Control HCLK into USB host block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>LCDC [5]</td>
<td>Control HCLK into LCDC block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>NAND Flash Controller [4]</td>
<td>Control HCLK into NAND Flash Controller block. 0 = Disable, 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>POWER-OFF [3]</td>
<td>Control Power Off mode of S3C2410. 0 = Disable, 1 = Transition to Power_OFF mode</td>
<td>0</td>
</tr>
<tr>
<td>IDLE BIT [2]</td>
<td>Enter IDLE mode. This bit is not cleared automatically. 0 = Disable, 1 = Transition to IDLE mode</td>
<td>0</td>
</tr>
<tr>
<td>Reserved [1]</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td>SM_BIT [0]</td>
<td>SPECIAL mode. '0' is recommended normally. This bit can be used to enter SPECIAL mode in only the special condition, OM3=1 &amp; wake-up by nRESET. Please contact us to use this bit.</td>
<td>0</td>
</tr>
</tbody>
</table>
## CLOCK SLOW CONTROL (CLKSLOW) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKSLOW</td>
<td>0x4C000010</td>
<td>R/W</td>
<td>Slow clock control register</td>
<td>0x00000004</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCLK_ON</td>
<td>0: UCLK ON (UPLL is also turned on and the UPLL lock time is inserted automatically.)&lt;br&gt;1: UCLK OFF (UPLL is also turned off.)</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>MPDLL_OFF</td>
<td>0: PLL is turned on. After PLL stabilization time (minimum 150us), SLOW_BIT can be cleared to 0.&lt;br&gt;1: PLL is turned off. PLL is turned off only when SLOW_BIT is 1.</td>
<td>0</td>
</tr>
<tr>
<td>SLOW_BIT</td>
<td>0: FCLK = Mpll (MPLL output)&lt;br&gt;1: SLOW mode&lt;br&gt;FCLK = input clock / (2 x SLOW_VAL) (SLOW_VAL &gt; 0)&lt;br&gt;FCLK = input clock (SLOW_VAL = 0)&lt;br&gt;input clock = XTIpll or EXTCLK</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>SLOW_VAL</td>
<td>The divider value for the slow clock when SLOW_BIT is on.</td>
<td>0x4</td>
</tr>
</tbody>
</table>

## CLOCK DIVIDER CONTROL (CLKDIVN) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKDIVN</td>
<td>0x4C000014</td>
<td>R/W</td>
<td>Clock divider control register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Special bus clock ratio for the chip verification.</td>
<td>0</td>
</tr>
<tr>
<td>HDIVN</td>
<td>0: HCLK has the clock same as the FCLK.&lt;br&gt;1: HCLK has the clock same as the FCLK/2.</td>
<td>0</td>
</tr>
<tr>
<td>PDIVN</td>
<td>0: PCLK has the clock same as the HCLK.&lt;br&gt;1: PCLK has the clock same as the HCLK/2.</td>
<td>0</td>
</tr>
</tbody>
</table>
OVERVIEW

The S3C2410X supports four-channel DMA controller that is located between the system bus and the peripheral bus. Each channel of DMA controller can perform data movements between devices in the system bus and/or peripheral bus with no restrictions. In other words, each channel can handle the following four cases: 1) both source and destination are in the system bus, 2) the source is in the system bus while the destination is in the peripheral bus, 3) the source is in the peripheral bus while the destination is in the system bus, and 4) both source and destination are in the peripheral bus.

The main advantage of the DMA is that it can transfer the data without CPU intervention. The operation of DMA can be initiated by software, or requests from internal peripherals or external request pins.
DMA REQUEST SOURCES

Each channel of the DMA controller can select one of DMA request source among four DMA sources if H/W DMA request mode is selected by DCON register. (Note that if S/W request mode is selected, this DMA request sources have no meaning at all.) Table 8-1 shows four DMA sources for each channel.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Source0</th>
<th>Source1</th>
<th>Source2</th>
<th>Source3</th>
<th>Source4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ch-0</td>
<td>nXDREQ0</td>
<td>UART0</td>
<td>SDI</td>
<td>Timer</td>
<td>USB device EP1</td>
</tr>
<tr>
<td>Ch-1</td>
<td>nXDREQ1</td>
<td>UART1</td>
<td>I2SSDI</td>
<td>SPI0</td>
<td>USB device EP2</td>
</tr>
<tr>
<td>Ch-2</td>
<td>I2SSDO</td>
<td>I2SSDI</td>
<td>SDI</td>
<td>Timer</td>
<td>USB device EP3</td>
</tr>
<tr>
<td>Ch-3</td>
<td>UART2</td>
<td>SDI</td>
<td>SPI1</td>
<td>Timer</td>
<td>USB device EP4</td>
</tr>
</tbody>
</table>

Here, nXDREQ0 and nXDREQ1 represent two external sources (External Devices), and I2SSDO and I2SSDI represent IIS transmitting and receiving, respectively.

DMA OPERATION

DMA uses three-state FSM (finite state machine) for its operation, which is described in the three following steps:

State-1. As an initial state, the DMA waits for a DMA request. If it comes, it goes to state-2. At this state, DMA ACK and INT REQ are 0.

State-2. In this state, DMA ACK becomes 1 and the counter (CURR_TC) is loaded from DCON[19:0] register. Note that the DMA ACK remains 1 until it is cleared later.

State-3. In this state, sub-FSM handling the atomic operation of DMA is initiated. The sub-FSM reads the data from the source address and then writes it to destination address. In this operation, data size and transfer size (single or burst) are considered. This operation is repeated until the counter (CURR_TC) becomes 0 in Whole service mode, while performed only once in Single service mode. The main FSM (this FSM) counts down the CURR_TC when the sub-FSM finishes each of atomic operation. In addition, this main FSM asserts the INT REQ signal when CURR_TC becomes 0 and the interrupt setting of DCON[29] register is set to 1. In addition, it clears DMA ACK if one of the following conditions is met.

1) CURR_TC becomes 0 in the Whole service mode
2) Atomic operation finishes in the Single service mode.

Note that in the Single service mode, these three states of main FSM are performed and then stops, and waits for another DMA REQ. And if DMA REQ comes in, all three states are repeated. Therefore, DMA ACK is asserted and then deasserted for each atomic transfer. In contrast, in the Whole service mode, main FSM waits at state-3 until CURR_TC becomes 0. Therefore, DMA ACK is asserted during all the transfers and then deasserted when TC reaches 0.

However, INT REQ is asserted only if CURR_TC becomes 0 regardless of the service mode (Single service mode or Whole service mode).
EXTERNAL DMA DREQ/DACK PROTOCOL

There are three types of external DMA request/acknowledge protocols (Single service Demand, Single service Handshake and Whole service Handshake mode). Each type defines how the signals like DMA request and acknowledge are related to these protocols.

Basic DMA Timing

The DMA service means performing paired Reads and Writes cycles during DMA operation, which can make one DMA operation. Figure 8-1 shows the basic Timing in the DMA operation of the S3C2410X.

— The setup time and the delay time of XnXDREQ and XnXDACK are the same in all the modes.
— If the completion of XnXDREQ meets its setup time, it is synchronized twice and then XnXDACK is asserted.
— After assertion of XnXDACK, DMA requests the bus and if it gets the bus it performs its operations. XnXDACK is deasserted when DMA operation is completed.

![Figure 8-1. Basic DMA Timing Diagram](image-url)

Table 8-2. DMA Controller Module Signal Timing Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ.</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXternal Request Setup</td>
<td>$t_{\text{XRS}}$</td>
<td>2</td>
<td>–</td>
<td>6</td>
<td>ns</td>
</tr>
<tr>
<td>aCcess to Ack Delay when Low transition</td>
<td>$t_{\text{CADL}}$</td>
<td>9</td>
<td>–</td>
<td>11</td>
<td>ns</td>
</tr>
<tr>
<td>aCcess to Ack Delay when High transition</td>
<td>$t_{\text{CADH}}$</td>
<td>9</td>
<td>–</td>
<td>11</td>
<td>ns</td>
</tr>
<tr>
<td>eXternal Request Delay</td>
<td>$t_{\text{XAD}}$</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>SCLK</td>
</tr>
</tbody>
</table>
Demand/Handshake Mode Comparison

Demand and Handshake modes are related to the protocol between XnXDREQ and XnXDACK. Figure 8-2 shows the differences between the two modes.

At the end of one transfer (Single/Burst transfer), DMA checks the state of double-synched XnXDREQ.

**Demand Mode**
- If XnXDREQ remains asserted, the next transfer starts immediately. Otherwise it waits for XnXDREQ to be asserted.

**Handshake Mode**
- If XnXDREQ is deasserted, DMA deasserts XnXDACK in 2cycles. Otherwise it waits until XnXDREQ is deasserted.

**CAUTION**

XnXDREQ has to be asserted (low) only after the deassertion (high) of XnXDACK.

---

**Figure 8-2. Demand/Handshake Mode Comparison**
Transfer Size
— There are two different transfer sizes; unit and Burst 4.
— DMA holds the bus firmly during the transfer of the chunk of data. Thus, other bus masters cannot get the bus.

Burst 4 Transfer Size
Four sequential Reads and Writes respectively are performed in the Burst 4 Transfer.

NOTE: Unit Transfer size: One read and one write are performed.

Figure 8-3. Burst 4 Transfer Size
EXAMPLES

Single service in Demand Mode with Unit Transfer Size

The assertion of XnXDREQ is need for every unit transfer (Single service mode). The operation continues while the XnXDREQ is asserted (Demand mode), and one pair of Read and Write (Single transfer size) is performed.

Figure 8-4. Single service in Demand Mode with Unit Transfer Size

Single service in Handshake Mode with Unit Transfer Size

Figure 8-5. Single service in Handshake Mode with Unit Transfer Size

Whole service in Handshake Mode with Unit Transfer Size

Figure 8-6. Whole service in Handshake Mode with Unit Transfer Size
DMA SPECIAL REGISTERS

Each DMA channel has nine control registers (36 in total since there are four channels for DMA controller). Six of the control registers control the DMA transfer, and other three ones monitor the status of DMA controller. The details of those registers are as follows.

DMA INITIAL SOURCE (DISRC) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISRC0</td>
<td>0x4B000000</td>
<td>R/W</td>
<td>DMA 0 initial source register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DISRC1</td>
<td>0x4B000040</td>
<td>R/W</td>
<td>DMA 1 initial source register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DISRC2</td>
<td>0x4B000080</td>
<td>R/W</td>
<td>DMA 2 initial source register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DISRC3</td>
<td>0x4B0000C0</td>
<td>R/W</td>
<td>DMA 3 initial source register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISRCn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ADDR</td>
<td>[30:0]</td>
<td>Base address (start address) of source data to transfer. This bit value will be loaded into CURR_SRC only if the CURR_SRC is 0 and the DMA ACK is 1.</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

DMA INITIAL SOURCE CONTROL (DISRCC) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISRCC0</td>
<td>0x4B000004</td>
<td>R/W</td>
<td>DMA 0 initial source control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DISRCC1</td>
<td>0x4B000044</td>
<td>R/W</td>
<td>DMA 1 initial source control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DISRCC2</td>
<td>0x4B000084</td>
<td>R/W</td>
<td>DMA 2 initial source control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DISRCC3</td>
<td>0x4B0000C4</td>
<td>R/W</td>
<td>DMA 3 initial source control register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISRCCn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>[1]</td>
<td>Bit 1 is used to select the location of source. 0: the source is in the system bus (AHB). 1: the source is in the peripheral bus (APB).</td>
<td>0</td>
</tr>
</tbody>
</table>
| INC      | [0] | Bit 0 is used to select the address increment. 0 = Increment 1 = Fixed  
|          |     | If it is 0, the address is increased by its data size after each transfer in burst and single transfer mode. If it is 1, the address is not changed after the transfer. (In the burst mode, address is increased during the burst transfer, but the address is recovered to its first value after the transfer.) | 0             |
### DMA INITIAL DESTINATION (DIDST) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIDST0</td>
<td>0x4B000008</td>
<td>R/W</td>
<td>DMA 0 initial destination register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DIDST1</td>
<td>0x4B000048</td>
<td>R/W</td>
<td>DMA 1 initial destination register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DIDST2</td>
<td>0x4B000088</td>
<td>R/W</td>
<td>DMA 2 initial destination register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DIDST3</td>
<td>0x4B0000B8</td>
<td>R/W</td>
<td>DMA 3 initial destination register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

**DIDSTn Bit Description**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_ADDR [30:0]</td>
<td>Base address (start address) of destination for the transfer. This bit value will be loaded into CURR_SRC only if the CURR_DST is 0 and the DMA ACK is 1.</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### DMA INITIAL DESTINATION CONTROL (DIDSTC) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIDSTC0</td>
<td>0x4B00000C</td>
<td>R/W</td>
<td>DMA 0 initial destination control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DIDSTC1</td>
<td>0x4B00004C</td>
<td>R/W</td>
<td>DMA 1 initial destination control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DIDSTC2</td>
<td>0x4B00008C</td>
<td>R/W</td>
<td>DMA 2 initial destination control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DIDSTC3</td>
<td>0x4B0000CC</td>
<td>R/W</td>
<td>DMA 3 initial destination control register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

**DIDSTCn Bit Description**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC [1]</td>
<td>Bit 1 is used to select the location of destination. 0: the destination is in the system bus (AHB). 1: the destination is in the peripheral bus (APB).</td>
<td>0</td>
</tr>
<tr>
<td>INC [0]</td>
<td>Bit 0 is used to select the address increment. 0 = Increment 1 = Fixed If it is 0, the address is increased by its data size after each transfer in burst and single transfer mode. If it is 1, the address is not changed after the transfer. (In the burst mode, address is increased during the burst transfer, but the address is recovered to its first value after the transfer.)</td>
<td>0</td>
</tr>
</tbody>
</table>
DMA CONTROL (DCON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCON0</td>
<td>0x4B000010</td>
<td>R/W</td>
<td>DMA 0 control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCON1</td>
<td>0x4B000050</td>
<td>R/W</td>
<td>DMA 1 control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCON2</td>
<td>0x4B000090</td>
<td>R/W</td>
<td>DMA 2 control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCON3</td>
<td>0x4B0000D0</td>
<td>R/W</td>
<td>DMA 3 control register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DCONn Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMD_HS [31]</td>
<td>Select one between Demand mode and Handshake mode.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: Demand mode is selected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Handshake mode is selected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>In both modes, DMA controller starts its transfer and asserts DACK for a given asserted DREQ. The difference between the two modes is whether it waits for the deasserted DACK or not. In the Handshake mode, DMA controller waits for the deasserted DREQ before starting a new transfer. If it finds the deasserted DREQ, it deasserts DACK and waits for another asserted DREQ. In contrast, in the Demand mode, DMA controller does not wait until the DREQ is deasserted. It just deasserts DACK and then starts another transfer if DREQ is asserted. We recommend using Handshake mode for external DMA request sources to prevent unintended starts of new transfers.</td>
<td></td>
</tr>
<tr>
<td>SYNC [30]</td>
<td>Select DREQ/DACK synchronization.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: DREQ and DACK are synchronized to PCLK (APB clock).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: DREQ and DACK are synchronized to HCLK (AHB clock).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Therefore, for devices attached to AHB system bus, this bit has to be set to 1, while for those attached to APB system, it should be set to 0. For the devices attached to external systems, the user should select this bit depending on which the external system is synchronized with between AHB system and APB system.</td>
<td></td>
</tr>
<tr>
<td>INT [29]</td>
<td>Enable/Disable the interrupt setting for CURR_TC (terminal count)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: CURR_TC interrupt is disabled. The user has to view the transfer count in the status register (i.e. polling).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: interrupt request is generated when all the transfer is done (i.e. CURR_TC becomes 0).</td>
<td></td>
</tr>
<tr>
<td>TSZ [28]</td>
<td>Select the transfer size of an atomic transfer (i.e. transfer performed each time DMA owns the bus before releasing the bus).</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: a unit transfer is performed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: a burst transfer of length four is performed.</td>
<td></td>
</tr>
</tbody>
</table>
### DMA CONTROL (DCON) REGISTER (Continued)

<table>
<thead>
<tr>
<th>DCONn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVMODE</td>
<td>[27]</td>
<td>Select the service mode between Single service mode and Whole service mode.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Single service mode is selected in which after each atomic transfer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(single or burst of length four) DMA stops and waits for another DMA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>request.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Whole service mode is selected in which one request gets atomic</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>transfers to be repeated until the transfer count reaches to 0. In this</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode, additional request are not required.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note that even in the Whole service mode, DMA releases the bus after</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>each atomic transfer and then tries to re-get the bus to prevent starving</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>of other bus masters.</td>
<td></td>
</tr>
</tbody>
</table>

| HWSRCSEL | [26:24] | Select DMA request source for each DMA.                                      | 0             |
|          |         | DCON0: 000:nXDREQ0  001:UART0   010:SDI        011:Timer    100:USB device EP1 |               |
|          |         | DCON1: 000:nXDREQ1  001:UART1   010:SSDI     011:SPI       100:USB device EP2 |               |
|          |         | DCON2: 000:SSDO      001:SSDI   010:SDI        011:Timer    100:USB device EP3 |               |
|          |         | DCON3: 000:UART2     001:SSDI    010:SDI        011:Timer    100:USB device EP4 |               |
|          |         | These bits control the 4-1 MUX to select the DMA request source of each DMA.|               |
|          |         | These bits have meanings only if H/W request mode is selected by DCONn[23]. |               |

| SWHW_SEL | [23]   | Select the DMA source between software (S/W request mode) and hardware (H/W request mode). | 0             |
|          |       | 0: S/W request mode is selected and DMA is triggered by setting SW_TRIG bit of DMASKTRIG control register. |               |
|          |       | 1: DMA source selected by bit[26:24] triggers the DMA operation.                  |               |

| RELOAD   | [22]   | Set the reload on/off option.                                                    | 0             |
|          |       | 0: auto reload is performed when a current value of transfer count becomes 0 (i.e. all the required transfers are performed). |               |
|          |       | 1: DMA channel (DMA REQ) is turned off when a current value of transfer count becomes 0. The channel on/off bit (DMASKTRIGn[1]) is set to 0 (DREQ off) to prevent unintended further start of new DMA operation. |               |

| DSZ      | [21:20] | Data size to be transferred.                                                     | 00            |
|          |        | 00 = Byte                       01 = Half word                             |               |
|          |        | 10 = Word                      11 = reserved                              |               |

| TC       | [19:0] | Initial transfer count (or transfer beat).                                      | 00000         |
|          |        | Note that the actual number of bytes that are transferred is computed by the   |               |
|          |        | following equation: DSZ x TSZ x TC. Where, DSZ, TSZ (1 or 4), and TC represent data size (DCONn[21:20]), transfer size (DCONn[28]), and initial transfer count, respectively. |               |
|          |        | This value will be loaded into CURR_SRC only if the CURR_SRC is 0 and the DMA ACK is 1. |               |
### DMA STATUS (DSTAT) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSTAT0</td>
<td>0x4B000014</td>
<td>R</td>
<td>DMA 0 count register</td>
<td>000000h</td>
</tr>
<tr>
<td>DSTAT1</td>
<td>0x4B000054</td>
<td>R</td>
<td>DMA 1 count register</td>
<td>000000h</td>
</tr>
<tr>
<td>DSTAT2</td>
<td>0x4B000094</td>
<td>R</td>
<td>DMA 2 count register</td>
<td>000000h</td>
</tr>
<tr>
<td>DSTAT3</td>
<td>0x4B0000D4</td>
<td>R</td>
<td>DMA 3 count register</td>
<td>000000h</td>
</tr>
</tbody>
</table>

#### DSTATn Bit Description

| STAT     | [21:20] | Status of this DMA controller. |
|          |        | 00: Indicates that DMA controller is ready for another DMA request. |
|          |        | 01: Indicates that DMA controller is busy for transfers. |

| CURR_TC  | [19:0]  | Current value of transfer count. Note that transfer count is initially set to the value of DCONn[19:0] register and decreased by one at the end of every atomic transfer. |

### DMA CURRENT SOURCE (DCSRC) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCSRC0</td>
<td>0x4B000018</td>
<td>R</td>
<td>DMA 0 current Source Register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCSRC1</td>
<td>0x4B000058</td>
<td>R</td>
<td>DMA 1 current Source Register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCSRC2</td>
<td>0x4B000098</td>
<td>R</td>
<td>DMA 2 current Source Register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCSRC3</td>
<td>0x4B0000D8</td>
<td>R</td>
<td>DMA 3 current Source Register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

#### DCSRCn Bit Description

| CURR_SRC | [30:0]  | Current source address for DMAn |

<table>
<thead>
<tr>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
</tr>
</tbody>
</table>
CURRENT DESTINATION (DCDST) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCDST0</td>
<td>0x4B00001C</td>
<td>R</td>
<td>DMA 0 current destination register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCDST1</td>
<td>0x4B00005C</td>
<td>R</td>
<td>DMA 1 current destination register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCDST2</td>
<td>0x4B00009C</td>
<td>R</td>
<td>DMA 2 current destination register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>DCDST3</td>
<td>0x4B0000DC</td>
<td>R</td>
<td>DMA 3 current destination register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DCDSTn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURR_DST</td>
<td>[30:0]</td>
<td>Current destination address for DMA{n}</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
DMA MASK TRIGGER (DMASKTRIG) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMASKTRIG0</td>
<td>0x4B000020</td>
<td>R/W</td>
<td>DMA 0 mask trigger register</td>
<td>000</td>
</tr>
<tr>
<td>DMASKTRIG1</td>
<td>0x4B000060</td>
<td>R/W</td>
<td>DMA 1 mask trigger register</td>
<td>000</td>
</tr>
<tr>
<td>DMASKTRIG2</td>
<td>0x4B0000A0</td>
<td>R/W</td>
<td>DMA 2 mask trigger register</td>
<td>000</td>
</tr>
<tr>
<td>DMASKTRIG3</td>
<td>0x4B0000E0</td>
<td>R/W</td>
<td>DMA 3 mask trigger register</td>
<td>000</td>
</tr>
</tbody>
</table>

**DMASKTRIGn**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOP</td>
<td><strong>Stop the DMA operation.</strong></td>
</tr>
<tr>
<td></td>
<td>1: DMA stops as soon as the current atomic transfer ends. If there is no current running atomic transfer, DMA stops immediately. The CURR_Tc will be 0.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> Due to possible current atomic transfer, &quot;stop&quot; operation may take several cycles. The finish of the operation (i.e. actual stop time) can be detected as soon as the channel on/off bit (DMASKTRIGn[1]) is set to off. This stop is &quot;actual stop&quot;.</td>
</tr>
<tr>
<td>ON_OFF</td>
<td><strong>DMA channel on/off bit.</strong></td>
</tr>
<tr>
<td></td>
<td>0: DMA channel is turned off. (DMA request to this channel is ignored.)</td>
</tr>
<tr>
<td></td>
<td>1: DMA channel is turned on and the DMA request is handled. This bit is automatically set to off if we set the DCONn[22] bit to &quot;no auto reload&quot; and/or STOP bit of DMASKTRIGn to &quot;stop&quot;.</td>
</tr>
<tr>
<td></td>
<td>Note that when DCON[22] bit is &quot;no auto reload&quot;, this bit becomes 0 when CURR_Tc reaches 0. If the STOP bit is 1, this bit becomes 0 as soon as the current atomic transfer is completed.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> This bit should not be changed manually during DMA operations (i.e. this has to be changed only by using DCON[22] or STOP bit).</td>
</tr>
<tr>
<td>SW_TRIG</td>
<td><strong>Trigger the DMA channel in S/W request mode.</strong></td>
</tr>
<tr>
<td></td>
<td>1: it requests a DMA operation to this controller.</td>
</tr>
<tr>
<td></td>
<td>Note that this trigger gets effective after S/W request mode has to be selected (DCONn[23]) and channel ON_OFF bit has to be set to 1 (channel on). When DMA operation starts, this bit is cleared automatically.</td>
</tr>
</tbody>
</table>

**NOTE:** You can freely change the values of DISRC register, DIDST registers, and TC field of DCON register. Those changes take effect only after the finish of current transfer (i.e. when CURR_Tc becomes 0). On the other hand, any change made to other registers and/or fields takes immediate effect. Therefore, be careful in changing those registers and fields.
S/W Work-Around

The DMA auto-reload is occurred only when the DMA request is issued after the DMA counter reaches 0. So, the following code should be used in the DMA done interrupt handler before setting the DMA source address, destination address and counter register for the next auto-reload. This code will wait until the first DMA request is issued and the previous auto-reload value is loaded.

while((rDSTATn&0xfffff)==0);
I/O PORTS

OVERVIEW

The S3C2410X has 117 multi-functional input/output port pins. The ports are:

- Port A (GPA): 23-output port
- Port B (GPB): 11-input/output port
- Port C (GPC): 16-input/output port
- Port D (GPD): 16-input/output port
- Port E (GPE): 16-input/output port
- Port F (GPF): 8-input/output port
- Port G (GPG): 16-input/output port
- Port H (GPH): 11-input/output port

Each port can be easily configured by software to meet various system configurations and design requirements. You have to define which function of each pin is used before starting the main program. If a pin is not used for multiplexed functions, the pin can be configured as I/O ports.

Initial pin states are configured seamlessly to avoid problems.
Table 9-1. S3C2410X Port Configuration (Sheet 1 of 5)

<table>
<thead>
<tr>
<th>Port A</th>
<th>Selectable Pin Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA22</td>
<td>Output only nFCE</td>
</tr>
<tr>
<td>GPA21</td>
<td>Output only nRSTOUT</td>
</tr>
<tr>
<td>GPA20</td>
<td>Output only nFRE</td>
</tr>
<tr>
<td>GPA19</td>
<td>Output only nFWE</td>
</tr>
<tr>
<td>GPA18</td>
<td>Output only ALE</td>
</tr>
<tr>
<td>GPA17</td>
<td>Output only CLE</td>
</tr>
<tr>
<td>GPA16</td>
<td>Output only nGCS5</td>
</tr>
<tr>
<td>GPA15</td>
<td>Output only nGCS4</td>
</tr>
<tr>
<td>GPA14</td>
<td>Output only nGCS3</td>
</tr>
<tr>
<td>GPA13</td>
<td>Output only nGCS2</td>
</tr>
<tr>
<td>GPA12</td>
<td>Output only nGCS1</td>
</tr>
<tr>
<td>GPA11</td>
<td>Output only ADDR26</td>
</tr>
<tr>
<td>GPA10</td>
<td>Output only ADDR25</td>
</tr>
<tr>
<td>GPA9</td>
<td>Output only ADDR24</td>
</tr>
<tr>
<td>GPA8</td>
<td>Output only ADDR23</td>
</tr>
<tr>
<td>GPA7</td>
<td>Output only ADDR22</td>
</tr>
<tr>
<td>GPA6</td>
<td>Output only ADDR21</td>
</tr>
<tr>
<td>GPA5</td>
<td>Output only ADDR20</td>
</tr>
<tr>
<td>GPA4</td>
<td>Output only ADDR19</td>
</tr>
<tr>
<td>GPA3</td>
<td>Output only ADDR18</td>
</tr>
<tr>
<td>GPA2</td>
<td>Output only ADDR17</td>
</tr>
<tr>
<td>GPA1</td>
<td>Output only ADDR16</td>
</tr>
<tr>
<td>GPA0</td>
<td>Output only ADDR0</td>
</tr>
</tbody>
</table>
### Table 9-1. S3C2410X Port Configuration (Sheet 2 of 5)

<table>
<thead>
<tr>
<th>Port B</th>
<th>Selectable Pin Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPB10</td>
<td>Input/output</td>
<td>nXDREQ0</td>
</tr>
<tr>
<td>GPB9</td>
<td>Input/output</td>
<td>nXDACK0</td>
</tr>
<tr>
<td>GPB8</td>
<td>Input/output</td>
<td>nXDREQ1</td>
</tr>
<tr>
<td>GPB7</td>
<td>Input/output</td>
<td>nXDACK1</td>
</tr>
<tr>
<td>GPB6</td>
<td>Input/output</td>
<td>nXBREQ</td>
</tr>
<tr>
<td>GPB5</td>
<td>Input/output</td>
<td>nXBACK</td>
</tr>
<tr>
<td>GPB4</td>
<td>Input/output</td>
<td>TCLK0</td>
</tr>
<tr>
<td>GPB3</td>
<td>Input/output</td>
<td>TOUT3</td>
</tr>
<tr>
<td>GPB2</td>
<td>Input/output</td>
<td>TOUT2</td>
</tr>
<tr>
<td>GPB1</td>
<td>Input/output</td>
<td>TOUT1</td>
</tr>
<tr>
<td>GPB0</td>
<td>Input/output</td>
<td>TOUT0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port C</th>
<th>Selectable Pin Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPC15</td>
<td>Input/output</td>
<td>VD7</td>
</tr>
<tr>
<td>GPC14</td>
<td>Input/output</td>
<td>VD6</td>
</tr>
<tr>
<td>GPC13</td>
<td>Input/output</td>
<td>VD5</td>
</tr>
<tr>
<td>GPC12</td>
<td>Input/output</td>
<td>VD4</td>
</tr>
<tr>
<td>GPC11</td>
<td>Input/output</td>
<td>VD3</td>
</tr>
<tr>
<td>GPC10</td>
<td>Input/output</td>
<td>VD2</td>
</tr>
<tr>
<td>GPC9</td>
<td>Input/output</td>
<td>VD1</td>
</tr>
<tr>
<td>GPC8</td>
<td>Input/output</td>
<td>VD0</td>
</tr>
<tr>
<td>GPC7</td>
<td>Input/output</td>
<td>LCDVF2</td>
</tr>
<tr>
<td>GPC6</td>
<td>Input/output</td>
<td>LCDVF1</td>
</tr>
<tr>
<td>GPC5</td>
<td>Input/output</td>
<td>LCDVF0</td>
</tr>
<tr>
<td>GPC4</td>
<td>Input/output</td>
<td>VM</td>
</tr>
<tr>
<td>GPC3</td>
<td>Input/output</td>
<td>VFRAME</td>
</tr>
<tr>
<td>GPC2</td>
<td>Input/output</td>
<td>VLINE</td>
</tr>
<tr>
<td>GPC1</td>
<td>Input/output</td>
<td>VCLK</td>
</tr>
<tr>
<td>GPC0</td>
<td>Input/output</td>
<td>LEND</td>
</tr>
</tbody>
</table>
### Table 9-1. S3C2410X Port Configuration (Sheet 3 of 5)

<table>
<thead>
<tr>
<th>Port D</th>
<th>Selectable Pin Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPD15</td>
<td>Input/output</td>
<td>VD23</td>
</tr>
<tr>
<td>GPD14</td>
<td>Input/output</td>
<td>VD22</td>
</tr>
<tr>
<td>GPD13</td>
<td>Input/output</td>
<td>VD21</td>
</tr>
<tr>
<td>GPD12</td>
<td>Input/output</td>
<td>VD20</td>
</tr>
<tr>
<td>GPD11</td>
<td>Input/output</td>
<td>VD19</td>
</tr>
<tr>
<td>GPD10</td>
<td>Input/output</td>
<td>VD18</td>
</tr>
<tr>
<td>GPD9</td>
<td>Input/output</td>
<td>VD17</td>
</tr>
<tr>
<td>GPD8</td>
<td>Input/output</td>
<td>VD16</td>
</tr>
<tr>
<td>GPD7</td>
<td>Input/output</td>
<td>VD15</td>
</tr>
<tr>
<td>GPD6</td>
<td>Input/output</td>
<td>VD14</td>
</tr>
<tr>
<td>GPD5</td>
<td>Input/output</td>
<td>VD13</td>
</tr>
<tr>
<td>GPD4</td>
<td>Input/output</td>
<td>VD12</td>
</tr>
<tr>
<td>GPD3</td>
<td>Input/output</td>
<td>VD11</td>
</tr>
<tr>
<td>GPD2</td>
<td>Input/output</td>
<td>VD10</td>
</tr>
<tr>
<td>GPD1</td>
<td>Input/output</td>
<td>VD9</td>
</tr>
<tr>
<td>GPD0</td>
<td>Input/output</td>
<td>VD8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port E</th>
<th>Selectable Pin Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPE15</td>
<td>Input/output</td>
<td>IICSDA</td>
</tr>
<tr>
<td>GPE14</td>
<td>Input/output</td>
<td>IICSCL</td>
</tr>
<tr>
<td>GPE13</td>
<td>Input/output</td>
<td>SPICLK0</td>
</tr>
<tr>
<td>GPE12</td>
<td>Input/output</td>
<td>SPIOMOSI0</td>
</tr>
<tr>
<td>GPE11</td>
<td>Input/output</td>
<td>SPIOMISO0</td>
</tr>
<tr>
<td>GPE10</td>
<td>Input/output</td>
<td>SDDAT3</td>
</tr>
<tr>
<td>GPE9</td>
<td>Input/output</td>
<td>SDDAT2</td>
</tr>
<tr>
<td>GPE8</td>
<td>Input/output</td>
<td>SDDAT1</td>
</tr>
<tr>
<td>GPE7</td>
<td>Input/output</td>
<td>SDDAT0</td>
</tr>
<tr>
<td>GPE6</td>
<td>Input/output</td>
<td>SDCMD</td>
</tr>
<tr>
<td>GPE5</td>
<td>Input/output</td>
<td>SDCLK</td>
</tr>
<tr>
<td>GPE4</td>
<td>Input/output</td>
<td>I2SSDO</td>
</tr>
<tr>
<td>GPE3</td>
<td>Input/output</td>
<td>I2SSDI</td>
</tr>
<tr>
<td>GPE2</td>
<td>Input/output</td>
<td>nSS0</td>
</tr>
<tr>
<td>GPE1</td>
<td>Input/output</td>
<td>I2SSCLK</td>
</tr>
<tr>
<td>GPE0</td>
<td>Input/output</td>
<td>I2SLRCK</td>
</tr>
</tbody>
</table>
Table 9-1. S3C2410X Port Configuration (Sheet 4 of 5)

<table>
<thead>
<tr>
<th>Port F</th>
<th>Selectable Pin Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPF7 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT7 – – –</td>
</tr>
<tr>
<td></td>
<td>GPF6 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT6 – – –</td>
</tr>
<tr>
<td></td>
<td>GPF5 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT5 – – –</td>
</tr>
<tr>
<td></td>
<td>GPF4 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT4 – – –</td>
</tr>
<tr>
<td></td>
<td>GPF3 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT3 – – –</td>
</tr>
<tr>
<td></td>
<td>GPF2 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT2</td>
</tr>
<tr>
<td></td>
<td>GPF1 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT1</td>
</tr>
<tr>
<td></td>
<td>GPF0 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port G</th>
<th>Selectable Pin Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPG15 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT23 nYPON –</td>
</tr>
<tr>
<td></td>
<td>GPG14 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT22 YMON –</td>
</tr>
<tr>
<td></td>
<td>GPG13 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT21 nXPON –</td>
</tr>
<tr>
<td></td>
<td>GPG12 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT20 XMON –</td>
</tr>
<tr>
<td></td>
<td>GPG11 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT19 TCLK1 –</td>
</tr>
<tr>
<td></td>
<td>GPG10 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT18 – – –</td>
</tr>
<tr>
<td></td>
<td>GPG9 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT17 – – –</td>
</tr>
<tr>
<td></td>
<td>GPG8 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT16 – – –</td>
</tr>
<tr>
<td></td>
<td>GPG7 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT15 SPICLK1 –</td>
</tr>
<tr>
<td></td>
<td>GPG6 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT14 SPIMOSI1 –</td>
</tr>
<tr>
<td></td>
<td>GPG5 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT13 SPIMISO1 –</td>
</tr>
<tr>
<td></td>
<td>GPG4 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT12 LCD_PWREN –</td>
</tr>
<tr>
<td></td>
<td>GPG3 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT11 nSS1 –</td>
</tr>
<tr>
<td></td>
<td>GPG2 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT10 nSS0 –</td>
</tr>
<tr>
<td></td>
<td>GPG1 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT9 – – –</td>
</tr>
<tr>
<td></td>
<td>GPG0 Input/output</td>
</tr>
<tr>
<td></td>
<td>EINT8 – – –</td>
</tr>
</tbody>
</table>
Table 9-1. S3C2410X Port Configuration (Sheet 5 of 5)

<table>
<thead>
<tr>
<th>Port H</th>
<th>Selectable Pin Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPH10</td>
<td>Input/output, CLKOUT1</td>
</tr>
<tr>
<td>GPH9</td>
<td>Input/output, CLKOUT0</td>
</tr>
<tr>
<td>GPH8</td>
<td>Input/output, UCLK</td>
</tr>
<tr>
<td>GPH7</td>
<td>Input/output, RXD2, nCTS1</td>
</tr>
<tr>
<td>GPH6</td>
<td>Input/output, TXD2, nRTS1</td>
</tr>
<tr>
<td>GPH5</td>
<td>Input/output, RXD1, nRTS0</td>
</tr>
<tr>
<td>GPH4</td>
<td>Input/output, TXD1</td>
</tr>
<tr>
<td>GPH3</td>
<td>Input/output, RXD0</td>
</tr>
<tr>
<td>GPH2</td>
<td>Input/output, TXD0</td>
</tr>
<tr>
<td>GPH1</td>
<td>Input/output, nRTS0</td>
</tr>
<tr>
<td>GPH0</td>
<td>Input/output, nCTS0</td>
</tr>
</tbody>
</table>
PORT CONTROL DESCRIPTIONS

PORT CONFIGURATION REGISTER (GPACON-GPHCON)

In the S3C2410X, most pins are multiplexed. So, it is require to determine which function is selected for each pin. port control register (PnCON) determines the function of each pin.

If GPF0 – GPF7 and GPG0 – GPG7 are used for wakeup signals in Power-OFF mode, these ports must be configured in Interrupt mode.

PORT DATA REGISTER (GPADAT-GPHDAT)

If ports are configured as output ports, data can be written to the corresponding bit of the PnDAT. If ports are configured as input ports, the data can be read from the corresponding bit of the PnDAT.

PORT PULL-UP REGISTER (GPBUP-GPHUP)

The port pull-up register controls the pull-up resister enable/disable of each port group. When the corresponding bit is 0, the pull-up resister of the pin is enabled. When 1, the pull-up resister is disabled.

If the port pull-up register is enabled, the pull-up resisters work without pin’s functional setting (input, output, DATAn, EINTn, etc).

MISCELLANEOUS CONTROL REGISTER

This register controls DATA port pull-up resister, hi-z state, USB pad, and CLKOUT selection.

EXTERNAL INTERRUPT CONTROL REGISTER (EXTINTN)

The 24 external interrupts are requested by various signaling methods. The EXTINTn register configures the signaling method among the low level trigger, high level trigger, falling edge trigger, rising edge trigger, and both edge trigger for the external interrupt request.

The 8 external interrupt pin has a digital filter (refer to EINTFLTn on page 9-26).

Only 16 EINT pins (EINT [15:0]) are used for wakeup sources.

POWER_OFF MODE AND I/O PORTS

All GPIO register values are preserved in Power_OFF mode. Refer to the Power_OFF mode in the chapter, Clock & Power Management.

The EINTMASK can’t prohibit the wake-up from Power_OFF mode, But, If ENTMASK is masking one of EINT[15:4], the wake-up can be done but the EINT4_7 bit and EINT8_23 bit of the SRCPND will not set to 1 just after the wake-up.
## I/O PORT CONTROL REGISTER

### PORT A CONTROL REGISTERS (GPACON/GPADAT)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPACON</td>
<td>0x56000000</td>
<td>R/W</td>
<td>Configure the pins of port A</td>
<td>0x7FFFFFFF</td>
</tr>
<tr>
<td>GPADAT</td>
<td>0x56000004</td>
<td>R/W</td>
<td>The data register for port A</td>
<td>Undefined</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x56000008</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5600000C</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

### GPACON Bit Description

<table>
<thead>
<tr>
<th>GPA22</th>
<th>[22] 0 = Output 1 = nFCE</th>
</tr>
</thead>
</table>
| GPA21 | [21] 0 = Output 1 = nRSTOUT  
(nRSTOUT = nRESET & nWDTRST & SW_RESET(MISCCR[16])) |
| GPA20 | [20] 0 = Output 1 = nFRE   |
| GPA19 | [19] 0 = Output 1 = nFWE   |
| GPA18 | [18] 0 = Output 1 = ALE    |
| GPA17 | [17] 0 = Output 1 = CLE    |
| GPA16 | [16] 0 = Output 1 = nGCS5  |
| GPA15 | [15] 0 = Output 1 = nGCS4  |
| GPA14 | [14] 0 = Output 1 = nGCS3  |
| GPA13 | [13] 0 = Output 1 = nGCS2  |
| GPA12 | [12] 0 = Output 1 = nGCS1  |
| GPA11 | [11] 0 = Output 1 = ADDR26 |
| GPA10 | [10] 0 = Output 1 = ADDR25 |
| GPA9  | [9] 0 = Output 1 = ADDR24  |
| GPA8  | [8] 0 = Output 1 = ADDR23  |
| GPA7  | [7] 0 = Output 1 = ADDR22  |
| GPA6  | [6] 0 = Output 1 = ADDR21  |
| GPA5  | [5] 0 = Output 1 = ADDR20  |
| GPA4  | [4] 0 = Output 1 = ADDR19  |
| GPA3  | [3] 0 = Output 1 = ADDR18  |
| GPA2  | [2] 0 = Output 1 = ADDR17  |
| GPA1  | [1] 0 = Output 1 = ADDR16  |
| GPA0  | [0] 0 = Output 1 = ADDR0   |

### GPADAT Bit Description

<table>
<thead>
<tr>
<th>GPA[22:0]</th>
<th>[22:0]</th>
</tr>
</thead>
</table>
| GPA[22:0] | 0 = Output 1 = ADDR0  
When the port is configured as output port, the pin state is the same as the that of the corresponding bit.  
When the port is configured as functional pin, undefined value will be read. |
### PORT B CONTROL REGISTERS (GPBCON, GPBDAT, and GPBUP)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPBCON</td>
<td>0x56000010</td>
<td>R/W</td>
<td>Configure the pins of port B</td>
<td>0x0</td>
</tr>
<tr>
<td>GPBDAT</td>
<td>0x56000014</td>
<td>R/W</td>
<td>The data register for port B</td>
<td>Undefined</td>
</tr>
<tr>
<td>GPBUP</td>
<td>0x56000018</td>
<td>R/W</td>
<td>Pull-up disable register for port B</td>
<td>0x0</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5600001C</td>
<td></td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

### GPBCON Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
</table>
| GPB10 [21:20] | 00 = Input  
10 = nXDREQ0  
11 = reserved | |
| GPB9 [19:18] | 00 = Input  
10 = nXDACK0  
11 = reserved | |
| GPB8 [17:16] | 00 = Input  
10 = nXDREQ1  
11 = Reserved | |
| GPB7 [15:14] | 00 = Input  
10 = nXDACK1  
11 = Reserved | |
| GPB6 [13:12] | 00 = Input  
10 = nXBREQ  
11 = reserved | |
| GPB5 [11:10] | 00 = Input  
10 = nXBACK  
11 = reserved | |
| GPB4 [9:8] | 00 = Input  
10 = TCLK0  
11 = reserved | |
| GPB3 [7:6] | 00 = Input  
10 = TOUT3  
11 = reserved | |
| GPB2 [5:4] | 00 = Input  
10 = TOUT2  
11 = reserved | |
| GPB1 [3:2] | 00 = Input  
10 = TOUT1  
11 = reserved | |
| GPB0 [1:0] | 00 = Input  
10 = TOUT0  
11 = reserved | |

### GPBDAT Bit Description

When the port is configured as input port, data from external sources can be read to the corresponding pin. When the port is configured as output port, data written in this register can be sent to the corresponding pin. When the port is configured as functional pin, undefined value will be read.

### GPBUP Bit Description

0: The pull-up function attached to the corresponding port pin is enabled.
1: The pull-up function is disabled.
## PORT C CONTROL REGISTERS (GPCCON, GPCDAT, and GPCUP)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPCCON</td>
<td>0x560000020</td>
<td>R/W</td>
<td>Configure the pins of port C</td>
<td>0x0</td>
</tr>
<tr>
<td>GPCDAT</td>
<td>0x560000024</td>
<td>R/W</td>
<td>The data register for port C</td>
<td>Undefined</td>
</tr>
<tr>
<td>GPCUP</td>
<td>0x560000028</td>
<td>R/W</td>
<td>Pull-up disable register for port C</td>
<td>0x0</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x56000002C</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

### GPCCON Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPC15</td>
<td>[31:30]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC14</td>
<td>[29:28]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC13</td>
<td>[27:26]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC12</td>
<td>[25:24]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC11</td>
<td>[23:22]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC10</td>
<td>[21:20]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC9</td>
<td>[19:18]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC8</td>
<td>[17:16]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC7</td>
<td>[15:14]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC6</td>
<td>[13:12]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC5</td>
<td>[11:10]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC4</td>
<td>[9:8]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC3</td>
<td>[7:6]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC2</td>
<td>[5:4]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC1</td>
<td>[3:2]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC0</td>
<td>[1:0]</td>
<td>00 = Input</td>
</tr>
</tbody>
</table>

### GPCDAT Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPC15</td>
<td>[31:30]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC14</td>
<td>[29:28]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC13</td>
<td>[27:26]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC12</td>
<td>[25:24]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC11</td>
<td>[23:22]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC10</td>
<td>[21:20]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC9</td>
<td>[19:18]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC8</td>
<td>[17:16]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC7</td>
<td>[15:14]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC6</td>
<td>[13:12]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC5</td>
<td>[11:10]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC4</td>
<td>[9:8]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC3</td>
<td>[7:6]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC2</td>
<td>[5:4]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC1</td>
<td>[3:2]</td>
<td>00 = Input</td>
</tr>
<tr>
<td>GPC0</td>
<td>[1:0]</td>
<td>00 = Input</td>
</tr>
</tbody>
</table>
When the port is configured as input port, data from external sources can be read to the corresponding pin. When the port is configured as output port, data written in this register can be sent to the corresponding pin. When the port is configured as functional pin, undefined value will be read.

<table>
<thead>
<tr>
<th>GPC[15:0]</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[15:0]</td>
<td>0: The pull-up function attached to the corresponding port pin is enabled. 1: The pull-up function is disabled.</td>
</tr>
</tbody>
</table>
### PORT D CONTROL REGISTERS (GPDCON, GPDDAT, and GPDUP)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPDCON</td>
<td>0x56000030</td>
<td>R/W</td>
<td>Configure the pins of port D</td>
<td>0x0</td>
</tr>
<tr>
<td>GPDDAT</td>
<td>0x56000034</td>
<td>R/W</td>
<td>The data register for port D</td>
<td>Undefined</td>
</tr>
<tr>
<td>GPDUP</td>
<td>0x56000038</td>
<td>R/W</td>
<td>Pull-up disable register for port D</td>
<td>0xF000</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5600003C</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

#### GPDCON Bit Description

- **GPD15** 
  - [31:30]: 00 = Input, 01 = Output, 10 = VD23, 11 = nSS0
- **GPD14** 
  - [29:28]: 00 = Input, 01 = Output, 10 = VD22, 11 = nSS1
- **GPD13** 
  - [27:26]: 00 = Input, 01 = Output, 10 = VD21, 11 = Reserved
- **GPD12** 
  - [25:24]: 00 = Input, 01 = Output, 10 = VD20, 11 = Reserved
- **GPD11** 
  - [23:22]: 00 = Input, 01 = Output, 10 = VD19, 11 = Reserved
- **GPD10** 
  - [21:20]: 00 = Input, 01 = Output, 10 = VD18, 11 = Reserved
- **GPD9**   
  - [19:18]: 00 = Input, 01 = Output, 10 = VD17, 11 = Reserved
- **GPD8**   
  - [17:16]: 00 = Input, 01 = Output, 10 = VD16, 11 = Reserved
- **GPD7**   
  - [15:14]: 00 = Input, 01 = Output, 10 = VD15, 11 = Reserved
- **GPD6**   
  - [13:12]: 00 = Input, 01 = Output, 10 = VD14, 11 = Reserved
- **GPD5**   
  - [11:10]: 00 = Input, 01 = Output, 10 = VD13, 11 = Reserved
- **GPD4**   
  - [9:8]: 00 = Input, 01 = Output, 10 = VD12, 11 = Reserved
- **GPD3**   
  - [7:6]: 00 = Input, 01 = Output, 10 = VD11, 11 = Reserved
- **GPD2**   
  - [5:4]: 00 = Input, 01 = Output, 10 = VD10, 11 = Reserved
- **GPD1**   
  - [3:2]: 00 = Input, 01 = Output, 10 = VD9, 11 = Reserved
- **GPD0**   
  - [1:0]: 00 = Input, 01 = Output, 10 = VD8, 11 = Reserved

#### GPDDAT Bit Description

- **GPD15** [31:30]: 00 = Input, 01 = Output, 10 = VD23, 11 = nSS0
- **GPD14** [29:28]: 00 = Input, 01 = Output, 10 = VD22, 11 = nSS1
- **GPD13** [27:26]: 00 = Input, 01 = Output, 10 = VD21, 11 = Reserved
- **GPD12** [25:24]: 00 = Input, 01 = Output, 10 = VD20, 11 = Reserved
- **GPD11** [23:22]: 00 = Input, 01 = Output, 10 = VD19, 11 = Reserved
- **GPD10** [21:20]: 00 = Input, 01 = Output, 10 = VD18, 11 = Reserved
- **GPD9**  [19:18]: 00 = Input, 01 = Output, 10 = VD17, 11 = Reserved
- **GPD8**  [17:16]: 00 = Input, 01 = Output, 10 = VD16, 11 = Reserved
- **GPD7**  [15:14]: 00 = Input, 01 = Output, 10 = VD15, 11 = Reserved
- **GPD6**  [13:12]: 00 = Input, 01 = Output, 10 = VD14, 11 = Reserved
- **GPD5**  [11:10]: 00 = Input, 01 = Output, 10 = VD13, 11 = Reserved
- **GPD4**  [9:8]: 00 = Input, 01 = Output, 10 = VD12, 11 = Reserved
- **GPD3**  [7:6]: 00 = Input, 01 = Output, 10 = VD11, 11 = Reserved
- **GPD2**  [5:4]: 00 = Input, 01 = Output, 10 = VD10, 11 = Reserved
- **GPD1**  [3:2]: 00 = Input, 01 = Output, 10 = VD9, 11 = Reserved
- **GPD0**  [1:0]: 00 = Input, 01 = Output, 10 = VD8, 11 = Reserved
When the port is configured as input port, data from external sources can be read to the corresponding pin. When the port is configured as output port, data written in this register can be sent to the corresponding pin. When the port is configured as functional pin, undefined value will be read.

<table>
<thead>
<tr>
<th>GPDUP</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPD[15:0]</td>
<td>[15:0]</td>
<td>0: The pull-up function attached to the corresponding port pin is enabled. 1: The pull-up function is disabled. (GPD[15:12] are ‘pull-up disabled’ state at the initial condition.)</td>
</tr>
</tbody>
</table>
## PORT E CONTROL REGISTERS (GPECON, GPEDAT, and GPEUP)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPECON</td>
<td>0x56000040</td>
<td>R/W</td>
<td>Configure the pins of port E</td>
<td>0x0</td>
</tr>
<tr>
<td>GPEDAT</td>
<td>0x56000044</td>
<td>R/W</td>
<td>The data register for port E</td>
<td>Undefined</td>
</tr>
<tr>
<td>GPEUP</td>
<td>0x56000048</td>
<td>R/W</td>
<td>pull-up disable register for port E</td>
<td>0x0</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5600004C</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPECON</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[31:30]</td>
<td>00 = Input 10 = IICSDA 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[29:28]</td>
<td>00 = Input 10 = IICSCL 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[27:26]</td>
<td>00 = Input 10 = SPICLK0 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[25:24]</td>
<td>00 = Input 10 = SPIMOSI0 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[23:22]</td>
<td>00 = Input 10 = SPIMISO0 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[21:20]</td>
<td>00 = Input 10 = SDDAT3 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[19:18]</td>
<td>00 = Input 10 = SDDAT2 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[17:16]</td>
<td>00 = Input 10 = SDDAT1 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[15:14]</td>
<td>00 = Input 10 = SDDAT0 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[13:12]</td>
<td>00 = Input 10 = SDCMD 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[11:10]</td>
<td>00 = Input 10 = SDCLK 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[9:8]</td>
<td>00 = Input 10 = I2SSDO 11 = I2SSDI</td>
</tr>
<tr>
<td></td>
<td>[7:6]</td>
<td>00 = Input 10 = I2SSDI 11 = nSS0</td>
</tr>
<tr>
<td></td>
<td>[5:4]</td>
<td>00 = Input 10 = CDCLK 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[3:2]</td>
<td>00 = Input 10 = I2SSCLK 11 = Reserved</td>
</tr>
<tr>
<td></td>
<td>[1:0]</td>
<td>00 = Input 10 = I2SLRCK 11 = Reserved</td>
</tr>
</tbody>
</table>
### GPEDAT Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPE[15:0] [15:0]</td>
<td>When the port is configured as input port, data from external sources can be read to the corresponding pin. When the port is configured as output port, data written in this register can be sent to the corresponding pin. When the port is configured as a functional pin, undefined value will be read.</td>
</tr>
</tbody>
</table>

### GPEUP Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPE[15:0] [15:0]</td>
<td>0: The pull-up function attached to the corresponding port pin is enabled. 1: The pull-up function is disabled.</td>
</tr>
</tbody>
</table>
PORT F CONTROL REGISTERS (GPFCON, GPFDAT, and GPFPU)

If GPF0 - GPF7 will be used for wakeup signals in Power-OFF mode, the ports will be set in Interrupt mode.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPFCON</td>
<td>0x56000050</td>
<td>R/W</td>
<td>Configure the pins of port F</td>
<td>0x0</td>
</tr>
<tr>
<td>GPFDAT</td>
<td>0x56000054</td>
<td>R/W</td>
<td>The data register for port F</td>
<td>Undefined</td>
</tr>
<tr>
<td>GPFUP</td>
<td>0x56000058</td>
<td>R/W</td>
<td>Pull-up disable register for port F</td>
<td>0x0</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5600005C</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPFCON</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPF7</td>
<td>[15:14]</td>
<td>00 = Input 01 = Output 10 = EINT7 11 = Reserved</td>
</tr>
<tr>
<td>GPF6</td>
<td>[13:12]</td>
<td>00 = Input 01 = Output 10 = EINT6 11 = Reserved</td>
</tr>
<tr>
<td>GPF5</td>
<td>[11:10]</td>
<td>00 = Input 01 = Output 10 = EINT5 11 = Reserved</td>
</tr>
<tr>
<td>GPF4</td>
<td>[9:8]</td>
<td>00 = Input 01 = Output 10 = EINT4 11 = Reserved</td>
</tr>
<tr>
<td>GPF3</td>
<td>[7:6]</td>
<td>00 = Input 01 = Output 10 = EINT3 11 = Reserved</td>
</tr>
<tr>
<td>GPF2</td>
<td>[5:4]</td>
<td>00 = Input 01 = Output 10 = EINT2 11 = Reserved</td>
</tr>
<tr>
<td>GPF1</td>
<td>[3:2]</td>
<td>00 = Input 01 = Output 10 = EINT1 11 = Reserved</td>
</tr>
<tr>
<td>GPF0</td>
<td>[1:0]</td>
<td>00 = Input 01 = Output 10 = EINT0 11 = Reserved</td>
</tr>
</tbody>
</table>

NOTES:

1. 

<table>
<thead>
<tr>
<th>GPFDAT</th>
<th>Bit</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPF[7:0]</td>
<td>[7:0]</td>
<td>When the port is configured as input port, data from external sources can be read to the corresponding pin. When the port is configured as output port, data written in this register can be sent to the corresponding pin. When the port is configured as functional pin, undefined value will be read.</td>
<td></td>
</tr>
</tbody>
</table>

2. 

<table>
<thead>
<tr>
<th>GPFUP</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPF[7:0]</td>
<td>[7:0]</td>
<td>0: The pull-up function attached to the corresponding port pin is enabled. 1: The pull-up function is disabled.</td>
</tr>
</tbody>
</table>
PORT G CONTROL REGISTERS (GPGCON, GPGDAT, AND GPUP)

If GPG[7:0] will be used for wakeup signals at Power-OFF mode, the ports will be set in Interrupt mode.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPGCON</td>
<td>0x56000060</td>
<td>R/W</td>
<td>Configure the pins of port G</td>
<td>0x0</td>
</tr>
<tr>
<td>GPGDAT</td>
<td>0x56000064</td>
<td>R/W</td>
<td>The data register for port G</td>
<td>Undefined</td>
</tr>
<tr>
<td>GPUP</td>
<td>0x56000068</td>
<td>R/W</td>
<td>Pull-up disable register for port G</td>
<td>0xF800</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5600006C</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPGCON</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPG15</td>
<td>[31:30]</td>
<td>00 = Input 10 = EINT23 11 = nYPON</td>
</tr>
<tr>
<td>GPG14</td>
<td>[29:28]</td>
<td>00 = Input 10 = EINT22 11 = YMON</td>
</tr>
<tr>
<td>GPG13</td>
<td>[27:26]</td>
<td>00 = Input 10 = EINT21 11 = nXPON</td>
</tr>
<tr>
<td>GPG12</td>
<td>[25:24]</td>
<td>00 = Input 10 = EINT20 11 = XMON</td>
</tr>
<tr>
<td>GPG11</td>
<td>[23:22]</td>
<td>00 = Input 10 = EINT19 11 = TCLK1</td>
</tr>
<tr>
<td>GPG10</td>
<td>[21:20]</td>
<td>00 = Input 10 = EINT18 11 = Reserved</td>
</tr>
<tr>
<td>GPG9</td>
<td>[19:18]</td>
<td>00 = Input 10 = EINT17 11 = Reserved</td>
</tr>
<tr>
<td>GPG8</td>
<td>[17:16]</td>
<td>00 = Input 10 = EINT16 11 = Reserved</td>
</tr>
<tr>
<td>GPG7</td>
<td>[15:14]</td>
<td>00 = Input 10 = EINT15 11 = SPICLK1</td>
</tr>
<tr>
<td>GPG6</td>
<td>[13:12]</td>
<td>00 = Input 10 = EINT14 11 = SPIMOSI1</td>
</tr>
<tr>
<td>GPG5</td>
<td>[11:10]</td>
<td>00 = Input 10 = EINT13 11 = SPIMISO1</td>
</tr>
<tr>
<td>GPG4</td>
<td>[9:8]</td>
<td>00 = Input 10 = EINT12 11 = LCD_PWREN</td>
</tr>
<tr>
<td>GPG3</td>
<td>[7:6]</td>
<td>00 = Input 10 = EINT11 11 = nSS1</td>
</tr>
<tr>
<td>GPG2</td>
<td>[5:4]</td>
<td>00 = Input 10 = EINT10 11 = nSS0</td>
</tr>
<tr>
<td>GPG1</td>
<td>[3:2]</td>
<td>00 = Input 10 = EINT9 11 = Reserved</td>
</tr>
<tr>
<td>GPG0</td>
<td>[1:0]</td>
<td>00 = Input 10 = EINT8 11 = Reserved</td>
</tr>
</tbody>
</table>
### GPGDAT Bit Description

<table>
<thead>
<tr>
<th>GPG[15:0]</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[15:0]</td>
<td>When the port is configured as input port, data from external sources can be read to the corresponding pin. When the port is configured as output port, data written in this register can be sent to the corresponding pin. When the port is configured as functional pin, undefined value will be read.</td>
</tr>
</tbody>
</table>

### GPGUP Bit Description

<table>
<thead>
<tr>
<th>GPG[15:0]</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[15:0]</td>
<td>0: The pull-up function attached to the corresponding port pin is enabled. 1: The pull-up function is disabled. (GPG[15:11] are ‘pull-up disabled’ state at the initial condition.)</td>
</tr>
</tbody>
</table>
## PORT H CONTROL REGISTERS (GPHCON, GPHDAT, AND GPHUP)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPHCON</td>
<td>0x56000070</td>
<td>R/W</td>
<td>Configure the pins of port H</td>
<td>0x0</td>
</tr>
<tr>
<td>GPHDAT</td>
<td>0x56000074</td>
<td>R/W</td>
<td>The data register for port H</td>
<td>Undefined</td>
</tr>
<tr>
<td>GPHUP</td>
<td>0x56000078</td>
<td>R/W</td>
<td>Pull-up disable register for port H</td>
<td>0x0</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5600007C</td>
<td>–</td>
<td>Reserved</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

### GPHCON Bit Description

| GPH10  | [21:20] | 00 = Input 01 = Output 10 = CLKOUT1 11 = Reserved |
| GPH9   | [19:18] | 00 = Input 01 = Output 10 = CLKOUT0 11 = Reserved |
| GPH8   | [17:16] | 00 = Input 01 = Output 10 = UCLK 11 = Reserved    |
| GPH7   | [15:14] | 00 = Input 01 = Output 10 = RXD2 11 = nCTS1       |
| GPH6   | [13:12] | 00 = Input 01 = Output 10 = TXD2 11 = nRTS1       |
| GPH5   | [11:10] | 00 = Input 01 = Output 10 = RXD1 11 = Reserved    |
| GPH4   | [9:8]    | 00 = Input 01 = Output 10 = TXD1 11 = Reserved    |
| GPH3   | [7:6]    | 00 = Input 01 = Output 10 = RXD0 11 = reserved    |
| GPH2   | [5:4]    | 00 = Input 01 = Output 10 = TXD0 11 = Reserved    |
| GPH1   | [3:2]    | 00 = Input 01 = Output 10 = nRTS0 11 = Reserved   |
| GPH0   | [1:0]    | 00 = Input 01 = Output 10 = nCTS0 11 = Reserved   |

### GPHDAT Bit Description

- **GPH[10:0]**: [10:0] 0: The pull-up function attached to to the corresponding port pin is enabled. 1: The pull-up function is disabled.
### MISCELLANEOUS CONTROL REGISTER (MISCCR)

Pads related USB are controlled by this register for USB host, or for USB device.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISCCR</td>
<td>0x56000080</td>
<td>R/W</td>
<td>Miscellaneous control register</td>
<td>0x10330</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MISCCR</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[21:20]</td>
<td>Reserved to 00b</td>
</tr>
<tr>
<td>nEN_SCKE</td>
<td>[19]</td>
<td>0: SCKE = Normal    1: SCKE = L level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used to protect SDRAM during the Power_OFF mode.</td>
</tr>
<tr>
<td>nEN_SCLK1</td>
<td>[18]</td>
<td>0: SCLK1= SCLK    1: SCLK1= L level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used to protect SDRAM during the Power_OFF mode.</td>
</tr>
<tr>
<td>nEN_SCLK0</td>
<td>[17]</td>
<td>0: SCLK0= SCLK    1: SCLK0= L level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used to protect SDRAM during the Power_OFF mode.</td>
</tr>
<tr>
<td>nRSTCON</td>
<td>[16]</td>
<td>nRSTOUT software control(SW_RESET)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: nRSTOUT = 0, 1: nRSTOUT = 1.</td>
</tr>
<tr>
<td>Reserved</td>
<td>[15:14]</td>
<td>Reserved to 00b</td>
</tr>
<tr>
<td>USBSUSPND1</td>
<td>[13]</td>
<td>[13] USB Port 1 mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Normal    1= Suspend</td>
</tr>
<tr>
<td>USBSUSPND0</td>
<td>[12]</td>
<td>[12] USB Port 0 mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Normal    1= Suspend</td>
</tr>
<tr>
<td>Reserved</td>
<td>[11]</td>
<td>Reserved to 0b</td>
</tr>
<tr>
<td>CLKSEL1</td>
<td>[10:8]</td>
<td>CLKOUT1 output signal source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000 = MPLL CLK    001 = UPLL CLK    010 = FCLK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>011 = HCLK    100 = PCLK    101 = DCLK1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11x = Reserved</td>
</tr>
<tr>
<td>Reserved</td>
<td>[7]</td>
<td>0</td>
</tr>
<tr>
<td>CLKSEL0</td>
<td>[6:4]</td>
<td>CLKOUT0 output signal source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000 = MPLL CLK    001 = UPLL CLK    010 = FCLK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>011 = HCLK    100 = PCLK    101 = DCLK0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11x = Reserved</td>
</tr>
<tr>
<td>USBPAD</td>
<td>[3]</td>
<td>0 = Use pads related USB for USB device</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Use pads related USB for USB host</td>
</tr>
<tr>
<td>MEM_HZ_CON</td>
<td>[2]</td>
<td>This bit is recommended to be 0. nGCS[7:0], nWE, nOE, nBE[3:0], nSRAS, nSCAS, ADDR[26:0] are affected during CLKCON[0] = 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Hi-Z    1 = Previous state</td>
</tr>
<tr>
<td>SPUCR_L</td>
<td>[1]</td>
<td>DATA[15:0] port pull-up register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Enabled    1 = Disabled</td>
</tr>
<tr>
<td>SPUCR_H</td>
<td>[0]</td>
<td>DATA[31:16] port pull-up register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Enabled    1 = Disabled</td>
</tr>
</tbody>
</table>

**NOTE:** CLKOUT is prepared only for monitoring an internal clock situation (On/Off status or frequency).
DCLK CONTROL REGISTERS (DCLKCON)

This register defines DCLKn signals, which work as clocks for external sources. See the following figure for how to make the DCLKn signals. The DCLKCON can actually operate only when CLKOUT[1:0] is set to send the DCLKn signals.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCLKCON</td>
<td>0x56000084</td>
<td>R/W</td>
<td>DCLK0/1 control register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

### DCLKCON Bit Description

- **DCLK1CMP [27:24]** DCLK1 Compare value clock toggle value. ( < DCLK1DIV )
  - If the DCLK1DIV is n, Low level duration is ( n + 1 ).
  - High level duration is ( (DCLK1DIV + 1) – ( n +1 ) ).

- **DCLK1DIV [23:20]** DCLK1 Divide value
  - DCLK1 frequency = source clock / ( DCLK1DIV + 1 )

- **Reserved [19:18]** 00b

- **DCLK1SelCK [17]** Select DCLK1 source clock
  - 0 = PCLK
  - 1 = UCLK ( USB )

- **DCLK1EN [16]** DCLK1 Enable
  - 0 = Disable
  - 1 = Enable

- **Reserved [15:12]** 0000b

- **DCLK0CMP [11:8]** DCLK0 Compare value clock toggle value. ( < DCLK0DIV )
  - If the DCLK0DIV is n, Low level duration is ( n + 1 ).
  - High level duration is ( (DCLK0DIV + 1) – ( n +1 ) ).

- **DCLK0DIV [7:4]** DCLK0 Divide value.
  - DCLK0 frequency = source clock / ( DCLK0DIV + 1 )

- **Reserved [3:2]** 00b

- **DCLK0SelCK [1]** Select DCLK0 source clock
  - 0 = PCLK
  - 1 = UCLK ( USB )

- **DCLK0EN [0]** DCLK0 Enable
  - 0 = Disable
  - 1 = Enable
EXTERNAL INTERRUPT CONTROL REGISTER (EXTINTn)

The 24 external interrupts can be requested by various signaling methods. The EXTINTn configures the signaling method between the level trigger and edge trigger for the external interrupt request, and also configures the signal polarity.

To recognize the level interrupt, the valid logic level on EXTINTn pin must be retained at least for 40ns because of the noise filter (EINT[15:0]).

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTINT0</td>
<td>0x56000088</td>
<td>R/W</td>
<td>External interrupt control register 0</td>
<td>0x0</td>
</tr>
<tr>
<td>EXTINT1</td>
<td>0x5600008C</td>
<td>R/W</td>
<td>External interrupt control register 1</td>
<td>0x0</td>
</tr>
<tr>
<td>EXTINT2</td>
<td>0x56000090</td>
<td>R/W</td>
<td>External interrupt control register 2</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXTINT0 Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINT7 [30:28]</td>
<td>Set the signaling method of the EINT7. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EINT6 [26:24]</td>
<td>Set the signaling method of the EINT6. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EINT5 [22:20]</td>
<td>Set the signaling method of the EINT5. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EINT4 [18:16]</td>
<td>Set the signaling method of the EINT4. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EINT3 [14:12]</td>
<td>Set the signaling method of the EINT3. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EINT2 [10:8]</td>
<td>Set the signaling method of the EINT2. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EINT1 [6:4]</td>
<td>Set the signaling method of the EINT1. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EINT0 [2:0]</td>
<td>Set the signaling method of the EINT0. 000 = Low level 001 = High level 10x = Rising edge triggered 01x = Falling edge triggered 11x = Both edge triggered</td>
</tr>
<tr>
<td>EXTINT1</td>
<td>Bit</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>[31]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[27]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[23]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[19]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[15]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[7]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3]</td>
</tr>
<tr>
<td>EINT8</td>
<td>[2:0]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTINT2</td>
<td>Bit</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td>FLTEN23</td>
<td>[31]</td>
</tr>
<tr>
<td>EINT23</td>
<td>[30:28]</td>
</tr>
<tr>
<td>FLTEN22</td>
<td>[27]</td>
</tr>
<tr>
<td>EINT22</td>
<td>[26:24]</td>
</tr>
<tr>
<td>FLTEN21</td>
<td>[23]</td>
</tr>
<tr>
<td>EINT21</td>
<td>[22:20]</td>
</tr>
<tr>
<td>FLTEN20</td>
<td>[19]</td>
</tr>
<tr>
<td>EINT20</td>
<td>[18:16]</td>
</tr>
<tr>
<td>FLTEN19</td>
<td>[15]</td>
</tr>
<tr>
<td>EINT19</td>
<td>[14:12]</td>
</tr>
<tr>
<td>FLTEN18</td>
<td>[11]</td>
</tr>
<tr>
<td>EINT18</td>
<td>[10:8]</td>
</tr>
<tr>
<td>FLTEN17</td>
<td>[7]</td>
</tr>
<tr>
<td>EINT17</td>
<td>[6:4]</td>
</tr>
<tr>
<td>FLTEN16</td>
<td>[3]</td>
</tr>
<tr>
<td>EINT16</td>
<td>[2:0]</td>
</tr>
</tbody>
</table>
EXTERNAL INTERRUPT FILTER REGISTER (EINTFLTn)

The EINTFLTn controls the length of filter for 8 external interrupts (EINT[23:16]).

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINTFLT0</td>
<td>0x56000094</td>
<td>R/W</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>EINTFLT1</td>
<td>0x56000098</td>
<td>R/W</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>EINTFLT2</td>
<td>0x5600009C</td>
<td>R/W</td>
<td>External interrupt control register 2</td>
<td>0x0</td>
</tr>
<tr>
<td>EINTFLT3</td>
<td>0x4C6000A0</td>
<td>R/W</td>
<td>External interrupt control register 3</td>
<td>0x0</td>
</tr>
</tbody>
</table>

**EINTFLT2 Bit Description**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLTCLK19</td>
<td>Filter clock of EINT19</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT19</td>
<td>Filter width of EINT19</td>
<td>0x0</td>
</tr>
<tr>
<td>FLTCLK18</td>
<td>Filter clock of EINT18</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT18</td>
<td>Filter width of EINT18</td>
<td>0x0</td>
</tr>
<tr>
<td>FLTCLK17</td>
<td>Filter clock of EINT17</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT17</td>
<td>Filter width of EINT17</td>
<td>0x0</td>
</tr>
<tr>
<td>FLTCLK16</td>
<td>Filter clock of EINT16</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT16</td>
<td>Filter width of EINT16</td>
<td>0x0</td>
</tr>
</tbody>
</table>

**EINTFLT3 Bit Description**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLTCLK23</td>
<td>Filter clock of EINT23</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT23</td>
<td>Filter width of EINT23</td>
<td>0x0</td>
</tr>
<tr>
<td>FLTCLK22</td>
<td>Filter clock of EINT22</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT22</td>
<td>Filter width of EINT22</td>
<td>0x0</td>
</tr>
<tr>
<td>FLTCLK21</td>
<td>Filter clock of EINT21</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT21</td>
<td>Filter width of EINT21</td>
<td>0x0</td>
</tr>
<tr>
<td>FLTCLK20</td>
<td>Filter clock of EINT20</td>
<td>0 = PCLK, 1 = EXTCLK/OSC_CLK (Selected by OM pin)</td>
</tr>
<tr>
<td>EINTFLT20</td>
<td>Filter width of EINT20</td>
<td>0x0</td>
</tr>
</tbody>
</table>
EXTERNAL INTERRUPT MASK REGISTER (EINTMASK)

Interrupt mask register for 20 external interrupts (EINT[23:4]).

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINTMASK</td>
<td>0x560000A4</td>
<td>R/W</td>
<td>External interrupt mask register</td>
<td>0xFFFF0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EINT</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINT23</td>
<td>[23]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT22</td>
<td>[22]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT21</td>
<td>[21]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT20</td>
<td>[20]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT19</td>
<td>[19]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT18</td>
<td>[18]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT17</td>
<td>[17]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT16</td>
<td>[16]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT15</td>
<td>[15]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT14</td>
<td>[14]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT13</td>
<td>[13]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT12</td>
<td>[12]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT11</td>
<td>[11]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT10</td>
<td>[10]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT9</td>
<td>[9]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT8</td>
<td>[8]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT7</td>
<td>[7]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT6</td>
<td>[6]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT5</td>
<td>[5]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>EINT4</td>
<td>[4]</td>
<td>0 = Enable Interrupt, 1 = Masked</td>
</tr>
<tr>
<td>Reserved</td>
<td>[3:0]</td>
<td>0</td>
</tr>
</tbody>
</table>
EXTERNAL INTERRUPT PENDING REGISTER (EINTPENDn)

Interrupt pending register for 20 external interrupts (EINT[23:4]). You can clear a specific bit of the EINTPEND register by writing ‘1’ on the corresponding bit of this register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINTPEND</td>
<td>0x560000A8</td>
<td>R/W</td>
<td>External interrupt pending register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EINTPEND</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINT23</td>
<td>[23]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT22</td>
<td>[22]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT21</td>
<td>[21]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT20</td>
<td>[20]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT19</td>
<td>[19]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT18</td>
<td>[18]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT17</td>
<td>[17]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT16</td>
<td>[16]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT15</td>
<td>[15]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT14</td>
<td>[14]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT13</td>
<td>[13]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT12</td>
<td>[12]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT11</td>
<td>[11]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT10</td>
<td>[10]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT9</td>
<td>[9]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT8</td>
<td>[8]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT7</td>
<td>[7]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT6</td>
<td>[6]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT5</td>
<td>[5]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>EINT4</td>
<td>[4]</td>
<td>0 = Not requested 1 = Requested</td>
</tr>
<tr>
<td>Reserved</td>
<td>[3:0]</td>
<td>0</td>
</tr>
</tbody>
</table>
# GENERAL STATUS REGISTER (GSTATUS<sub>n</sub>)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSTATUS0</td>
<td>0x560000AC</td>
<td>R</td>
<td>External pin status</td>
<td>Undefined</td>
</tr>
<tr>
<td>GSTATUS1</td>
<td>0x560000B0</td>
<td>R</td>
<td>Chip ID</td>
<td>0x32410000</td>
</tr>
<tr>
<td>GSTATUS2</td>
<td>0x560000B4</td>
<td>R/W</td>
<td>Reset status</td>
<td>0x1</td>
</tr>
<tr>
<td>GSTATUS3</td>
<td>0x560000B8</td>
<td>R/W</td>
<td>Inform register</td>
<td>0x0</td>
</tr>
<tr>
<td>GSTATUS4</td>
<td>0x560000BC</td>
<td>R/W</td>
<td>Inform register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

### GSTATUS0 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nWAIT [3]</td>
<td>Status of nWAIT pin</td>
</tr>
<tr>
<td>NCON [2]</td>
<td>Status of NCON pin</td>
</tr>
<tr>
<td>RnB [1]</td>
<td>Status of R/nB pin</td>
</tr>
<tr>
<td>nBATT_FLT [0]</td>
<td>Status of nBATT_FLT pin</td>
</tr>
</tbody>
</table>

### GSTATUS1 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIP ID [31:0]</td>
<td>ID register = 0x32410000</td>
</tr>
</tbody>
</table>

### GSTATUS2 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWRST [0]</td>
<td>Power on reset, if this bit is set to “1”. The setting is cleared by writing “1” to this bit.</td>
</tr>
<tr>
<td>OFFRST [1]</td>
<td>Power_OFF reset. The reset after the wakeup from Power_OFF mode. The setting is cleared by writing “1” to this bit.</td>
</tr>
<tr>
<td>WDTRST [2]</td>
<td>Watchdog reset. The reset derived from Watchdog timer. The setting is cleared by writing “1” to this bit.</td>
</tr>
</tbody>
</table>

### GSTATUS3 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORM [31:0]</td>
<td>Inform register. This register is cleared by nRESET or watchdog timer. Otherwise, preserve data value.</td>
</tr>
</tbody>
</table>

### GSTATUS4 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORM [31:0]</td>
<td>Inform register. This register is cleared by nRESET or watchdog timer. Otherwise, preserve data value.</td>
</tr>
</tbody>
</table>
OVERVIEW

The S3C2410X has five 16-bit timers. Timer 0, 1, 2, and 3 have Pulse Width Modulation (PWM) function. Timer 4 has an internal timer only with no output pins. The timer 0 has a dead-zone generator, which is used with a large current device.

The timer 0 and 1 share an 8-bit prescaler, while the timer 2, 3 and 4 share other 8-bit prescaler. Each timer has a clock divider which 5 different divided signals (1/2, 1/4, 1/8, 1/16, and TCLK). Each timer block receives its own clock signals from the clock divider, which receives the clock from the corresponding 8-bit prescaler. The 8-bit prescaler is programmable and divides the PCLK according to the loading value, which is stored in TCFG0 and TCFG1 registers.

The timer count buffer register (TCNTBn) has an initial value which is loaded into the down-counter when the timer is enabled. The timer compare buffer register (TCMPBn) has an initial value which is loaded into the compare register to be compared with the down-counter value. This double buffering feature of TCNTBn and TCMPBn makes the timer generate a stable output when the frequency and duty ratio are changed.

Each timer has its own 16-bit down counter, which is driven by the timer clock. When the down counter reaches zero, the timer interrupt request is generated to inform the CPU that the timer operation has been completed. When the timer counter reaches zero, the value of corresponding TCNTBn is automatically loaded into the down counter to continue the next operation. However, if the timer stops, for example, by clearing the timer enable bit of TCONn during the timer running mode, the value of TCNTBn will not be reloaded into the counter.

The value of TCMPBn is used for pulse width modulation (PWM). The timer control logic changes the output level when the down-counter value matches the value of the compare register in the timer control logic. Therefore, the compare register determines the turn-on time (or turn-off time) of an PWM output.

FEATURE

— Five 16-bit timers
— Two 8-bit prescalers & Two 4-bit divider
— Programmable duty control of output waveform (PWM)
— Auto reload mode or one-shot pulse mode
— Dead-zone generator
Figure 10-1. 16-bit PWM Timer Block Diagram
PWM TIMER OPERATION

PRESCALER & DIVIDER

An 8-bit prescaler and a 4-bit divider make the following output frequencies:

<table>
<thead>
<tr>
<th>4-bit divider settings</th>
<th>Minimum resolution (prescaler = 0)</th>
<th>Maximum resolution (prescaler = 255)</th>
<th>Maximum interval (TCNTBn = 65535)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2 (PCLK = 50 MHz)</td>
<td>0.0400 us (25.0000 MHz)</td>
<td>10.2400 us (97.6562 kHz)</td>
<td>0.6710 sec</td>
</tr>
<tr>
<td>1/4 (PCLK = 50 MHz)</td>
<td>0.0800 us (12.5000 MHz)</td>
<td>20.4800 us (48.8281 kHz)</td>
<td>1.3421 sec</td>
</tr>
<tr>
<td>1/8 (PCLK = 50 MHz)</td>
<td>0.1600 us (  6.2500 MHz)</td>
<td>40.9601 us (24.4140 kHz)</td>
<td>2.6843 sec</td>
</tr>
<tr>
<td>1/16 (PCLK = 50 MHz)</td>
<td>0.3200 us (  3.1250 MHz)</td>
<td>81.9188 us (12.2070 kHz)</td>
<td>5.3686 sec</td>
</tr>
</tbody>
</table>

BASIC TIMER OPERATION

Figure 10-2. Timer Operations

A timer (except the timer ch-5) has TCNTBn, TCNTn, TCMPBn and TCMPn. (TCNTn and TCMPn are the names of the internal registers. The TCNTn register can be read from the TCNTo register) The TCNTBn and the TCMPBn are loaded into the TCNTn and the TCMPn when the timer reaches 0. When the TCNTn reaches 0, an interrupt request will occur if the interrupt is enabled.
### AUTO RELOAD & DOUBLE BUFFERING

S3C2410X PWM Timers have a double buffering function, enabling the reload value changed for the next timer operation without stopping the current timer operation. So, although the new timer value is set, a current timer operation is completed successfully.

The timer value can be written into Timer Count Buffer register (TCNTBn) and the current counter value of the timer can be read from Timer Count Observation register (TCNTOn). If the TCNTBn is read, the read value does not indicate the current state of the counter but the reload value for the next timer duration.

The auto reload operation copies the TCNTBn into TCNTn when the TCNTn reaches 0. The value, written into the TCNTBn, is loaded to the TCNTn only when the TCNTn reaches 0 and auto reload is enabled. If the TCNTn becomes 0 and the auto reload bit is 0, the TCNTn does not operate any further.

![Diagram of Double Buffering Function](image)

**Figure 10-3. Example of Double Buffering Function**
TIMER INITIALIZATION USING MANUAL UPDATE BIT AND INVERTER BIT

An auto reload operation of the timer occurs when the down counter reaches 0. So, a starting value of the TCNTn has to be defined by the user in advance. In this case, the starting value has to be loaded by the manual update bit. The following steps describe how to start a timer:

1) Write the initial value into TCNTBn and TCMPBn.

2) Set the manual update bit of the corresponding timer. It is recommended that you configure the inverter on/off bit. (whether use inverter or not).

3) Set start bit of the corresponding timer to start the timer (and clear the manual update bit).

If the timer is stopped by force, the TCNTn retains the counter value and is not reloaded from TCNTBn. If a new value has to be set, perform manual update.

NOTE

Whenever TOUT inverter on/off bit is changed, the TOUTn logic value will also be changed whether the timer runs. Therefore, it is desirable that the inverter on/off bit is configured with the manual update bit.
Figure 10-4 shows the result of the following procedure:

1. Enable the auto reload function. Set the TCNTBn to 160 (50+110) and the TCMPBn to 110. Set the manual update bit and configure the inverter bit (on/off). The manual update bit sets TCNTn and TCMPn to the values of TCNTBn and TCMPBn, respectively.

And then, set the TCNTBn and the TCMPBn to 80 (40+40) and 40, respectively, to determine the next reload value.

2. Set the start bit, provided that manual_update is 0 and the inverter is off and auto reload is on. The timer starts counting down after latency time within the timer resolution.

3. When the TCNTn has the same value as that of the TCMPn, the logic level of the TOUTn is changed from low to high.

4. When the TCNTn reaches 0, the interrupt request is generated and TCNTBn value is loaded into a temporary register. At the next timer tick, the TCNTn is reloaded with the temporary register value (TCNTBn).

5. In Interrupt Service Routine (ISR), the TCNTBn and the TCMPBn are set to 80 (20+60) and 60, respectively, for the next duration.

6. When the TCNTn has the same value as the TCMPn, the logic level of TOUTn is changed from low to high.

7. When the TCNTn reaches 0, the TCNTn is reloaded automatically with the TCNTBn, triggering an interrupt request.

8. In Interrupt Service Routine (ISR), auto reload and interrupt request are disabled to stop the timer.

9. When the value of the TCNTn is same as the TCMPn, the logic level of the TOUTn is changed from low to high.

10. Even when the TCNTn reaches 0, the TCNTn is not any more reloaded and the timer is stopped because auto reload has been disabled.

11. No more interrupt requests are generated.
PULSE WIDTH MODULATION (PWM)

PWM function can be implemented by using the TCMPBn. PWM frequency is determined by TCNTBn. Figure 10-5 shows a PWM value determined by TCMPBn.

For a higher PWM value, decrease the TCMPBn value. For a lower PWM value, increase the TCMPBn value. If an output inverter is enabled, the increment/decrement may be reversed.

The double buffering function allows the TCMPBn, for the next PWM cycle, written at any point in the current PWM cycle by ISR or other routine.
OUTPUT LEVEL CONTROL

The following procedure describes how to maintain TOUT as high or low (assume the inverter is off):

1. Turn off the auto reload bit. And then, TOUTn goes to high level and the timer is stopped after the TCNTn reaches 0 (recommended).

2. Stop the timer by clearing the timer start/stop bit to 0. If TCNTn \( \leq \) TCMPn, the output level is high. If TCNTn > TCMPn, the output level is low.

3. The TOUTn can be inverted by the inverter on/off bit in TCON. The inverter removes the additional circuit to adjust the output level.
DEAD ZONE GENERATOR

The dead zone is for the PWM control in a power device. This function enables the insertion of the time gap between a turn-off of a switching device and a turn on of another switching device. This time gap prohibits the two switching devices from being turned on simultaneously, even for a very short time.

TOUT0 is the PWM output. nTOUT0 is the inversion of the TOUT0. If the dead zone is enabled, the output waveform of TOUT0 and nTOUT0 will be TOUT0_DZ and nTOUT0_DZ, respectively. nTOUT0_DZ is routed to the TOUT1 pin.

In the dead zone interval, TOUT0_DZ and nTOUT0_DZ can never be turned on simultaneously.

Figure 10-7. The Wave Form When a Dead Zone Feature is Enabled
DMA REQUEST MODE

The PWM timer can generate a DMA request at every specific time. The timer keeps DMA request signals (nDMA_REQ) low until the timer receives an ACK signal. When the timer receives the ACK signal, it makes the request signal inactive. The timer, which generates the DMA request, is determined by setting DMA mode bits (in TCFG1 register). If one of timers is configured as DMA request mode, that timer does not generate an interrupt request. The others can generate interrupt normally.

DMA mode configuration and DMA / interrupt operation

<table>
<thead>
<tr>
<th>DMA Mode</th>
<th>DMA Request</th>
<th>Timer0 INT</th>
<th>Timer1 INT</th>
<th>Timer2 INT</th>
<th>Timer3 INT</th>
<th>Timer4 INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>No select</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>0001</td>
<td>Timer0</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>0010</td>
<td>Timer1</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>0011</td>
<td>Timer2</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>0100</td>
<td>Timer3</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>0101</td>
<td>Timer4</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td>0110</td>
<td>No select</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
</tbody>
</table>

![Figure 10-8. Timer4 DMA Mode Operation](image-url)
PWM TIMER CONTROL REGISTERS

TIMER CONFIGURATION REGISTER0 (TCFG0)

Timer input clock Frequency = \( \frac{PCLK}{(\text{prescaler value} + 1) \div (\text{divider value})} \)

(prescaler value) = 0~255
(divider value) = 2, 4, 8, 16

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCFG0</td>
<td>0x51000000</td>
<td>R/W</td>
<td>Configures the two 8-bit prescalers</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCFG0</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[31:24]</td>
<td></td>
<td>0x00</td>
</tr>
<tr>
<td>Dead zone length</td>
<td>[23:16]</td>
<td>These 8 bits determine the dead zone length. The 1 unit time of the dead zone length is equal to that of timer 0.</td>
<td>0x00</td>
</tr>
<tr>
<td>Prescaler 1</td>
<td>[15:8]</td>
<td>These 8 bits determine prescaler value for Timer 2, 3 and 4.</td>
<td>0x00</td>
</tr>
<tr>
<td>Prescaler 0</td>
<td>[7:0]</td>
<td>These 8 bits determine prescaler value for Timer 0 and 1.</td>
<td>0x00</td>
</tr>
</tbody>
</table>
TIMER CONFIGURATION REGISTER1 (TCFG1)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCFG1</td>
<td>0x51000004</td>
<td>R/W</td>
<td>5-MUX &amp; DMA mode selection register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCFG1 Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[31:24]</td>
<td>00000000</td>
</tr>
<tr>
<td>DMA mode</td>
<td>[23:20] Select DMA request channel</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000 = No select (all interrupt)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 = Timer0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0010 = Timer1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0011 = Timer2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0100 = Timer3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0101 = Timer4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110 = Reserved</td>
<td></td>
</tr>
<tr>
<td>MUX 4</td>
<td>[19:16] Select MUX input for PWM Timer4.</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000 = 1/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 = 1/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0010 = 1/8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0011 = 1/16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01xx = External TCLK1</td>
<td></td>
</tr>
<tr>
<td>MUX 3</td>
<td>[15:12] Select MUX input for PWM Timer3.</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000 = 1/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 = 1/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0010 = 1/8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0011 = 1/16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01xx = External TCLK1</td>
<td></td>
</tr>
<tr>
<td>MUX 2</td>
<td>[11:8] Select MUX input for PWM Timer2.</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000 = 1/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 = 1/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0010 = 1/8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0011 = 1/16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01xx = External TCLK1</td>
<td></td>
</tr>
<tr>
<td>MUX 1</td>
<td>[7:4] Select MUX input for PWM Timer1.</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000 = 1/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 = 1/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0010 = 1/8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0011 = 1/16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01xx = External TCLK0</td>
<td></td>
</tr>
<tr>
<td>MUX 0</td>
<td>[3:0] Select MUX input for PWM Timer0.</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000 = 1/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 = 1/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0010 = 1/8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0011 = 1/16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01xx = External TCLK0</td>
<td></td>
</tr>
</tbody>
</table>
## TIMER CONTROL (TCON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCON</td>
<td>0x51000008</td>
<td>R/W</td>
<td>Timer control register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TCON Bits Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>[22]</td>
<td>Timer 4 auto reload on/off</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine auto reload on/off for Timer 4.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = One-shot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Interval mode (auto reload)</td>
<td></td>
</tr>
<tr>
<td>[21]</td>
<td>Timer 4 manual update (note)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine the manual update for Timer 4.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = No operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Update TCNTB4</td>
<td></td>
</tr>
<tr>
<td>[20]</td>
<td>Timer 4 start/stop</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine start/stop for Timer 4.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Stop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Start for Timer 4</td>
<td></td>
</tr>
<tr>
<td>[19]</td>
<td>Timer 3 auto reload on/off</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine auto reload on/off for Timer 3.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = One-shot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Interval mode (auto reload)</td>
<td></td>
</tr>
<tr>
<td>[18]</td>
<td>Timer 3 output inverter on/off (note)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine output inverter on/off for Timer 3.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Inverter off</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Inverter on for TOUT3</td>
<td></td>
</tr>
<tr>
<td>[17]</td>
<td>Timer 3 manual update (note)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine manual update for Timer 3.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = No operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Update TCNTB3 &amp; TCMPB3</td>
<td></td>
</tr>
<tr>
<td>[16]</td>
<td>Timer 3 start/stop</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine start/stop for Timer 3.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Stop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Start for Timer 3</td>
<td></td>
</tr>
<tr>
<td>[15]</td>
<td>Timer 2 auto reload on/off</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine auto reload on/off for Timer 2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = One-shot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Interval mode (auto reload)</td>
<td></td>
</tr>
<tr>
<td>[14]</td>
<td>Timer 2 output inverter on/off (note)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine output inverter on/off for Timer 2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Inverter off</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Inverter on for TOUT2</td>
<td></td>
</tr>
<tr>
<td>[13]</td>
<td>Timer 2 manual update (note)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine the manual update for Timer 2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = No operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Update TCNTB2 &amp; TCMPB2</td>
<td></td>
</tr>
<tr>
<td>[12]</td>
<td>Timer 2 start/stop</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine start/stop for Timer 2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Stop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Start for Timer 2</td>
<td></td>
</tr>
<tr>
<td>[11]</td>
<td>Timer 1 auto reload on/off</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine the auto reload on/off for Timer1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = One-shot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Interval mode (auto reload)</td>
<td></td>
</tr>
<tr>
<td>[10]</td>
<td>Timer 1 output inverter on/off (note)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine the output inverter on/off for Timer1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Inverter off</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Inverter on for TOUT1</td>
<td></td>
</tr>
<tr>
<td>[9]</td>
<td>Timer 1 manual update (note)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine the manual update for Timer1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = No operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Update TCNTB1 &amp; TCMPB1</td>
<td></td>
</tr>
<tr>
<td>[8]</td>
<td>Timer 1 start/stop</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Determine start/stop for Timer 1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Stop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Start for Timer 1</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** The bits have to be cleared at next writing.
### TIMER CONTROL (TCON) REGISTER (Continued)

<table>
<thead>
<tr>
<th>TCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:5]</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
| Dead zone enable | [4]       | Determine the dead zone operation.  
0 = Disable     1 = Enable                          | 0             |
| Timer 0 auto reload on/off | [3]       | Determine auto reload on/off for Timer 0.  
0 = One-shot     1 = Interval mode(auto reload)               | 0             |
| Timer 0 output inverter on/off | [2]       | Determine the output inverter on/off for Timer 0.  
0 = Inverter off     1 = Inverter on for TOUT0 | 0             |
| Timer 0 manual update (note) | [1]       | Determine the manual update for Timer 0.  
0 = No operation     1 = Update TCNTB0 & TCMPB0 | 0             |
| Timer 0 start/stop | [0]       | Determine start/stop for Timer 0.  
0 = Stop     1 = Start for Timer 0                        | 0             |

**NOTE:** The bit have to be cleared at next writing.
### TIMER 0 COUNT BUFFER REGISTER & COMPARE BUFFER REGISTER (TCNTB0/TCMPB0)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTB0</td>
<td>0x5100000C</td>
<td>R/W</td>
<td>Timer 0 count buffer register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>TCMPB0</td>
<td>0x51000010</td>
<td>R/W</td>
<td>Timer 0 compare buffer register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### Timer 0 compare buffer register
- **Bit**: [15:0]
- **Description**: Set compare buffer value for Timer 0
- **Initial State**: 0x00000000

### Timer 0 count buffer register
- **Bit**: [15:0]
- **Description**: Set count buffer value for Timer 0
- **Initial State**: 0x00000000

### TIMER 0 COUNT OBSERVATION REGISTER (TCNTO0)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTO0</td>
<td>0x51000014</td>
<td>R</td>
<td>Timer 0 count observation register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### Timer 0 observation register
- **Bit**: [15:0]
- **Description**: Set count observation value for Timer 0
- **Initial State**: 0x00000000
### TIMER 1 COUNT BUFFER REGISTER & COMPARE BUFFER REGISTER (TCNTB1/TCMPB1)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTB1</td>
<td>0x51000018</td>
<td>R/W</td>
<td>Timer 1 count buffer register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>TCMPB1</td>
<td>0x5100001C</td>
<td>R/W</td>
<td>Timer 1 compare buffer register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TCMPB1

<table>
<thead>
<tr>
<th>Bit Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer 1 compare buffer register [15:0]</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TCNTB1

<table>
<thead>
<tr>
<th>Bit Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer 1 count buffer register [15:0]</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TIMER 1 COUNT OBSERVATION REGISTER (TCNTO1)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTO1</td>
<td>0x51000020</td>
<td>R</td>
<td>Timer 1 count observation register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TCNTO1

<table>
<thead>
<tr>
<th>Bit Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer 1 observation register [15:0]</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
## TIMER 2 COUNT BUFFER REGISTER & COMPARE BUFFER REGISTER (TCNTB2/TCMPB2)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTB2</td>
<td>0x51000024</td>
<td>R/W</td>
<td>Timer 2 count buffer register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>TCMPB2</td>
<td>0x51000028</td>
<td>R/W</td>
<td>Timer 2 compare buffer register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TCMPB2

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15:0]</td>
<td>Set compare buffer value for Timer 2</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TCNTB2

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15:0]</td>
<td>Set count buffer value for Timer 2</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

## TIMER 2 COUNT OBSERVATION REGISTER (TCNTO2)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTO2</td>
<td>0x5100002C</td>
<td>R</td>
<td>Timer 2 count observation register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TCNTO2

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15:0]</td>
<td>Set count observation value for Timer 2</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
### TIMER 3 COUNT BUFFER REGISTER & COMPARE BUFFER REGISTER (TCNTB3/TCMPB3)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTB3</td>
<td>0x51000030</td>
<td>R/W</td>
<td>Timer 3 count buffer register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>TCMPB3</td>
<td>0x51000034</td>
<td>R/W</td>
<td>Timer 3 compare buffer register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCMPB3</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer 3 compare buffer register</td>
<td>[15:0]</td>
<td>Set compare buffer value for Timer 3</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCNTB3</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer 3 count buffer register</td>
<td>[15:0]</td>
<td>Set count buffer value for Timer 3</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TIMER 3 COUNT OBSERVATION REGISTER (TCNTO3)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTO3</td>
<td>0x51000038</td>
<td>R</td>
<td>Timer 3 count observation register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCNTO3</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer 3 observation register</td>
<td>[15:0]</td>
<td>Set count observation value for Timer 3</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
### TIMER 4 COUNT BUFFER REGISTER (TCNTB4)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTB4</td>
<td>0x5100003C</td>
<td>R/W</td>
<td>Timer 4 count buffer register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

#### TCNTB4 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15:0]</td>
<td>Set count buffer value for Timer 4</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### TIMER 4 COUNT OBSERVATION REGISTER (TCNTO4)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNTO4</td>
<td>0x51000040</td>
<td>R</td>
<td>Timer 4 count observation register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

#### TCNTO4 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15:0]</td>
<td>Set count observation value for Timer 4</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
OVERVIEW

The S3C2410X UART (Universal Asynchronous Receiver and Transmitter) provides three independent asynchronous serial I/O (SIO) ports, each of which can operate in Interrupt-based or DMA-based mode. In other words, the UART can generate an interrupt or a DMA request to transfer data between CPU and the UART. The UART can support bit rates of up to 230.4K bps using system clock. If an external device provides the UART with UCLK, then the UART can operate at higher speed. Each UART channel contains two 16-byte FIFOs for receive and transmit.

The S3C2410X UART includes programmable baud rates, infra-red (IR) transmit/receive, one or two stop bit insertion, 5-bit, 6-bit, 7-bit or 8-bit data width and parity checking.

Each UART contains a baud-rate generator, a transmitter, a receiver and a control unit, as shown in Figure11-1. The baud-rate generator can be clocked by PCLK or UCLK. The transmitter and the receiver contain 16-byte FIFOs and data shifters. Data is written to FIFO and then copied to the transmit shifter before being transmitted. The data is then shifted out by the transmit data pin (TxDn). Meanwhile, received data is shifted from the receive data pin (RxDn), and then copied to FIFO from the shifter.

FEATURES

- RxD0, TxD0, RxD1, TxD1, RxD2, and TxD2 with DMA-based or interrupt-based operation
- UART Ch 0, 1, and 2 with IrDA 1.0 & 16-byte FIFO
- UART Ch 0 and 1 with nRTS0, nCTS0, nRTS1, and nCTS1
- Supports handshake transmit/receive
In FIFO mode, all 16 Byte of Buffer register are used as FIFO register.
In non-FIFO mode, only 1 Byte of Buffer register is used as Holding register.

Figure 11-1. UART Block Diagram (with FIFO)
UART OPERATION

The following sections describe the UART operations that include data transmission, data reception, interrupt generation, baud-rate generation, Loopback mode, Infra-red mode, and auto flow control.

Data Transmission

The data frame for transmission is programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits, which can be specified by the line control register (ULCONn). The transmitter can also produce the break condition, which forces the serial output to logic 0 state for one frame transmission time. This block transmits break signals after the present transmission word is transmitted completely. After the break signal transmission, it continuously transmits data into the Tx FIFO (Tx holding register in the case of Non-FIFO mode).

Data Reception

Like the transmission, the data frame for reception is also programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits in the line control register (ULCONn). The receiver can detect overrun error, parity error, frame error and break condition, each of which can set an error flag.

— The overrun error indicates that new data has overwritten the old data before the old data has been read.
— The parity error indicates that the receiver has detected an unexpected parity condition.
— The frame error indicates that the received data does not have a valid stop bit.
— The break condition indicates that the RxDn input is held in the logic 0 state for a duration longer than one frame transmission time.

Receive time-out condition occurs when it does not receive any data during the 3 word time (this interval follows the setting of Word Length bit) and the Rx FIFO is not empty in the FIFO mode.
Auto Flow Control (AFC)

The S3C2410X's UART 0 and UART 1 support auto flow control with nRTS and nCTS signals. In case, it can be connected to external UARTs. If users want to connect a UART to a Modem, disable auto flow control bit in UMCONn register and control the signal of nRTS by software.

In AFC, nRTS depends on the condition of the receiver and nCTS signals control the operation of the transmitter. The UART's transmitter transfers the data in FIFO only when nCTS signals are activated (in AFC, nCTS means that other UART's FIFO is ready to receive data). Before the UART receives data, nRTS has to be activated when its receive FIFO has a spare more than 2-byte and has to be inactivated when its receive FIFO has a spare under 1-byte (in AFC, nRTS means that its own receive FIFO is ready to receive data).

![Figure 11-2. UART AFC Interface](image)

**NOTE:** UART 2 does not support AFC function, because the S3C2410X has no nRTS2 and nCTS2.

**Example of Non Auto-Flow control (controlling nRTS and nCTS by software)**

**Rx operation with FIFO**
1. Select receive mode (Interrupt or DMA mode).
2. Check the value of Rx FIFO count in UFSTATn register. If the value is less than 15, users have to set the value of UMCONn[0] to '1' (activating nRTS), and if it is equal or larger than 15 users have to set the value to '0' (inactivating nRTS).
3. Repeat the Step 2.

**Tx operation with FIFO**
1. Select transmit mode (Interrupt or DMA mode).
2. Check the value of UMSTATn[0]. If the value is '1' (activating nCTS), users write the data to Tx FIFO register.
RS-232C interface

If users want to connect the UART to modem interface (instead of null modem), nRTS, nCTS, nDSR, nDTR, DCD and nRI signals are needed. In this case, the users can control these signals with general I/O ports by software because the AFC does not support the RS-232C interface.

Interrupt/DMA Request Generation

Each UART of the S3C2410X has seven status (Tx/Rx/Error) signals: Overrun error, Parity error, Frame error, Break, Receive buffer data ready, Transmit buffer empty, and Transmit shifter empty, all of which are indicated by the corresponding UART status register (UTRSTATn/UERSTATn).

The overrun error, parity error, frame error and break condition are referred to as the receive error status, each of which can cause the receive error status interrupt request, if the receive-error-status-interrupt-enable bit is set to one in the control register, UCONn. When a receive-error-status-interrupt-request is detected, the signal causing the request can be identified by reading the value of UERSTSTn.

When the receiver transfers the data of the receive shifter to the receive FIFO register in FIFO mode and the number of received data reaches Rx FIFO Trigger Level, Rx interrupt is generated, if Receive mode in control register (UCONn) is selected as 1 (Interrupt request or polling mode).

In the Non-FIFO mode, transferring the data of the receive shifter to the receive holding register will cause Rx interrupt under the Interrupt request and polling mode.

When the transmitter transfers data from its transmit FIFO register to its transmit shifter and the number of data left in transmit FIFO reaches Tx FIFO Trigger Level, Tx interrupt is generated, if Transmit mode in control register is selected as Interrupt request or polling mode.

In the Non-FIFO mode, transferring data from the transmit holding register to the transmit shifter will cause Tx interrupt under the Interrupt request and polling mode.

If the Receive mode and Transmit mode in control register are selected as the DMAn request mode then DMAn request occurs instead of Rx or Tx interrupt in the situation mentioned above.

<table>
<thead>
<tr>
<th>Type</th>
<th>FIFO Mode</th>
<th>Non-FIFO Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rx interrupt</td>
<td>Generated whenever receive data reaches the trigger level of receive FIFO.</td>
<td>Generated by the receive holding register whenever receive buffer becomes full.</td>
</tr>
<tr>
<td></td>
<td>Generated when the number of data in FIFO does not reach Rx FIFO trigger Level and does not receive any data during 3 word time (receive time out). This interval follows the setting of Word Length bit.</td>
<td></td>
</tr>
<tr>
<td>Tx interrupt</td>
<td>Generated whenever transmit data reaches the trigger level of transmit FIFO (Tx FIFO trigger Level).</td>
<td>Generated by the transmit holding register whenever transmit buffer becomes empty.</td>
</tr>
<tr>
<td>Error interrupt</td>
<td>Generated when frame error, parity error, or break signal are detected.</td>
<td>Generated by all errors. However if another error occurs at the same time, only one interrupt is generated.</td>
</tr>
<tr>
<td></td>
<td>Generated when it gets to the top of the receive FIFO without reading out data in it (overrun error).</td>
<td></td>
</tr>
</tbody>
</table>
UART Error Status FIFO

UART has the error status FIFO besides the Rx FIFO register. The error status FIFO indicates which data, among FIFO registers, is received with an error. The error interrupt will be issued only when the data, which has an error, is ready to read out. To clear the error status FIFO, the URXHn with an error and UERSTATn must be read out.

For example,

It is assumed that the UART Rx FIFO receives A, B, C, D, and E characters sequentially and the frame error occurs while receiving 'B', and the parity error occurs while receiving 'D'.

The actual UART receive error will not generate any error interrupt because the character, which was received with an error, has not been read yet. The error interrupt will occur when the character is read out.

Figure 11-3 shows the UART receiving the five characters including the two errors.

<table>
<thead>
<tr>
<th>Time</th>
<th>Sequence Flow</th>
<th>Error Interrupt</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0</td>
<td>When no character is read out</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>A, B, C, D, and E is received</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>After A is read out</td>
<td>The frame error (in B) interrupt</td>
<td>The ‘B’ has to be read out.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>occurs.</td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>After B is read out</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>After C is read out</td>
<td>The parity error (in D) interrupt</td>
<td>The ‘D’ has to be read out.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>occurs.</td>
<td></td>
</tr>
<tr>
<td>#5</td>
<td>After D is read out</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>#6</td>
<td>After E is read out</td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11-3. UART Receiving 5 Characters with 2 Errors
Baud-Rate Generation

Each UART’s baud-rate generator provides the serial clock for the transmitter and the receiver. The source clock for the baud-rate generator can be selected with the S3C2410X’s internal system clock or UCLK. In other words, the dividend is selectable by setting Clock Selection of UCONn. The baud-rate clock is generated by dividing the source clock (PCLK or UCLK) by 16 and a 16-bit divisor specified in the UART baud-rate divisor register (UBRDIVn). The UBRDIVn can be determined by the following expression:

$$\text{UBRDIVn} = \left( \text{int} \left( \frac{\text{PCLK}}{(\text{bps} \times 16)} \right) \right) - 1$$

Where, the divisor should be from 1 to \((2^{16} - 1)\).

For accurate UART operation, the S3C2410X also supports UCLK as a dividend. If the S3C2410X uses UCLK, which is supplied by an external UART device or system, then the serial clock of UART is exactly synchronized with UCLK. So, the user can get the more precise UART operation. The UBRDIVn can be determined:

$$\text{UBRDIVn} = \left( \text{int} \left( \frac{\text{UCLK}}{(\text{bps} \times 16)} \right) \right) - 1$$

Where, the divisor should be from 1 to \((2^{16} - 1)\) and UCLK should be smaller than PCLK.

For example, if the baud-rate is 115200 bps and PCLK or UCLK is 40 MHz, UBRDIVn is determined:

$$\text{UBRDIVn} = \left( \text{int} \left( \frac{40000000}{(115200 \times 16)} \right) \right) - 1$$

$$= \left( \text{int} \left( 21.7 \right) \right) - 1$$

$$= 21 - 1 = 20$$

UART baud-rate generator error tolerance

UART 10-bit time error should be less than 1.87\%(3/160).

$$\text{tUPCLK} = \left( \text{UBRDIVn} + 1 \right) \times 16 \times 10 / \text{PCLK} \quad \text{tUPCLK} : \text{Real UART 10-bit time}$$

$$\text{tUEXACT} = 10 / \text{baud-rate} \quad \text{tUEXACT} : \text{Ideal UART 10-bit time}$$

UART error = \((\text{tUPCLK} - \text{tUEXACT}) / \text{tUEXACT} \times 100\%\)

Loopback Mode

The S3C2410X UART provides a test mode referred to as the Loopback mode, to aid in isolating faults in the communication link. This mode structurally enables the connection of RXD and TXD in the UART. In this mode, therefore, transmitted data is received to the receiver, via RXD. This feature allows the processor to verify the internal transmit and to receive the data path of each SIO channel. This mode can be selected by setting the loopback bit in the UART control register (UCONn).

Break Condition

The break is defined as a continuous low level signal for one frame transmission time on the transmit data output.
Infra-Red (IR) Mode

The S3C2410X UART block supports infra-red (IR) transmission and reception, which can be selected by setting the Infra-red-mode bit in the UART line control register (ULCONn). Figure 11-4 illustrates how to implement the IR mode.

In IR transmit mode, the transmit pulse comes out at a rate of 3/16, the normal serial transmit rate (when the transmit data bit is zero); In IR receive mode, the receiver must detect the 3/16 pulsed period to recognize a zero value (see the frame timing diagrams shown in Figure 11-6 and 11-7).

Figure 11-4. IrDA Function Block Diagram
Figure 11-5. Serial I/O Frame Timing Diagram (Normal UART)

Figure 11-6. Infra-Red Transmit Mode Frame Timing Diagram

Figure 11-7. Infra-Red Receive Mode Frame Timing Diagram
UART SPECIAL REGISTERS

UART LINE CONTROL REGISTER

There are three UART line control registers including ULCON0, ULCON1, and ULCON2 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULCON0</td>
<td>0x50000000</td>
<td>R/W</td>
<td>UART channel 0 line control register</td>
<td>0x00</td>
</tr>
<tr>
<td>ULCON1</td>
<td>0x50004000</td>
<td>R/W</td>
<td>UART channel 1 line control register</td>
<td>0x00</td>
</tr>
<tr>
<td>ULCON2</td>
<td>0x50008000</td>
<td>R/W</td>
<td>UART channel 2 line control register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ULCONn</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Normal mode operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Infra-Red Tx/Rx mode</td>
</tr>
<tr>
<td>Parity Mode</td>
<td>[5:3]</td>
<td>Specify the type of parity generation and checking during UART transmit and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>receive operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xx = No parity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100 = Odd parity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101 = Even parity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>110 = Parity forced/checked as 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>111 = Parity forced/checked as 0</td>
</tr>
<tr>
<td>Number of Stop Bit</td>
<td>[2]</td>
<td>Specify how many stop bits are to be used for end-of-frame signal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = One stop bit per frame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Two stop bit per frame</td>
</tr>
<tr>
<td>Word Length</td>
<td>[1:0]</td>
<td>Indicate the number of data bits to be transmitted or received per frame.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 5-bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 6-bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 7-bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 8-bits</td>
</tr>
</tbody>
</table>
UART CONTROL REGISTER

There are three UART control registers including UCON0, UCON1 and UCON2 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCON0</td>
<td>0x50000004</td>
<td>R/W</td>
<td>UART channel 0 control register</td>
<td>0x00</td>
</tr>
<tr>
<td>UCON1</td>
<td>0x50004004</td>
<td>R/W</td>
<td>UART channel 1 control register</td>
<td>0x00</td>
</tr>
<tr>
<td>UCON2</td>
<td>0x50008004</td>
<td>R/W</td>
<td>UART channel 2 control register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UCONn Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
</table>
| Clock Selection [10] | Select PCLK or UCLK for the UART baud rate.  
0 = PCLK : UBRDIVn = (int)(PCLK / (bps x 16)) -1  
1 = UCLK(@GPH8) : UBRDIVn = (int)(UCLK / (bps x 16)) -1 | 0             |
| Tx Interrupt Type [9] | Interrupt request type.  
0 = Pulse (Interrupt is requested as soon as the Tx buffer becomes empty in Non-FIFO mode or reaches Tx FIFO Trigger Level in FIFO mode.)  
1 = Level (Interrupt is requested while Tx buffer is empty in Non-FIFO mode or reaches Tx FIFO Trigger Level in FIFO mode.) | 0             |
| Rx Interrupt Type [8] | Interrupt request type.  
0 = Pulse (Interrupt is requested the instant Rx buffer receives the data in Non-FIFO mode or reaches Rx FIFO Trigger Level in FIFO mode.)  
1 = Level (Interrupt is requested while Rx buffer is receiving data in Non-FIFO mode or reaches Rx FIFO Trigger Level in FIFO mode.) | 0             |
| Rx Time Out Enable [7] | Enable/Disable Rx time out interrupt when UART FIFO is enabled. The interrupt is a receive interrupt.  
0 = Disable  
1 = Enable | 0             |
| Rx Error Status Interrupt Enable [6] | Enable the UART to generate an interrupt upon an exception, such as a break, frame error, parity error, or overrun error during a receive operation.  
0 = Do not generate receive error status interrupt.  
1 = Generate receive error status interrupt | 0             |
| Loopback Mode [5] | Setting loopback bit to 1 causes the UART to enter the loopback mode. This mode is provided for test purposes only.  
0 = Normal operation  
1 = Loopback mode | 0             |
| Send Break Signal [4] | Setting this bit causes the UART to send a break during 1 frame time. This bit is automatically cleared after sending the break signal.  
0 = Normal transmit  
1 = Send break signal | 0             |
UART CONTROL REGISTER (CONTINUED)

<table>
<thead>
<tr>
<th>Transmit Mode</th>
<th>[3:2]</th>
<th>Determine which function is currently able to write Tx data to the UART transmit buffer register.</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>00 = Disable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = Interrupt request or polling mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = DMA0 request (Only for UART0),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMA3 request (Only for UART2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = DMA1 request (Only for UART1)</td>
<td></td>
</tr>
<tr>
<td>Receive Mode</td>
<td>[1:0]</td>
<td>Determine which function is currently able to read data from UART receive buffer register.</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = Disable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = Interrupt request or polling mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = DMA0 request (Only for UART0),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMA3 request (Only for UART2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = DMA1 request (Only for UART1)</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** When the UART does not reach the FIFO trigger level and does not receive data during 3 word time in DMA receive mode with FIFO, the Rx interrupt will be generated (receive time out), and the users should check the FIFO status and read out the rest.
UART FIFO CONTROL REGISTER

There are three UART FIFO control registers including UFCON0, UFCON1 and UFCON2 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFCON0</td>
<td>0x50000008</td>
<td>R/W</td>
<td>UART channel 0 FIFO control register</td>
<td>0x0</td>
</tr>
<tr>
<td>UFCON1</td>
<td>0x50004008</td>
<td>R/W</td>
<td>UART channel 1 FIFO control register</td>
<td>0x0</td>
</tr>
<tr>
<td>UFCON2</td>
<td>0x50008008</td>
<td>R/W</td>
<td>UART channel 2 FIFO control register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UFCONn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx FIFO Trigger Level</td>
<td>[7:6]</td>
<td>Determine the trigger level of transmit FIFO.</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = Empty</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 4-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 8-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 12-byte</td>
<td></td>
</tr>
<tr>
<td>Rx FIFO Trigger Level</td>
<td>[5:4]</td>
<td>Determine the trigger level of receive FIFO.</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 4-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = 8-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = 12-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = 16-byte</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>[3]</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Tx FIFO Reset</td>
<td>[2]</td>
<td>Auto-cleared after resetting FIFO</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Normal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Tx FIFO reset</td>
<td></td>
</tr>
<tr>
<td>Rx FIFO Reset</td>
<td>[1]</td>
<td>Auto-cleared after resetting FIFO</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Normal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Rx FIFO reset</td>
<td></td>
</tr>
<tr>
<td>FIFO Enable</td>
<td>[0]</td>
<td>0 = Disable</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Enable</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: When the UART does not reach the FIFO trigger level and does not receive data during 3 word time in DMA receive mode with FIFO, the Rx interrupt will be generated (receive time out), and the users should check the FIFO status and read out the rest.
UART MODEM CONTROL REGISTER

There are two UART MODEM control registers including UMCON0 and UMCON1 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCON0</td>
<td>0x5000000C</td>
<td>R/W</td>
<td>UART channel 0 Modem control register</td>
<td>0x0</td>
</tr>
<tr>
<td>UMCON1</td>
<td>0x5000400C</td>
<td>R/W</td>
<td>UART channel 1 Modem control register</td>
<td>0x0</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5000800C</td>
<td>-</td>
<td>Reserved</td>
<td>Undef</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UMCONn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:5]</td>
<td>These bits must be 0's</td>
<td>00</td>
</tr>
<tr>
<td>Auto Flow Control (AFC)</td>
<td>[4]</td>
<td>0 = Disable 1 = Enable</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[3:1]</td>
<td>These bits must be 0's</td>
<td>00</td>
</tr>
<tr>
<td>Request to Send</td>
<td>[0]</td>
<td>If AFC bit is enabled, this value will be ignored. In this case the S3C2410X will control nRTS automatically. If AFC bit is disabled, nRTS must be controlled by software. 0 = 'H' level (Inactivate nRTS) 1 = 'L' level (Activate nRTS)</td>
<td>0</td>
</tr>
</tbody>
</table>

NOTE: UART 2 does not support AFC function, because the S3C2410X has no nRTS2 and nCTS2.
# UART TX/RX STATUS REGISTER

There are three UART Tx/Rx status registers including UTRSTAT0, UTRSTAT1 and UTRSTAT2 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTRSTAT0</td>
<td>0x50000010</td>
<td>R</td>
<td>UART channel 0 Tx/Rx status register</td>
<td>0x6</td>
</tr>
<tr>
<td>UTRSTAT1</td>
<td>0x50004010</td>
<td>R</td>
<td>UART channel 1 Tx/Rx status register</td>
<td>0x6</td>
</tr>
<tr>
<td>UTRSTAT2</td>
<td>0x50008010</td>
<td>R</td>
<td>UART channel 2 Tx/Rx status register</td>
<td>0x6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UTRSTATn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitter empty</td>
<td>[2]</td>
<td>Set to 1 automatically when the transmit buffer register has no valid data to transmit and the transmit shift register is empty. 0 = Not empty 1 = Transmitter (transmit buffer &amp; shifter register) empty</td>
<td>1</td>
</tr>
<tr>
<td>Transmit buffer empty</td>
<td>[1]</td>
<td>Set to 1 automatically when transmit buffer register is empty. 0 = The buffer register is not empty 1 = Empty  (In Non-FIFO mode, Interrupt or DMA is requested. In FIFO mode, Interrupt or DMA is requested, when Tx FIFO Trigger Level is set to 00 (Empty)) If the UART uses the FIFO, users should check Tx FIFO Count bits and Tx FIFO Full bit in the UFSTAT register instead of this bit.</td>
<td>1</td>
</tr>
<tr>
<td>Receive buffer data ready</td>
<td>[0]</td>
<td>Set to 1 automatically whenever receive buffer register contains valid data, received over the RXDn port. 0 = Empty 1 = The buffer register has a received data  (In Non-FIFO mode, Interrupt or DMA is requested) If the UART uses the FIFO, users should check Rx FIFO Count bits and Rx FIFO Full bit in the UFSTAT register instead of this bit.</td>
<td>0</td>
</tr>
</tbody>
</table>
UART ERROR STATUS REGISTER

There are three UART Rx error status registers including UERSTAT0, UERSTAT1 and UERSTAT2 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UERSTAT0</td>
<td>0x50000014</td>
<td>R</td>
<td>UART channel 0 Rx error status register</td>
<td>0x0</td>
</tr>
<tr>
<td>UERSTAT1</td>
<td>0x50004014</td>
<td>R</td>
<td>UART channel 1 Rx error status register</td>
<td>0x0</td>
</tr>
<tr>
<td>UERSTAT2</td>
<td>0x50008014</td>
<td>R</td>
<td>UART channel 2 Rx error status register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UERSTATn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[3]</td>
<td>0 = No frame error during receive</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Frame error (Interrupt is requested.)</td>
<td></td>
</tr>
<tr>
<td>Frame Error</td>
<td>[2]</td>
<td>Set to 1 automatically whenever a frame error occurs during receive operation.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = No frame error during receive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Frame error (Interrupt is requested.)</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>[1]</td>
<td>0 = No frame error during receive</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Frame error (Interrupt is requested.)</td>
<td></td>
</tr>
<tr>
<td>Overrun Error</td>
<td>[0]</td>
<td>Set to 1 automatically whenever an overrun error occurs during receive operation.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = No overrun error during receive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Overrun error (Interrupt is requested.)</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: These bits (UERSATn[3:0]) are automatically cleared to 0 when the UART error status register is read.
**UART FIFO STATUS REGISTER**

There are three UART FIFO status registers including UFSTAT0, UFSTAT1 and UFSTAT2 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFSTAT0</td>
<td>0x50000018</td>
<td>R</td>
<td>UART channel 0 FIFO status register</td>
<td>0x00</td>
</tr>
<tr>
<td>UFSTAT1</td>
<td>0x50004018</td>
<td>R</td>
<td>UART channel 1 FIFO status register</td>
<td>0x00</td>
</tr>
<tr>
<td>UFSTAT2</td>
<td>0x50008018</td>
<td>R</td>
<td>UART channel 2 FIFO status register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UFSTATn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[15:10]</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Tx FIFO Full</td>
<td>[9]</td>
<td>Set to 1 automatically whenever transmit FIFO is full during</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>transmit operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0 = 0$-byte $\leq$ Tx FIFO data $\leq 15$-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1 = $Full</td>
<td></td>
</tr>
<tr>
<td>Rx FIFO Full</td>
<td>[8]</td>
<td>Set to 1 automatically whenever receive FIFO is full during</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>receive operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0 = 0$-byte $\leq$ Rx FIFO data $\leq 15$-byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1 = $Full</td>
<td></td>
</tr>
<tr>
<td>Tx FIFO Count</td>
<td>[7:4]</td>
<td>Number of data in Tx FIFO</td>
<td>0</td>
</tr>
<tr>
<td>Rx FIFO Count</td>
<td>[3:0]</td>
<td>Number of data in Rx FIFO</td>
<td>0</td>
</tr>
</tbody>
</table>
UART MODEM STATUS REGISTER

There are two UART modem status registers including UMSTAT0 and UMSTAT1 in the UART block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMSTAT0</td>
<td>0x5000001C</td>
<td>R</td>
<td>UART channel 0 Modem status register</td>
<td>0x0</td>
</tr>
<tr>
<td>UMSTAT1</td>
<td>0x5000401C</td>
<td>R</td>
<td>UART channel 1 Modem status register</td>
<td>0x0</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x5000801C</td>
<td>–</td>
<td>Reserved</td>
<td>Undef</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UMSTAT0 Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[3] 0</td>
</tr>
<tr>
<td>Delta CTS</td>
<td>[2] Indicate that the nCTS input to the S3C2410X has changed state since the last time it was read by CPU. (Refer to Figure 11-8.) 0 = Has not changed 1 = Has changed</td>
</tr>
<tr>
<td>Reserved</td>
<td>[1] 0</td>
</tr>
<tr>
<td>Clear to Send</td>
<td>[0] 0 = CTS signal is not activated (nCTS pin is high.) 1 = CTS signal is activated (nCTS pin is low.)</td>
</tr>
</tbody>
</table>

Figure 11-8. nCTS and Delta CTS Timing Diagram

nCTS
Delta CTS
Read_UMSTAT

Figure 11-8. nCTS and Delta CTS Timing Diagram
UART TRANSMIT BUFFER REGISTER (HOLDING REGISTER & FIFO REGISTER)

There are three UART transmit buffer registers including UTXH0, UTXH1 and UTXH2 in the UART block. UTXHn has an 8-bit data for transmission data.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTXH0</td>
<td>0x50000020(L)</td>
<td>W</td>
<td>UART channel 0 transmit buffer register</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>0x50000023(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTXH1</td>
<td>0x50004020(L)</td>
<td>W</td>
<td>UART channel 1 transmit buffer register</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>0x50004023(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTXH2</td>
<td>0x50008020(L)</td>
<td>W</td>
<td>UART channel 2 transmit buffer register</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>0x50008023(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: (L): The endian mode is Little endian.
(B): The endian mode is Big endian.

UART RECEIVE BUFFER REGISTER (HOLDING REGISTER & FIFO REGISTER)

There are three UART receive buffer registers including URXH0, URXH1 and URXH2 in the UART block. URXHn has an 8-bit data for received data.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>URXH0</td>
<td>0x50000024(L)</td>
<td>R</td>
<td>UART channel 0 receive buffer register</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>0x50000027(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>URXH1</td>
<td>0x50004024(L)</td>
<td>R</td>
<td>UART channel 1 receive buffer register</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>0x50004027(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>URXH2</td>
<td>0x50008024(L)</td>
<td>R</td>
<td>UART channel 2 receive buffer register</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>0x50008027(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: When an overrun error occurs, the URXHn must be read. If not, the next received data will also make an overrun error, even though the overrun bit of UERSTATn had been cleared.
UART BAUD RATE DIVISOR REGISTER

There are three UART baud rate divisor registers including UBRDIV0, UBRDIV1 and UBRDIV2 in the UART block. The value stored in the baud rate divisor register (UBRDIVn), is used to determine the serial Tx/Rx clock rate (baud rate) as follows:

\[
UBRDIVn = \left( \text{int} \right) \left( \frac{PCLK}{(bps \times 16)} \right) - 1
\]

or

\[
UBRDIVn = \left( \text{int} \right) \left( \frac{UCLK}{(bps \times 16)} \right) - 1
\]

Where, the divisor should be from 1 to \(2^{16}-1\) and UCLK should be smaller than PCLK.

For example, if the baud-rate is 115200 bps and PCLK or UCLK is 40 MHz, UBRDIVn is:

\[
UBRDIVn = \left( \text{int} \right) \left( \frac{40000000}{(115200 \times 16)} \right) - 1 \\
= \left( \text{int} \right) (21.7) - 1 \\
= 21 - 1 = 20
\]

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBRDIV0</td>
<td>0x50000028</td>
<td>R/W</td>
<td>Baud rate divisor register 0</td>
<td>–</td>
</tr>
<tr>
<td>UBRDIV1</td>
<td>0x50004028</td>
<td>R/W</td>
<td>Baud rate divisor register 1</td>
<td>–</td>
</tr>
<tr>
<td>UBRDIV2</td>
<td>0x50008028</td>
<td>R/W</td>
<td>Baud rate divisor register 2</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UBRDIVn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBRDIV</td>
<td>[15:0]</td>
<td>Baud rate division value</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UBRDIVn &gt;0</td>
<td></td>
</tr>
</tbody>
</table>
OVERVIEW

S3C2410X supports 2-port USB host interface as follows:

- OHCI Rev 1.0 compatible
- USB Rev1.1 compatible
- Two downstream ports
- Support for both LowSpeed and HighSpeed USB devices

Figure 12-1. USB Host Controller Block Diagram
USB HOST CONTROLLER SPECIAL REGISTERS

The S3C2410X USB host controller complies with OHCI Rev 1.0. Refer to Open Host Controller Interface Rev 1.0 specification for detail information.

<table>
<thead>
<tr>
<th>Register</th>
<th>Base Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HcRevision</td>
<td>0x49000000</td>
<td>–</td>
<td>Control and status group</td>
<td>–</td>
</tr>
<tr>
<td>HcControl</td>
<td>0x49000004</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcCommonStatus</td>
<td>0x49000008</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcInterruptStatus</td>
<td>0x4900000C</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcInterruptEnable</td>
<td>0x49000010</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcInterruptDisable</td>
<td>0x49000014</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcHCCA</td>
<td>0x49000018</td>
<td>–</td>
<td>Memory pointer group</td>
<td>–</td>
</tr>
<tr>
<td>HcPeriodCurrentED</td>
<td>0x4900001C</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcControlHeadED</td>
<td>0x49000020</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcControlCurrentED</td>
<td>0x49000024</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcBulkHeadED</td>
<td>0x49000028</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcBulkCurrentED</td>
<td>0x4900002C</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcDoneHead</td>
<td>0x49000030</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcRmInterval</td>
<td>0x49000034</td>
<td>–</td>
<td>Frame counter group</td>
<td>–</td>
</tr>
<tr>
<td>HcFmRemaining</td>
<td>0x49000038</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcFmNumber</td>
<td>0x4900003C</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcPeriodicStart</td>
<td>0x49000040</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcLSThreshold</td>
<td>0x49000044</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcRhDescriptorA</td>
<td>0x49000048</td>
<td>–</td>
<td>Root hub group</td>
<td>–</td>
</tr>
<tr>
<td>HcRhDescriptorB</td>
<td>0x4900004C</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcRhStatus</td>
<td>0x49000050</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcRhPortStatus1</td>
<td>0x49000054</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>HcRhPortStatus2</td>
<td>0x49000058</td>
<td>–</td>
<td></td>
<td>–</td>
</tr>
</tbody>
</table>
OVERVIEW

Universal Serial Bus (USB) device controller is designed to provide a high performance full speed function controller solution with DMA interface. USB device controller allows bulk transfer with DMA, interrupt transfer and control transfer.

**USB device controller supports:**

- Full speed USB device controller compatible with the USB specification version 1.1
- DMA interface for bulk transfer
- Five endpoints with FIFO
  - EP0: 16byte (Register)
  - EP1: 64byte IN/OUT FIFO (dual port asynchronous RAM): interrupt or DMA
  - EP2: 64byte IN/OUT FIFO (dual port asynchronous RAM): interrupt or DMA
  - EP3: 64byte IN/OUT FIFO (dual port asynchronous RAM): interrupt or DMA
  - EP4: 64byte IN/OUT FIFO (dual port asynchronous RAM): interrupt or DMA
- Integrated USB Transceiver

FEATURE

- Fully compliant with USB Specification Version 1.1
- Full speed (12Mbps) device
- Integrated USB Transceiver
- Supports control, interrupt and bulk transfer
- Five endpoints with FIFO:
  - One bi-directional control endpoint with 16-byte FIFO (EP0)
  - Four bi-directional bulk endpoints with 64-byte FIFO (EP1, EP2, EP3, and EP4)
- Independent 64byte receive and transmit FIFO to maximize throughput
- Supports suspend and remote wakeup function
Figure 13-1. USB Device Controller Block Diagram
USB DEVICE CONTROLLER SPECIAL REGISTERS

This section describes detailed functionalities about register sets of USB device controller. All special function register is byte-accessible or word-accessible. If you access byte mode offset-address is different in little endian and big endian. All reserved bit is zero. Common indexed registers depend on INDEX register (INDEX_REG) (offset address: 0x178) value. For example if you want to write EP0 CSR register, you must write ‘0x00’ on the INDEX_REG before writing IN_CSR1 register.

NOTE

All register must be resettled after performing Host Reset Signaling.

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
<th>Offset Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON INDEXED REGISTERS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FUNC_ADDR_REG</td>
<td>Function address register</td>
<td>0x140(L) / 0x143(B)</td>
</tr>
<tr>
<td>PWR_REG</td>
<td>Power management register</td>
<td>0x144(L) / 0x147(B)</td>
</tr>
<tr>
<td>EP_INT_REG (EP0–EP4)</td>
<td>Endpoint interrupt register</td>
<td>0x148(L) / 0x14B(B)</td>
</tr>
<tr>
<td>USB_INT_REG</td>
<td>USB interrupt register</td>
<td>0x158(L) / 0x15B(B)</td>
</tr>
<tr>
<td>EP_INT_EN_REG (EP0–EP4)</td>
<td>Endpoint interrupt enable register</td>
<td>0x15C(L) / 0x15F(B)</td>
</tr>
<tr>
<td>USB_INT_EN_REG</td>
<td>USB Interrupt enable register</td>
<td>0x16C(L) / 0x16F(B)</td>
</tr>
<tr>
<td>FRAME_NUM1_REG</td>
<td>Frame number 1 register</td>
<td>0x170(L) / 0x173(B)</td>
</tr>
<tr>
<td>FRAME_NUM2_REG</td>
<td>Frame number 2 register</td>
<td>0x174(L) / 0x177(B)</td>
</tr>
<tr>
<td>INDEX_REG</td>
<td>Index register</td>
<td>0x178(L) / 0x17B(B)</td>
</tr>
<tr>
<td>EP0_FIFO_REG</td>
<td>Endpoint0 FIFO register</td>
<td>0x1C0(L) / 0x1C3(B)</td>
</tr>
<tr>
<td>EP1_FIFO_REG</td>
<td>Endpoint1 FIFO register</td>
<td>0x1C4(L) / 0x1C7(B)</td>
</tr>
<tr>
<td>EP2_FIFO_REG</td>
<td>Endpoint2 FIFO register</td>
<td>0x1C8(L) / 0x1CB(B)</td>
</tr>
<tr>
<td>EP3_FIFO_REG</td>
<td>Endpoint3 FIFO register</td>
<td>0x1CC(L) / 0x1CF(B)</td>
</tr>
<tr>
<td>EP4_FIFO_REG</td>
<td>Endpoint4 FIFO register</td>
<td>0x1D0(L) / 0x1D3(B)</td>
</tr>
<tr>
<td>EP1_DMA_CON</td>
<td>Endpoint1 DMA control register</td>
<td>0x200(L) / 0x203(B)</td>
</tr>
<tr>
<td>EP1_DMA_UNIT</td>
<td>Endpoint1 DMA unit counter register</td>
<td>0x204(L) / 0x207(B)</td>
</tr>
<tr>
<td>EP1_DMA_FIFO</td>
<td>Endpoint1 DMA FIFO counter register</td>
<td>0x208(L) / 0x20B(B)</td>
</tr>
<tr>
<td>EP1_DMA_TTC_L</td>
<td>Endpoint1 DMA transfer counter low-byte register</td>
<td>0x20C(L) / 0x20F(B)</td>
</tr>
<tr>
<td>EP1_DMA_TTC_M</td>
<td>Endpoint1 DMA transfer counter middle-byte register</td>
<td>0x210(L) / 0x213(B)</td>
</tr>
<tr>
<td>EP1_DMA_TTC_H</td>
<td>Endpoint1 DMA transfer counter high-byte register</td>
<td>0x214(L) / 0x217(B)</td>
</tr>
<tr>
<td>EP2_DMA_CON</td>
<td>Endpoint2 DMA control register</td>
<td>0x218(L) / 0x21B(B)</td>
</tr>
<tr>
<td>EP2_DMA_UNIT</td>
<td>Endpoint2 DMA unit counter register</td>
<td>0x21C(L) / 0x21F(B)</td>
</tr>
<tr>
<td>EP2_DMA_FIFO</td>
<td>Endpoint2 DMA FIFO counter register</td>
<td>0x220(L) / 0x223(B)</td>
</tr>
<tr>
<td>EP2_DMA_TTC_L</td>
<td>Endpoint2 DMA transfer counter low-byte register</td>
<td>0x224(L) / 0x227(B)</td>
</tr>
</tbody>
</table>
### USB Device Controller Special Registers (Continued)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
<th>Offset Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP2_DMA_TTC_M</td>
<td>Endpoint2 DMA transfer counter middle-byte register</td>
<td>0x228(L) / 0x22B(B)</td>
</tr>
<tr>
<td>EP2_DMA_TTC_H</td>
<td>Endpoint2 DMA transfer counter high-byte register</td>
<td>0x22C(L) / 0x22F(B)</td>
</tr>
<tr>
<td>EP3_DMA_CON</td>
<td>Endpoint3 DMA control register</td>
<td>0x240(L) / 0x243(B)</td>
</tr>
<tr>
<td>EP3_DMA_UNIT</td>
<td>Endpoint3 DMA unit counter register</td>
<td>0x244(L) / 0x247(B)</td>
</tr>
<tr>
<td>EP3_DMA_FIFO</td>
<td>Endpoint3 DMA FIFO counter register</td>
<td>0x248(L) / 0x24B(B)</td>
</tr>
<tr>
<td>EP3_DMA_TTC_L</td>
<td>Endpoint3 DMA transfer counter low-byte register</td>
<td>0x24C(L) / 0x24F(B)</td>
</tr>
<tr>
<td>EP3_DMA_TTC_M</td>
<td>Endpoint3 DMA transfer counter middle-byte register</td>
<td>0x250(L) / 0x253(B)</td>
</tr>
<tr>
<td>EP3_DMA_TTC_H</td>
<td>Endpoint3 DMA transfer counter high-byte register</td>
<td>0x254(L) / 0x247(B)</td>
</tr>
<tr>
<td>EP4_DMA_CON</td>
<td>Endpoint4 DMA control register</td>
<td>0x258(L) / 0x25B(B)</td>
</tr>
<tr>
<td>EP4_DMA_UNIT</td>
<td>Endpoint4 DMA unit counter register</td>
<td>0x25C(L) / 0x25F(B)</td>
</tr>
<tr>
<td>EP4_DMA_FIFO</td>
<td>Endpoint4 DMA FIFO counter register</td>
<td>0x260(L) / 0x263(B)</td>
</tr>
<tr>
<td>EP4_DMA_TTC_L</td>
<td>Endpoint4 DMA transfer counter low-byte register</td>
<td>0x264(L) / 0x267(B)</td>
</tr>
<tr>
<td>EP4_DMA_TTC_M</td>
<td>Endpoint4 DMA transfer counter middle-byte register</td>
<td>0x268(L) / 0x26B(B)</td>
</tr>
<tr>
<td>EP4_DMA_TTC_H</td>
<td>Endpoint4 DMA transfer counter high-byte register</td>
<td>0x26C(L) / 0x26F(B)</td>
</tr>
</tbody>
</table>

### COMMON INDEXED REGISTERS

<table>
<thead>
<tr>
<th>REG</th>
<th>Description</th>
<th>Offset Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXP_REG</td>
<td>Endpoint MAX packet register</td>
<td>0x18C(L) / 0x18F(B)</td>
</tr>
</tbody>
</table>

### IN INDEXED REGISTERS

<table>
<thead>
<tr>
<th>REG</th>
<th>Description</th>
<th>Offset Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_CSR1_REG/EP0_CSR</td>
<td>EP In control status register 1/EP0 control status register</td>
<td>0x184(L) / 0x187(B)</td>
</tr>
<tr>
<td>IN_CSR2_REG</td>
<td>EP In control status register 2</td>
<td>0x188(L) / 0x18B(B)</td>
</tr>
</tbody>
</table>

### OUT INDEXED REGISTERS

<table>
<thead>
<tr>
<th>REG</th>
<th>Description</th>
<th>Offset Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_CSR1_REG</td>
<td>EP out control status register 1</td>
<td>0x190(L) / 0x193(B)</td>
</tr>
<tr>
<td>OUT_CSR2_REG</td>
<td>EP out control status register 2</td>
<td>0x194(L) / 0x197(B)</td>
</tr>
<tr>
<td>OUT_FIFO_CNT1_REG</td>
<td>EP out write count register 1</td>
<td>0x198(L) / 0x19B(B)</td>
</tr>
<tr>
<td>OUT_FIFO_CNT2_REG</td>
<td>EP out write count register 2</td>
<td>0x19C(L) / 0x19F(B)</td>
</tr>
</tbody>
</table>
FUNCTION ADDRESS REGISTER (FUNC_ADDR_REG)

This register maintains the USB device controller address assigned by the host. The Micro Controller Unit (MCU) writes the value received through a SET_ADDRESS descriptor to this register. This address is used for the next token.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_ADDR_REG</td>
<td>0x52000140(L)</td>
<td>R/W</td>
<td>Function address register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000143(B)</td>
<td>(byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FUNC_ADDR_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR_UPDATE</td>
<td>[7]</td>
<td>R/SET</td>
<td>R/CLEAR</td>
<td>Set by the MCU whenever it updates the FUNCTION_ADDR field in this register.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This bit will be cleared by USB when DATA_END bit in EP0_CSR register.</td>
<td></td>
</tr>
<tr>
<td>FUNCTION_ADDR</td>
<td>[6:0]</td>
<td>R/W</td>
<td>R</td>
<td>The MCU write the unique address, assigned by host, to this field.</td>
<td>00</td>
</tr>
</tbody>
</table>
POWER MANAGEMENT REGISTER (PWR_REG)

This register acts as a power control register in the USB block.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWR_REG</td>
<td>0x52000144(L)</td>
<td>R/W</td>
<td>Power management register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000147(B)</td>
<td>byte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PWR_ADDR</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO_UPDATE</td>
<td>[7]</td>
<td>R/W</td>
<td>R</td>
<td>Used for ISO mode only. If set, GFI waits for a SOF token to set IN_PKT_RDY even though a packet to send is already loaded by MCU. If an IN token is received before a SOF token, then a zero length data packet will be sent.</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[6:4]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>USB_RESET</td>
<td>[3]</td>
<td>R</td>
<td>SET</td>
<td>Set by the USB if reset signaling is received from the host. This bit remains set as long as reset signaling persists on the bus.</td>
<td>0</td>
</tr>
</tbody>
</table>
| MCU_RESUME | [2]   | R/W   | R /
/CLEAR | Set by the MCU for MCU Resume. The USB generates the resume signaling during 10ms, if this bit is set in suspend mode. | –             |
| SUSPEND_MODE | [1] | R     | SET /
/CLEAR | Set by USB automatically when the device enter into suspend mode. It is cleared under the following conditions: 1) The MCU clears the MCU_RESUME bit by writing ‘0’, in order to end remote resume signaling. 2) The resume signal form host is received. | 0             |
| SUSPEND_EN | [0]   | R/W   | R      | Suspend mode enable control bit 0 = Disable (default). The device will not enter suspend mode. 1 = Enable suspend mode. | 0             |
INTERRUPT REGISTER (EP_INT_REG/USB_INT_REG)

The USB core has two interrupt registers. These registers act as status registers for the MCU when it is interrupted. The bits are cleared by writing a ‘1’ (not ‘0’) to each bit that was set. Once the MCU is interrupted, MCU should read the contents of interrupt-related registers and write back to clear the contents if it is necessary.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP_INT_REG</td>
<td>0x52000148(L)</td>
<td>R/W</td>
<td>EP interrupt pending/clear register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200014B(B)</td>
<td>(byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### EP_INT_REG Bit MCU USB Description Initial State

/CLEAR | SET | Description | Initial State |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>For BULK/INTERRUPT IN endpoints:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set by the USB under the following conditions:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. IN_PKT_RDY bit is cleared.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. FIFO is flushed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. SENT_STALL set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| For BULK/INTERRUPT OUT endpoints: | | | | |
| Set by the USB under the following conditions: | | | | |
| 1. Sets OUT_PKT_RDY bit | | | | |
| 2. Sets SENT_STALL bit | | | | |

| For ISO IN endpoints: | | | | |
| Set by the USB under the following conditions: | | | | |
| 1. UNDER_RUN bit is set | | | | |
| 2. IN_PKT_RDY bit is cleared. | | | | |
| 3. FIFO is flushed | | | | |
| **NOTE:** Conditions 1 and 2 are mutually exclusive | | | | |

| For ISO OUT endpoints: | | | | |
| Set by the USB under the following conditions: | | | | |
| 1. OUT_PKT_RDY bit is set | | | | |
| 2. OVER_RUN bit is set. | | | | |
| **NOTE:** Conditions 1 and 2 are mutually exclusive. | | | | |

| EP0 Interrupt | [0] | R /
/CLEAR | SET | Description | Initial State |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Correspond to endpoint 0 interrupt.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set by the USB under the following conditions:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. OUT_PKT_RDY bit is set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. IN_PKT_RDY bit is cleared.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. SENT_STALL bit is set</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. SETUP_END bit is set</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. DATA_END bit is cleared</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(it indicates the end of control transfer).</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interrupt Register (EP_INT_REG/USB_INT_REG) (Continued)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_INT_REG</td>
<td>0x52000158(L)</td>
<td>R/W</td>
<td>USB interrupt pending/clear register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200015B(B)</td>
<td>(byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USB_INT_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET Interrupt</td>
<td>[2]</td>
<td>R/CLEAR</td>
<td>SET</td>
<td>Set by the USB when it receives reset signaling.</td>
<td>0</td>
</tr>
<tr>
<td>RESUME Interrupt</td>
<td>[1]</td>
<td>R/CLEAR</td>
<td>SET</td>
<td>Set by the USB when it receives resume signaling, <em>while in Suspend mode.</em></td>
<td>0</td>
</tr>
<tr>
<td>SUSPEND Interrupt</td>
<td>[0]</td>
<td>R/CLEAR</td>
<td>SET</td>
<td>Set by the USB when it receives suspend signalizing. This bit is set whenever</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>there is no activity for 3ms on the bus. Thus, if the MCU does not stop the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>clock after the first suspend interrupt, it will continue to be interrupted</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>every 3ms as long as there is no activity on the USB bus. By default, this</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>interrupt is disabled.</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: If the RESET interrupt is occurred, all USB device registers should be re-configured.
INTERRUPT ENABLE REGISTER (EP_INT_EN_REG/USB_INT_REG)

Corresponding to each interrupt register, the USB device controller also has two interrupt enable registers (except resume interrupt enable). By default, USB reset interrupt is enabled.

If bit = 0, the interrupt is disabled.
If bit = 1, the interrupt is enabled.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP_INT_EN_REG</td>
<td>0x5200015C(L)</td>
<td>R/W</td>
<td>Determine which interrupt is enabled</td>
<td>0xFF</td>
</tr>
<tr>
<td></td>
<td>0x5200015F(B)</td>
<td>(byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EP_INT_EN_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R/W</td>
<td>R</td>
<td>EP4 Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[4]</td>
<td>R/W</td>
<td>R</td>
<td>0 = Interrupt disable    1 = Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R/W</td>
<td>R</td>
<td>EP3 Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3]</td>
<td>R/W</td>
<td>R</td>
<td>0 = Interrupt disable    1 = Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R/W</td>
<td>R</td>
<td>EP2 Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[2]</td>
<td>R/W</td>
<td>R</td>
<td>0 = Interrupt disable    1 = Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R/W</td>
<td>R</td>
<td>EP1 Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>R/W</td>
<td>R</td>
<td>0 = Interrupt disable    1 = Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R/W</td>
<td>R</td>
<td>EP0 Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>R/W</td>
<td>R</td>
<td>0 = Interrupt disable    1 = Enable</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USB_INT_EN_REG</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x5200016C(L)</td>
<td>R/W</td>
<td>Determine which interrupt is enabled</td>
<td>0x04</td>
</tr>
<tr>
<td></td>
<td>0x5200016F(B)</td>
<td>(byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INT_MASK_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET_INT_EN</td>
<td>[2]</td>
<td>R/W</td>
<td>R</td>
<td>Reset interrupt enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = Interrupt disable    1 = Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Reserved</td>
<td>[1]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>SUSPEND_INT_EN</td>
<td>[0]</td>
<td>R/W</td>
<td>R</td>
<td>Suspend interrupt enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = Interrupt disable    1 = Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
FRAME NUMBER REGISTER (FRAME_NUM1_REG/FRAME_NUM2_REG)

When the host transfers USB packets, each Start Of Frame (SOF) packet includes a frame number. The USB device controller catches this frame number and loads it into this register automatically.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAME_NUM1_REG</td>
<td>0x52000170(L)</td>
<td>R</td>
<td>Frame number lower byte register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000173(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FRAME_NUM_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAME_NUM1</td>
<td>[7:0]</td>
<td>R</td>
<td>W</td>
<td>Frame number lower byte value</td>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAME_NUM2_REG</td>
<td>0x52000174(L)</td>
<td>R</td>
<td>Frame number higher byte register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000177(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FRAME_NUM_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAME_NUM2</td>
<td>[7:0]</td>
<td>R</td>
<td>W</td>
<td>Frame number higher byte value</td>
<td>00</td>
</tr>
</tbody>
</table>
INDEX REGISTER (INDEX_REG)

The INDEX register is used to indicate certain endpoint registers effectively. The MCU can access the endpoint registers (MAXP_REG, IN_CSR1_REG, IN_CSR2_REG, OUT_CSR1_REG, OUT_CSR2_REG, OUT_FIFO_CNT1_REG, and OUT_FIFO_CNT2_REG) for an endpoint inside the core using the INDEX register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_REG</td>
<td>0x52000178(L)</td>
<td>R/W</td>
<td>Register index register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200017B(B)</td>
<td>(byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INDEX_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>[7:0]</td>
<td>R/W</td>
<td>R</td>
<td>Indicate a certain endpoint</td>
<td>00</td>
</tr>
</tbody>
</table>
## END POINT0 CONTROL STATUS REGISTER (EP0_CSR)

This register has the control and status bits for Endpoint 0. Since a control transaction is involved with both IN and OUT tokens, there is only one CSR register, mapped to the IN CSR1 register. (share IN1_CSR and can access by writing index register “0” and read/write IN1_CSR)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP0_CSR</td>
<td>0x52000184(L)</td>
<td>R/W</td>
<td>Endpoint 0 status register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000187(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EP0_CSR Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICED_SETUP_END</td>
<td>[7]</td>
<td>W</td>
<td>CLEAR The MCU should write a &quot;1&quot; to this bit to clear SETUP_END.</td>
<td>0</td>
</tr>
<tr>
<td>SERVICED_OUT_PKT_RDY</td>
<td>[6]</td>
<td>W</td>
<td>CLEAR The MCU should write a &quot;1&quot; to this bit to clear OUT_PKT_RDY.</td>
<td>0</td>
</tr>
<tr>
<td>SEND_STALL</td>
<td>[5]</td>
<td>R/W</td>
<td>CLEAR MCU should write a &quot;1&quot; to this bit at the same time it clears OUT_PKT_RDY, if it decodes an invalid token. 0 = Finish the STALL condition 1 = The USB issues a STALL and shake to the current control transfer.</td>
<td>0</td>
</tr>
<tr>
<td>SETUP_END</td>
<td>[4]</td>
<td>R</td>
<td>SET Set by the USB when a control transfer ends before DATA_END is set. When the USB sets this bit, an interrupt is generated to the MCU. When such a condition occurs, the USB flushes the FIFO and invalidates MCU access to the FIFO.</td>
<td>0</td>
</tr>
<tr>
<td>DATA_END</td>
<td>[3]</td>
<td>SET</td>
<td>CLEAR Set by the MCU on the conditions below: 1. After loading the last packet of data into the FIFO, at the same time IN_PKT_RDY is set. 2. While it clears OUT_PKT_RDY after unloading the last packet of data. 3. For a zero length data phase.</td>
<td>0</td>
</tr>
<tr>
<td>SENT_STALL</td>
<td>[2]</td>
<td>CLEAR</td>
<td>SET Set by the USB if a control transaction is stopped due to a protocol violation. An interrupt is generated when this bit is set. The MCU should write &quot;0&quot; to clear this bit.</td>
<td>0</td>
</tr>
</tbody>
</table>
### END Point0 Control Status Register (EP0_CSR) (Continued)

<table>
<thead>
<tr>
<th>EP0_CSR</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_PKT_RDY</td>
<td>[1]</td>
<td>SET</td>
<td>CLEAR</td>
<td>Set by the MCU after writing a packet of data into EP0 FIFO. The USB clears this bit once the packet has been successfully sent to the host. An interrupt is generated when the USB clears this bit, so as the MCU to load the next packet. For a zero length data phase, the MCU sets DATA_END at the same time.</td>
<td>0</td>
</tr>
<tr>
<td>OUT_PKT_RDY</td>
<td>[0]</td>
<td>R</td>
<td>SET</td>
<td>Set by the USB once a valid token is written to the FIFO. An interrupt is generated when the USB sets this bit. The MCU clears this bit by writing a &quot;1&quot; to the SERVICED_OUT_PKT_RDY bit.</td>
<td>0</td>
</tr>
</tbody>
</table>
## END POINT IN CONTROL STATUS REGISTER (IN_CSR1_REG/IN_CSR2_REG)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_CSR1_REG</td>
<td>0x52000184(L)</td>
<td>R/W</td>
<td>IN END POINT control status register1</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000187(B)</td>
<td></td>
<td>(byte)</td>
<td></td>
</tr>
</tbody>
</table>

### IN_CSR1_REG

<table>
<thead>
<tr>
<th>Bit</th>
<th>MCU Bit</th>
<th>USB Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved [7]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>CLR_DATA_TOGGE [6]</td>
<td>R/W</td>
<td>R/CLEAR</td>
<td>Used in Set-up procedure. 0: There are alternation of DATA0 and DATA1 1: The data toggle bit is cleared and PID in packet will maintain DATA0</td>
<td>0</td>
</tr>
<tr>
<td>SENT_STALL</td>
<td>[5]</td>
<td>R/CLEAR</td>
<td>SET</td>
<td>0</td>
</tr>
<tr>
<td>SEND_STALL</td>
<td>[4]</td>
<td>W/R</td>
<td>R 0: The MCU clears this bit to finish the STALL condition. 1: The MCU issues a STALL handshake to the USB.</td>
<td>0</td>
</tr>
<tr>
<td>FIFO_FLUSH</td>
<td>[3]</td>
<td>R/W</td>
<td>CLEAR</td>
<td>0</td>
</tr>
<tr>
<td>UNDER_RUN</td>
<td>[2]</td>
<td>R/CLEAR</td>
<td>Valid only For Iso mode. Set by the USB when in ISO mode, an IN token is received and the IN_PKT_RDY bit is not set. The USB sends a zero length data packet for such conditions, and the next packet that is loaded into the FIFO is flushed. This bit is cleared by writing 0.</td>
<td>0</td>
</tr>
<tr>
<td>Reserved [1]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0</td>
</tr>
</tbody>
</table>
### END Point in Control Status Register (IN_CSR1_REG/IN_CSR2_REG) (Continued)

<table>
<thead>
<tr>
<th>IN_CSR1_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_PKT_RDY</td>
<td>[0]</td>
<td>R/SET</td>
<td>CLEAR</td>
<td>Set by the MCU after writing a packet of data into the FIFO. The USB clears this bit once the packet has been successfully sent to the host. An interrupt is generated when the USB clears this bit, so the MCU can load the next packet. While this bit is set, the MCU will not be able to write to the FIFO. If the MCU sets SEND STALL bit, this bit cannot be set.</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_CSR2_REG</td>
<td>0x52000188(L)</td>
<td>R/W</td>
<td>IN END POINT control status register2</td>
<td>0x20</td>
</tr>
<tr>
<td></td>
<td>0x5200018B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN_CSR2_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_SET</td>
<td>[7]</td>
<td>R/W</td>
<td>R</td>
<td>If set, whenever the MCU writes MAXP data, IN_PKT_RDY will automatically be set by the core without any intervention from MCU. If the MCU writes less than MAXP data, IN_PKT_RDY bit has to be set by the MCU.</td>
<td>0</td>
</tr>
<tr>
<td>ISO</td>
<td>[6]</td>
<td>R/W</td>
<td>R</td>
<td>Used only for endpoints whose transfer type is programmable. 1: Reserved 0: Configures endpoint to Bulk mode</td>
<td>0</td>
</tr>
<tr>
<td>MODE_IN</td>
<td>[5]</td>
<td>R/W</td>
<td>R</td>
<td>Used only for endpoints whose direction is programmable. 1: Configures Endpoint Direction as IN 0: Configures Endpoint Direction as OUT</td>
<td>1</td>
</tr>
<tr>
<td>IN_DMA_INT_EN</td>
<td>[4]</td>
<td>R/W</td>
<td>R</td>
<td>Determine whether the interrupt should be issued or not, when the EP1 IN_PKT_RDY condition happens. This is only useful for DMA mode. 0 = Interrupt enable, 1 = Interrupt Disable</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[3:0]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
## END POINT OUT CONTROL STATUS REGISTER (OUT_CSR1_REG/OUT_CSR2_REG)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_CSR1_REG</td>
<td>0x52000190(L) 0x52000193(B)</td>
<td>R/W (byte)</td>
<td>End Point out control status register1</td>
<td>0x00</td>
</tr>
</tbody>
</table>

### OUT_CSR1_REG Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>MCU Description</th>
<th>USB Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR_DATA_TOGGLE [7]</td>
<td>R/W CLEAR</td>
<td>When the MCU writes a 1 to this bit, the data toggle sequence bit is reset to DATA0.</td>
<td>0</td>
</tr>
<tr>
<td>SENT_STALL [6]</td>
<td>R/CLEAR</td>
<td>Set by the USB when an OUT token is ended with a STALL handshake. The USB issues a stall handshake to the host if it sends more than MAXP data for the OUT TOKEN.</td>
<td>0</td>
</tr>
<tr>
<td>SEND_STALL [5]</td>
<td>R/W R</td>
<td>0: The MCU clears this bit to end the STALL condition handshake, IN PKT RDY is cleared. 1: The MCU issues a STALL handshake to the USB. The MCU clears this bit to end the STALL condition handshake, IN PKT RDY is cleared.</td>
<td>0</td>
</tr>
<tr>
<td>FIFO_FLUSH [4]</td>
<td>R/W CLEAR</td>
<td>The MCU writes a 1 to flush the FIFO. This bit can be set only when OUT_PKT_RDY (D0) is set. The packet due to be unloaded by the MCU will be flushed.</td>
<td>0</td>
</tr>
<tr>
<td>DATA_ERROR [3]</td>
<td>R R/W</td>
<td>Valid only in ISO mode. This bit should be sampled with OUT_PKT_RDY. When set, it indicates the data packet due to be unloaded by the MCU has an error (either bit stuffing or CRC). If two packets are loaded into the FIFO, and the second packet has an error, then this bit gets set only after the first packet is unloaded. This bit is automatically cleared when OUT_PKT_RDY gets cleared.</td>
<td>0</td>
</tr>
<tr>
<td>OVER_RUN [2]</td>
<td>R/CLEAR R/W</td>
<td>Valid only in ISO mode. This bit is set if the core is not able to load an OUT ISO token into the FIFO. The MCU clears this bit by writing 0.</td>
<td>0</td>
</tr>
<tr>
<td>Reserved [1]</td>
<td>– – –</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>OUT_PKT_RDY [0]</td>
<td>R/CLEAR SET</td>
<td>Set by the USB after it has loaded a packet of data into the FIFO. Once the MCU reads the packet from FIFO, this bit should be cleared by MCU (write a &quot;0&quot;).</td>
<td>0</td>
</tr>
<tr>
<td>Register</td>
<td>Address</td>
<td>R/W</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>OUT_CSR2_REG</td>
<td>0x52000194(L)</td>
<td>R/W</td>
<td>End Point out control status register2</td>
</tr>
<tr>
<td></td>
<td>0x52000197(B)</td>
<td>(byte)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OUT_CSR2_REG</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_CLR</td>
<td>[7]</td>
<td>R/W</td>
<td>R</td>
<td>If the MCU is set, whenever the MCU reads data from the OUT FIFO, OUT_PKT_RDY will automatically be cleared by the logic without any intervention from the MCU.</td>
<td>0</td>
</tr>
<tr>
<td>ISO</td>
<td>[6]</td>
<td>R/W</td>
<td>R</td>
<td>Determine endpoint transfer type. 0: Configures endpoint to Bulk mode. 1: Reserved</td>
<td>0</td>
</tr>
<tr>
<td>OUT_DMA_INT_MASK</td>
<td>[5]</td>
<td>R/W</td>
<td>R</td>
<td>Determine whether the interrupt should be issued or not. OUT_PKT_RDY condition happens. This is only useful for DMA mode. 0 = Interrupt Enable 1 = Interrupt Disable</td>
<td>0</td>
</tr>
</tbody>
</table>
END POINT FIFO REGISTER (EPn_FIFO_REG)

The EPn_FIFO_REG enables the MCU to access to the EPn FIFO.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP0_FIFO</td>
<td>0x520001C0(L) 0x520001C3 (B)</td>
<td>R/W (byte)</td>
<td>End Point0 FIFO register</td>
<td>0xXX</td>
</tr>
<tr>
<td>EP1_FIFO</td>
<td>0x520001C4(L) 0x520001C7(B)</td>
<td>R/W (byte)</td>
<td>End Point1 FIFO register</td>
<td>0xx</td>
</tr>
<tr>
<td>EP2_FIFO</td>
<td>0x520001C8(L) 0x520001CB(B)</td>
<td>R/W (byte)</td>
<td>End Point2 FIFO register</td>
<td>0xx</td>
</tr>
<tr>
<td>EP3_FIFO</td>
<td>0x520001CC(L) 0x520001CF(B)</td>
<td>R/W (byte)</td>
<td>End Point3 FIFO register</td>
<td>0xx</td>
</tr>
<tr>
<td>EP4_FIFO</td>
<td>0x520001D0(L) 0x520001D3(B)</td>
<td>R/W (byte)</td>
<td>End Point4 FIFO register</td>
<td>0xx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EPn_FIFO</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO_DATA</td>
<td>[7:0]</td>
<td>R/W</td>
<td>R/W</td>
<td>FIFO data value</td>
<td>0xx</td>
</tr>
</tbody>
</table>
### MAX PACKET REGISTER (MAXP_REG)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXP_REG</td>
<td>0x5200018C(L)</td>
<td>R/W</td>
<td>End Point MAX packet register</td>
<td>0x01</td>
</tr>
<tr>
<td></td>
<td>0x5200018F(B)</td>
<td>(byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### MAXP_REG Bit MCU USB Description Initial State

<table>
<thead>
<tr>
<th>MAXP</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3:0]</td>
<td>R/W</td>
<td>R</td>
<td></td>
<td>0000: Reserved 0001: MAXP = 8 Byte 0100: MAXP = 32 Byte 1000: MAXP = 64 Byte</td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For EP0, MAXP=8 is recommended. For EP1~4, MAXP=32 or MAXP=64 is recommended.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>And, if MAXP=32, the dual packet mode will be enabled automatically.</td>
<td></td>
</tr>
</tbody>
</table>
END POINT OUT WRITE COUNT REGISTER (OUT_FIFO_CNT1_REG/OUT_FIFO_CNT2_REG)

These registers maintain the number of bytes in the packet as the number is unloaded by the MCU.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_FIFO_CNT1_REG</td>
<td>0x52000198(L)</td>
<td>R</td>
<td>End Point out write count register1</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200019B(B)</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT_FIFO_CNT1_REG</td>
<td>(byte)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_CNT_LOW</td>
<td>R</td>
<td>W</td>
<td>Lower byte of write count</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_FIFO_CNTL2_REG</td>
<td>0x5200019C(L)</td>
<td>R</td>
<td>End Point out write count register2</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200019F(B)</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(byte)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_CNT_HIGH</td>
<td>R</td>
<td>W</td>
<td>Higher byte of write count. The OUT_CNT_HIGH may be always 0 normally.</td>
<td>0x00</td>
</tr>
</tbody>
</table>
## DMA INTERFACE CONTROL REGISTER (EPn_DMA_CON)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP1_DMA_CON</td>
<td>0x52000200(L)</td>
<td>R/W</td>
<td>EP1 DMA interface control register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000203(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP2_DMA_CON</td>
<td>0x52000218(L)</td>
<td>R/W</td>
<td>EP2 DMA interface control register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200021B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP3_DMA_CON</td>
<td>0x52000240(L)</td>
<td>R/W</td>
<td>EP3 DMA interface control register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000243(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP4_DMA_CON</td>
<td>0x52000258(L)</td>
<td>R/W</td>
<td>EP4 DMA interface control register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200025B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### EPn_DMA_CON Bit MCU USB Description Initial State

#### IN_RUN_OB
- **Bit**: 7
- **MCU**: R/W
- **USB**: R
- **Description**: Read) IN_DMA_Run Observation
  - 0: DMA is stopped
  - 1: DMA is running
- **Initial State**: 0

#### STATE
- **Bit**: 6:4
- **MCU**: R
- **USB**: W
- **Description**: DMA State Monitoring
- **Initial State**: 0

#### DEMAND_MODE
- **Bit**: 3
- **MCU**: R/W
- **USB**: R
- **Description**: DMA Demand mode enable bit
  - 0: Demand mode disable
  - 1: Demand mode enable
- **Initial State**: 0

#### OUT_RUN_OB/ OUT_DMA_RUN
- **Bit**: 2
- **MCU**: R/W
- **USB**: R/W
- **Description**: Functionally separated into write and read operation.
  - Write operation: ‘0’ = Stop ‘1’ = Run
  - Read operation: OUT DMA Run Observation
- **Initial State**: 0

#### IN_DMA_RUN
- **Bit**: 1
- **MCU**: R/W
- **USB**: R
- **Description**: Start DMA operation.
  - 0 = Stop
  - 1 = Run
- **Initial State**: 0

#### DMA_MODE_EN
- **Bit**: 0
- **MCU**: R/CLEAR
- **USB**: R/W
- **Description**: Set DMA mode. If the IN_RUN_OB has been written as 0 and EPn_DMA_TTC_n reaches 0, DMA_MODE_EN bit will be cleared by USB.
  - 0 = Interrupt mode
  - 1 = DMA mode
- **Initial State**: 0
DMA UNIT COUNTER REGISTER (EPN_DMA_UNIT)

This register is valid in Demand mode. In other modes, this register value must be set to ‘0x01’

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP1_DMA_UNIT</td>
<td>0x52000204(L) 0x52000207(B)</td>
<td>R/W</td>
<td>EP1 DMA transfer unit counter base register</td>
<td>0x00</td>
</tr>
<tr>
<td>EP2_DMA_UNIT</td>
<td>0x5200021C(L) 0x5200021F(B)</td>
<td>R/W</td>
<td>EP2 DMA transfer unit counter base register</td>
<td>0x00</td>
</tr>
<tr>
<td>EP3_DMA_UNIT</td>
<td>0x52000244(L) 0x52000247(B)</td>
<td>R/W</td>
<td>EP3 DMA transfer unit counter base register</td>
<td>0x00</td>
</tr>
<tr>
<td>EP4_DMA_UNIT</td>
<td>0x5200025C(L) 0x5200025F(B)</td>
<td>R/W</td>
<td>EP4 DMA transfer unit counter base register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DMA_UNIT</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPn_UNIT_CNT</td>
<td>[7:0]</td>
<td>R/W</td>
<td>R</td>
<td>EP DMA transfer unit counter value</td>
<td>0x00</td>
</tr>
</tbody>
</table>
DMA FIFO COUNTER REGISTER (EP_{n}_DMA_FIFO)

This register has values in byte size in FIFO to be transferred by DMA. In case of OUT_{DMA_RUN} enabled, the value in OUT FIFO Write Count Register1 will be loaded in this register automatically. In case of IN DMA mode, the MCU should set proper value by software.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP1_DMA_FIFO</td>
<td>0x52000208(L) 0x5200020B(B)</td>
<td>R/W (byte)</td>
<td>EP1 DMA transfer FIFO counter base register</td>
<td>0x00</td>
</tr>
<tr>
<td>EP2_DMA_FIFO</td>
<td>0x52000220(L) 0x52000223(B)</td>
<td>R/W (byte)</td>
<td>EP2 DMA transfer FIFO counter base register</td>
<td>0x00</td>
</tr>
<tr>
<td>EP3_DMA_FIFO</td>
<td>0x52000248(L) 0x5200024B(B)</td>
<td>R/W (byte)</td>
<td>EP3 DMA transfer FIFO counter base register</td>
<td>0x00</td>
</tr>
<tr>
<td>EP4_DMA_FIFO</td>
<td>0x52000260(L) 0x52000263(B)</td>
<td>R/W (byte)</td>
<td>EP4 DMA transfer FIFO counter base register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DMA_FIFO</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPn_FIFO_CNT</td>
<td>[7:0]</td>
<td>R/W</td>
<td>R</td>
<td>EP DMA transfer FIFO counter value</td>
<td>0x00</td>
</tr>
</tbody>
</table>
DMA TOTAL TRANSFER COUNTER REGISTER (EPN_DMA_TTC_L,M,H)

This register should have total number of bytes to be transferred using DMA (total 20-bit counter).

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP1_DMA_TTC_L</td>
<td>0x5200020C(L)</td>
<td>R/W</td>
<td>EP1 DMA total transfer counter(lower byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200020F(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP1_DMA_TTC_M</td>
<td>0x52000210(L)</td>
<td>R/W</td>
<td>EP1 DMA total transfer counter(middle byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000213(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP1_DMA_TTC_H</td>
<td>0x52000214(L)</td>
<td>R/W</td>
<td>EP1 DMA total transfer counter(higher byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000217(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP2_DMA_TTC_L</td>
<td>0x52000224(L)</td>
<td>R/W</td>
<td>EP2 DMA total transfer counter(lower byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000227(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP2_DMA_TTC_M</td>
<td>0x52000228(L)</td>
<td>R/W</td>
<td>EP2 DMA total transfer counter(middle byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200022B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP2_DMA_TTC_H</td>
<td>0x5200022C(L)</td>
<td>R/W</td>
<td>EP2 DMA total transfer counter(higher byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200022F(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP3_DMA_TTC_L</td>
<td>0x5200024C(L)</td>
<td>R/W</td>
<td>EP3 DMA total transfer counter(lower byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200024F(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP3_DMA_TTC_M</td>
<td>0x52000250(L)</td>
<td>R/W</td>
<td>EP3 DMA total transfer counter(middle byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000253(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP3_DMA_TTC_H</td>
<td>0x52000254(L)</td>
<td>R/W</td>
<td>EP3 DMA total transfer counter(higher byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000257(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP4_DMA_TTC_L</td>
<td>0x52000264(L)</td>
<td>R/W</td>
<td>EP4 DMA total transfer counter(lower byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x52000267(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP4_DMA_TTC_M</td>
<td>0x52000268(L)</td>
<td>R/W</td>
<td>EP4 DMA total transfer counter(middle byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200026B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EP4_DMA_TTC_H</td>
<td>0x5200026C(L)</td>
<td>R/W</td>
<td>EP4 DMA total transfer counter(higher byte)</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5200026F(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DMA_TX</th>
<th>Bit</th>
<th>MCU</th>
<th>USB</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPn_TTC_L</td>
<td>[7:0]</td>
<td>R/W</td>
<td>R</td>
<td>DMA total transfer count value (lower byte)</td>
<td>0x00</td>
</tr>
<tr>
<td>EPn_TTC_M</td>
<td>[7:0]</td>
<td>R/W</td>
<td>R</td>
<td>DMA total transfer count value (middle byte)</td>
<td>0x00</td>
</tr>
<tr>
<td>EPn_TTC_H</td>
<td>[3:0]</td>
<td>R/W</td>
<td>R</td>
<td>DMA total transfer count value (higher byte)</td>
<td>0x00</td>
</tr>
</tbody>
</table>
14  INTERRUPT CONTROLLER

OVERVIEW

The interrupt controller in the S3C2410X receives the request from 56 interrupt sources. These interrupt sources are provided by internal peripherals such as the DMA controller, the UART, IIC, and others. In these interrupt sources, the UARTn and EINTn interrupts are 'OR'ed to the interrupt controller.

When receiving multiple interrupt requests from internal peripherals and external interrupt request pins, the interrupt controller requests FIQ or IRQ interrupt of the ARM920T core after the arbitration procedure.

The arbitration procedure depends on the hardware priority logic and the result is written to the interrupt pending register, which helps users notify which interrupt is generated out of various interrupt sources.

Figure 14-1. Interrupt Process Diagram
INTERRUPT CONTROLLER OPERATION

F-bit and I-bit of Program Status Register (PSR)

If the F-bit of PSR in ARM920T CPU is set to 1, the CPU does not accept the Fast Interrupt Request (FIQ) from the interrupt controller. Likewise, if I-bit of the PSR is set to 1, the CPU does not accept the Interrupt Request (IRQ) from the interrupt controller. So, the interrupt controller can receive interrupts by clearing F-bit or I-bit of the PSR to 0 and setting the corresponding bit of INTMSK to 0.

Interrupt Mode

The ARM920T has two types of Interrupt mode: FIQ or IRQ. All the interrupt sources determine which mode is used at interrupt request.

Interrupt Pending Register

The S3C2410X has two interrupt pending registers: source pending register (SRCPND) and interrupt pending register (INTPND). These pending registers indicate whether or not an interrupt request is pending. When the interrupt sources request interrupt service, the corresponding bits of SRCPND register are set to 1, and at the same time, only one bit of the INTPND register is set to 1 automatically after arbitration procedure. If interrupts are masked, the corresponding bits of the SRCPND register are set to 1. This does not cause the bit of INTPND register changed. When a pending bit of the INTPND register is set, the interrupt service routine starts whenever the I-flag or F-flag is cleared to 0. The SRCPND and INTPND registers can be read and written, so the service routine must clear the pending condition by writing a 1 to the corresponding bit in the SRCPND register first and then clear the pending condition in the INTPND registers by using the same method.

Interrupt Mask Register

This register indicates that an interrupt has been disabled if the corresponding mask bit is set to 1. If an interrupt mask bit of INTMSK is 0, the interrupt will be serviced normally. If the corresponding mask bit is 1 and the interrupt is generated, the source pending bit will be set.
## INTERRUPT SOURCES

The interrupt controller supports 56 interrupt sources as shown in the table below.

<table>
<thead>
<tr>
<th>Sources</th>
<th>Descriptions</th>
<th>Arbiter Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_ADC</td>
<td>ADC EOC and Touch interrupt (INT_ADC/INT_TC)</td>
<td>ARB5</td>
</tr>
<tr>
<td>INT_RTC</td>
<td>RTC alarm interrupt</td>
<td>ARB5</td>
</tr>
<tr>
<td>INT_SPI1</td>
<td>SPI1 interrupt</td>
<td>ARB5</td>
</tr>
<tr>
<td>INT_UART0</td>
<td>UART0 Interrupt (ERR, RXD, and TXD)</td>
<td>ARB5</td>
</tr>
<tr>
<td>INT_IIC</td>
<td>IIC interrupt</td>
<td>ARB4</td>
</tr>
<tr>
<td>INT_USBH</td>
<td>USB Host interrupt</td>
<td>ARB4</td>
</tr>
<tr>
<td>INT_USBD</td>
<td>USB Device interrupt</td>
<td>ARB4</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>ARB4</td>
</tr>
<tr>
<td>INT_UART1</td>
<td>UART1 Interrupt (ERR, RXD, and TXD)</td>
<td>ARB4</td>
</tr>
<tr>
<td>INT_SPI0</td>
<td>SPI0 interrupt</td>
<td>ARB4</td>
</tr>
<tr>
<td>INT_SDI</td>
<td>SDI interrupt</td>
<td>ARB 3</td>
</tr>
<tr>
<td>INT_DMA3</td>
<td>DMA channel 3 interrupt</td>
<td>ARB3</td>
</tr>
<tr>
<td>INT_DMA2</td>
<td>DMA channel 2 interrupt</td>
<td>ARB3</td>
</tr>
<tr>
<td>INT_DMA1</td>
<td>DMA channel 1 interrupt</td>
<td>ARB3</td>
</tr>
<tr>
<td>INT_DMA0</td>
<td>DMA channel 0 interrupt</td>
<td>ARB3</td>
</tr>
<tr>
<td>INT_LCD</td>
<td>LCD interrupt (INT_FrSyn and INT_FiCnt)</td>
<td>ARB3</td>
</tr>
<tr>
<td>INT_UART2</td>
<td>UART2 Interrupt (ERR, RXD, and TXD)</td>
<td>ARB2</td>
</tr>
<tr>
<td>INT_TIMER4</td>
<td>Timer4 interrupt</td>
<td>ARB2</td>
</tr>
<tr>
<td>INT_TIMER3</td>
<td>Timer3 interrupt</td>
<td>ARB2</td>
</tr>
<tr>
<td>INT_TIMER2</td>
<td>Timer2 interrupt</td>
<td>ARB2</td>
</tr>
<tr>
<td>INT_TIMER1</td>
<td>Timer1 interrupt</td>
<td>ARB 2</td>
</tr>
<tr>
<td>INT_TIMER0</td>
<td>Timer0 interrupt</td>
<td>ARB2</td>
</tr>
<tr>
<td>INT_WDT</td>
<td>Watch-Dog timer interrupt</td>
<td>ARB1</td>
</tr>
<tr>
<td>INT_TICK</td>
<td>RTC Time tick interrupt</td>
<td>ARB1</td>
</tr>
<tr>
<td>nBATT_FLT</td>
<td>Battery Fault interrupt</td>
<td>ARB1</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>ARB1</td>
</tr>
<tr>
<td>EINT8_23</td>
<td>External interrupt 8 – 23</td>
<td>ARB1</td>
</tr>
<tr>
<td>EINT4_7</td>
<td>External interrupt 4 – 7</td>
<td>ARB1</td>
</tr>
<tr>
<td>EINT3</td>
<td>External interrupt 3</td>
<td>ARB0</td>
</tr>
<tr>
<td>EINT2</td>
<td>External interrupt 2</td>
<td>ARB0</td>
</tr>
<tr>
<td>EINT1</td>
<td>External interrupt 1</td>
<td>ARB0</td>
</tr>
<tr>
<td>EINT0</td>
<td>External interrupt 0</td>
<td>ARB0</td>
</tr>
</tbody>
</table>
INTERRUPT PRIORITY GENERATING BLOCK

The priority logic for 32 interrupt requests is composed of seven rotation based arbiters: six first-level arbiters and one second-level arbiter as shown in Figure 14-2 below.

Figure 14-2. Priority Generating Block
INTERRUPT PRIORITY

Each arbiter can handle six interrupt requests based on the one bit arbiter mode control (ARB_MODE) and two bits of selection control signals (ARB_SEL) as follows:

- If ARB_SEL bits are 00b, the priority order is REQ0, REQ1, REQ2, REQ3, REQ4, and REQ5.
- If ARB_SEL bits are 01b, the priority order is REQ0, REQ2, REQ3, REQ4, REQ1, and REQ5.
- If ARB_SEL bits are 10b, the priority order is REQ0, REQ3, REQ4, REQ1, REQ2, and REQ5.
- If ARB_SEL bits are 11b, the priority order is REQ0, REQ4, REQ1, REQ2, REQ3, and REQ5.

Note that REQ0 of an arbiter always has the highest priority, and REQ5 has the lowest one. In addition, by changing the ARB_SEL bits, we can rotate the priority of REQ1 to REQ4.

Here, if ARB_MODE bit is set to 0, ARB_SEL bits are not automatically changed, making the arbiter to operate in the fixed priority mode (note that even in this mode, we can reconfigure the priority by manually changing the ARB_SEL bits). On the other hand, if ARB_MODE bit is 1, ARB_SEL bits are changed in rotation fashion, e.g., if REQ1 is serviced, ARB_SEL bits are changed to 01b automatically so as to put REQ1 into the lowest priority. The detailed rules of ARB_SEL change are as follows:

- If REQ0 or REQ5 is serviced, ARB_SEL bits are not changed at all.
- If REQ1 is serviced, ARB_SEL bits are changed to 01b.
- If REQ2 is serviced, ARB_SEL bits are changed to 10b.
- If REQ3 is serviced, ARB_SEL bits are changed to 11b.
- If REQ4 is serviced, ARB_SEL bits are changed to 00b.
INTERRUPT CONTROLLER SPECIAL REGISTERS

There are five control registers in the interrupt controller: source pending register, interrupt mode register, mask register, priority register, and interrupt pending register.

All the interrupt requests from the interrupt sources are first registered in the source pending register. They are divided into two groups including Fast Interrupt Request (FIQ) and Interrupt Request (IRQ), based on the interrupt mode register. The arbitration procedure for multiple IRQs is based on the priority register.

SOURCE PENDING (SRCPND) REGISTER

The SRCPND register is composed of 32 bits each of which is related to an interrupt source. Each bit is set to 1 if the corresponding interrupt source generates the interrupt request and waits for the interrupt to be serviced. Accordingly, this register indicates which interrupt source is waiting for the request to be serviced. Note that each bit of the SRCPND register is automatically set by the interrupt sources regardless of the masking bits in the INTMASK register. In addition, the SRCPND register is not affected by the priority logic of interrupt controller.

In the interrupt service routine for a specific interrupt source, the corresponding bit of the SRCPND register has to be cleared to get the interrupt request from the same source correctly. If you return from the ISR without clearing the bit, the interrupt controller operates as if another interrupt request came in from the same source. In other words, if a specific bit of the SRCPND register is set to 1, it is always considered as a valid interrupt request waiting to be serviced.

The time to clear the corresponding bit depends on the user's requirement. If you want to receive another valid request from the same source, you should clear the corresponding bit first, and then enable the interrupt.

You can clear a specific bit of the SRCPND register by writing a data to this register. It clears only the bit positions of the SRCPND corresponding to those set to one in the data. The bit positions corresponding to those that are set to 0 in the data remains as they are.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRCPND</td>
<td>0X4A000000</td>
<td>R/W</td>
<td>Indicate the interrupt request status. 0 = The interrupt has not been requested. 1 = The interrupt source has asserted the interrupt request.</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
### SOURCE PENDING (SRCPND) REGISTER (Continued)

<table>
<thead>
<tr>
<th>SRCPND</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_ADC</td>
<td>[31]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_RTC</td>
<td>[30]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_SPI1</td>
<td>[29]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART0</td>
<td>[28]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_IIC</td>
<td>[27]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_USBH</td>
<td>[26]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_USBD</td>
<td>[25]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[24]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART1</td>
<td>[23]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_SPI0</td>
<td>[22]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_SD1</td>
<td>[21]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA3</td>
<td>[20]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA2</td>
<td>[19]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA1</td>
<td>[18]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA0</td>
<td>[17]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_LCD</td>
<td>[16]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART2</td>
<td>[15]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER4</td>
<td>[14]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER3</td>
<td>[13]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER2</td>
<td>[12]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER1</td>
<td>[11]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER0</td>
<td>[10]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_WDT</td>
<td>[9]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TICK</td>
<td>[8]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>nBATT_FLT</td>
<td>[7]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[6]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>EINT8_23</td>
<td>[5]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT4_7</td>
<td>[4]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT3</td>
<td>[3]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT2</td>
<td>[2]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT1</td>
<td>[1]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT0</td>
<td>[0]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
</tbody>
</table>
INTERRUPT MODE (INTMOD) REGISTER

This register is composed of 32 bits each of which is related to an interrupt source. If a specific bit is set to 1, the corresponding interrupt is processed in the FIQ (fast interrupt) mode. Otherwise, it is processed in the IRQ mode (normal interrupt).

Note that only one interrupt source can be serviced in the FIQ mode in the interrupt controller (you should use the FIQ mode only for the urgent interrupt). Thus, only one bit of INTMOD can be set to 1.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTMOD</td>
<td>0x4A000004</td>
<td>R/W</td>
<td>Interrupt mode register.</td>
<td>0x00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = IRQ mode               1 = FIQ mode</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** If an interrupt mode is set to FIQ mode in the INTMOD register, FIQ interrupt will not affect both INTPND and INTOFFSET registers. In this case, the two registers are valid only for IRQ mode interrupt source.
<table>
<thead>
<tr>
<th>INTMOD</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_ADC</td>
<td>[31]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_RTC</td>
<td>[30]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_SPI1</td>
<td>[29]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART0</td>
<td>[28]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_IIC</td>
<td>[27]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_USBH</td>
<td>[26]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_USBD</td>
<td>[25]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[24]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>INT_URRT1</td>
<td>[23]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_SPI0</td>
<td>[22]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_SD1</td>
<td>[21]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA3</td>
<td>[20]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA2</td>
<td>[19]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA1</td>
<td>[18]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA0</td>
<td>[17]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_LCD</td>
<td>[16]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART2</td>
<td>[15]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER4</td>
<td>[14]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER3</td>
<td>[13]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER2</td>
<td>[12]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER1</td>
<td>[11]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER0</td>
<td>[10]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_WDT</td>
<td>[9]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>INT_TICK</td>
<td>[8]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>nBATT_FLT</td>
<td>[7]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[6]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>EINT8_23</td>
<td>[5]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>EINT4_7</td>
<td>[4]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>EINT3</td>
<td>[3]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>EINT2</td>
<td>[2]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>EINT1</td>
<td>[1]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
<tr>
<td>EINT0</td>
<td>[0]</td>
<td>0 = IRQ, 1 = FIQ</td>
<td>0</td>
</tr>
</tbody>
</table>
INTERRUPT MASK (INTMSK) REGISTER

This register also has 32 bits each of which is related to an interrupt source. If a specific bit is set to 1, the CPU does not service the interrupt request from the corresponding interrupt source (note that even in such a case, the corresponding bit of SSCPND register is set to 1). If the mask bit is 0, the interrupt request can be serviced.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTMSK</td>
<td>0x4A000008</td>
<td>R/W</td>
<td>Determine which interrupt source is masked. The masked interrupt source will not be serviced. 0 = Interrupt service is available. 1 = Interrupt service is masked.</td>
<td>0xFFFFFFFF</td>
</tr>
<tr>
<td>INTMSK</td>
<td>Bit</td>
<td>Description</td>
<td>Initial State</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>------------------------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>INT_ADC</td>
<td>[31]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_RTC</td>
<td>[30]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_SPI1</td>
<td>[29]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_UART0</td>
<td>[28]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_IIC</td>
<td>[27]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_USBH</td>
<td>[26]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_USBD</td>
<td>[25]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>[24]</td>
<td>Not used</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_UART1</td>
<td>[23]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_SPI0</td>
<td>[22]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_SD1</td>
<td>[21]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_DMA3</td>
<td>[20]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_DMA2</td>
<td>[19]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_DMA1</td>
<td>[18]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_DMA0</td>
<td>[17]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_LCD</td>
<td>[16]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_UART2</td>
<td>[15]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_TIMER4</td>
<td>[14]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_TIMER3</td>
<td>[13]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_TIMER2</td>
<td>[12]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_TIMER1</td>
<td>[11]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_TIMER0</td>
<td>[10]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_WDT</td>
<td>[9]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INT_TICK</td>
<td>[8]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>nBATT_FLT</td>
<td>[7]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>[6]</td>
<td>Not used</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EINT8_23</td>
<td>[5]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EINT4_7</td>
<td>[4]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EINT3</td>
<td>[3]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EINT2</td>
<td>[2]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EINT1</td>
<td>[1]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EINT0</td>
<td>[0]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
## Interrupt Controller S3C2410X

### Priority Register (PRIORITY)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY</td>
<td>0x4A00000C</td>
<td>R/W</td>
<td>IRQ priority control register</td>
<td>0x7F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRIORITY</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_SEL6</td>
<td>[20:19]</td>
<td>Arbiter 6 group priority order set</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = REQ 0-1-2-3-4-5 01 = REQ 0-2-3-4-1-5 10 = REQ 0-3-4-1-2-5 11 = REQ 0-4-1-2-3-5</td>
<td></td>
</tr>
<tr>
<td>ARB_SEL5</td>
<td>[18:17]</td>
<td>Arbiter 5 group priority order set</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = REQ 1-2-3-4 01 = REQ 2-3-4-1 10 = REQ 3-4-1-2 11 = REQ 4-1-2-3</td>
<td></td>
</tr>
<tr>
<td>ARB_SEL4</td>
<td>[16:15]</td>
<td>Arbiter 4 group priority order set</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = REQ 0-1-2-3-4-5 01 = REQ 0-2-3-4-1-5 10 = REQ 0-3-4-1-2-5 11 = REQ 0-4-1-2-3-5</td>
<td></td>
</tr>
<tr>
<td>ARB_SEL3</td>
<td>[14:13]</td>
<td>Arbiter 3 group priority order set</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = REQ 0-1-2-3-4-5 01 = REQ 0-2-3-4-1-5 10 = REQ 0-3-4-1-2-5 11 = REQ 0-4-1-2-3-5</td>
<td></td>
</tr>
<tr>
<td>ARB_SEL2</td>
<td>[12:11]</td>
<td>Arbiter 2 group priority order set</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = REQ 0-1-2-3-4-5 01 = REQ 0-2-3-4-1-5 10 = REQ 0-3-4-1-2-5 11 = REQ 0-4-1-2-3-5</td>
<td></td>
</tr>
<tr>
<td>ARB_SEL1</td>
<td>[10:9]</td>
<td>Arbiter 1 group priority order set</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = REQ 0-1-2-3-4-5 01 = REQ 0-2-3-4-1-5 10 = REQ 0-3-4-1-2-5 11 = REQ 0-4-1-2-3-5</td>
<td></td>
</tr>
<tr>
<td>ARB_SEL0</td>
<td>[8:7]</td>
<td>Arbiter 0 group priority order set</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = REQ 1-2-3-4 01 = REQ 2-3-4-1 10 = REQ 3-4-1-2 11 = REQ 4-1-2-3</td>
<td></td>
</tr>
<tr>
<td>ARB_MODE6</td>
<td>[6]</td>
<td>Arbiter 6 group priority rotate enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Priority does not rotate, 1 = Priority rotate enable</td>
<td></td>
</tr>
<tr>
<td>ARB_MODE5</td>
<td>[5]</td>
<td>Arbiter 5 group priority rotate enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Priority does not rotate, 1 = Priority rotate enable</td>
<td></td>
</tr>
<tr>
<td>ARB_MODE4</td>
<td>[4]</td>
<td>Arbiter 4 group priority rotate enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Priority does not rotate, 1 = Priority rotate enable</td>
<td></td>
</tr>
<tr>
<td>ARB_MODE3</td>
<td>[3]</td>
<td>Arbiter 3 group priority rotate enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Priority does not rotate, 1 = Priority rotate enable</td>
<td></td>
</tr>
<tr>
<td>ARB_MODE2</td>
<td>[2]</td>
<td>Arbiter 2 group priority rotate enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Priority does not rotate, 1 = Priority rotate enable</td>
<td></td>
</tr>
</tbody>
</table>
### Priority Register (PRIORITY) (Continued)

<table>
<thead>
<tr>
<th>PRIORITY</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_MODE1</td>
<td>[1]</td>
<td>Arbiter 1 group priority rotate enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Priority does not rotate,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Priority rotate enable</td>
<td></td>
</tr>
<tr>
<td>ARB_MODE0</td>
<td>[0]</td>
<td>Arbiter 0 group priority rotate enable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Priority does not rotate,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Priority rotate enable</td>
<td></td>
</tr>
</tbody>
</table>
INTERRUPT PENDING (INTPND) REGISTER

Each of the 32 bits in the interrupt pending register shows whether the corresponding interrupt request, which is unmasked and waits for the interrupt to be serviced, has the highest priority. Since the INTPND register is located after the priority logic, only one bit can be set to 1, and that interrupt request generates IRQ to CPU. In interrupt service routine for IRQ, you can read this register to determine which interrupt source is serviced among the 32 sources.

Like the SRCPND register, this register has to be cleared in the interrupt service routine after clearing the SRCPND register. We can clear a specific bit of the INTPND register by writing a data to this register. It clears only the bit positions of the INTPND register corresponding to those set to one in the data. The bit positions corresponding to those that are set to 0 in the data remains as they are.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTPND</td>
<td>0x4A000010</td>
<td>R/W</td>
<td>Indicate the interrupt request status.</td>
<td>0x00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = The interrupt has not been requested.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = The interrupt source has asserted the interrupt request.</td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
1. If the FIQ mode interrupt occurs, the corresponding bit of INTPND will not be turned on as the INTPND register is available only for IRQ mode interrupt.
2. Cautions in clearing the INTPND register.
   The INTPND register is cleared to ‘0’ by writing ‘1’. If the INTPND bit, which has ‘1’, is cleared by ‘0’, the INTPND register & INTOFFSET register may have unexpected value in some case.
   So, you never write ‘0’ on the INTPND bit having ‘1’. The convenient method to clear the INTPND register is writing the INTPND register value on the INTPND register. (In even our example code, this guide hasn't been applied yet.)
### INT_PND Bit Description Initial State

<table>
<thead>
<tr>
<th>INTPND</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_ADC</td>
<td>[31]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_RTC</td>
<td>[30]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_SPI1</td>
<td>[29]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART0</td>
<td>[28]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_IIC</td>
<td>[27]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_USBH</td>
<td>[26]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_USBD</td>
<td>[25]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[24]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART1</td>
<td>[23]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_SPI0</td>
<td>[22]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_SDI</td>
<td>[21]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA3</td>
<td>[20]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA2</td>
<td>[19]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA1</td>
<td>[18]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_DMA0</td>
<td>[17]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_LCD</td>
<td>[16]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_UART2</td>
<td>[15]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER4</td>
<td>[14]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER3</td>
<td>[13]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER2</td>
<td>[12]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER1</td>
<td>[11]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TIMER0</td>
<td>[10]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_WDT</td>
<td>[9]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TICK</td>
<td>[8]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>nBATT_FLT</td>
<td>[7]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[6]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>EINT8_23</td>
<td>[5]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT4_7</td>
<td>[4]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT3</td>
<td>[3]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT2</td>
<td>[2]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT1</td>
<td>[1]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>EINT0</td>
<td>[0]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
</tbody>
</table>
INTERRUPT OFFSET (INTOFFSET) REGISTER

The value in the interrupt offset register shows which interrupt request of IRQ mode is in the INTPND register. This bit can be cleared automatically by clearing SRCPND and INTPND.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTOFFSET</td>
<td>0X4A000014</td>
<td>R</td>
<td>Indicate the IRQ interrupt request source</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INT Source</th>
<th>The OFFSET Value</th>
<th>INT Source</th>
<th>The OFFSET Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_ADC</td>
<td>31</td>
<td>INT_UART2</td>
<td>15</td>
</tr>
<tr>
<td>INT_RTC</td>
<td>30</td>
<td>INT_TIMER4</td>
<td>14</td>
</tr>
<tr>
<td>INT_SPI1</td>
<td>29</td>
<td>INT_TIMER3</td>
<td>13</td>
</tr>
<tr>
<td>INT_UART0</td>
<td>28</td>
<td>INT_TIMER2</td>
<td>12</td>
</tr>
<tr>
<td>INT_IIC</td>
<td>27</td>
<td>INT_TIMER1</td>
<td>11</td>
</tr>
<tr>
<td>INT_USBH</td>
<td>26</td>
<td>INT_TIMER0</td>
<td>10</td>
</tr>
<tr>
<td>INT_USB0</td>
<td>25</td>
<td>INT_WDT</td>
<td>9</td>
</tr>
<tr>
<td>Reserved</td>
<td>24</td>
<td>INT_TICK</td>
<td>8</td>
</tr>
<tr>
<td>INT_UART1</td>
<td>23</td>
<td>nBATT_FLT</td>
<td>7</td>
</tr>
<tr>
<td>INT_SPI0</td>
<td>22</td>
<td>Reserved</td>
<td>6</td>
</tr>
<tr>
<td>INT_SD1</td>
<td>21</td>
<td>EINT8_23</td>
<td>5</td>
</tr>
<tr>
<td>INT_DMA3</td>
<td>20</td>
<td>EINT4_7</td>
<td>4</td>
</tr>
<tr>
<td>INT_DMA2</td>
<td>19</td>
<td>EINT3</td>
<td>3</td>
</tr>
<tr>
<td>INT_DMA1</td>
<td>18</td>
<td>EINT2</td>
<td>2</td>
</tr>
<tr>
<td>INT_DMA0</td>
<td>17</td>
<td>EINT1</td>
<td>1</td>
</tr>
<tr>
<td>INT_LCD</td>
<td>16</td>
<td>EINT0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Note:** FIQ mode interrupt does not affect the INTOFFSET register as the register is available only for IRQ mode interrupt.
SUB SOURCE PENDING (SUBSRCPND) REGISTER

You can clear a specific bit of the SUBSRCPND register by writing a data to this register. It clears only the bit positions of the SUBSRCPND register corresponding to those set to one in the data. The bit positions corresponding to those that are set to 0 in the data remains as they are.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSRCPND</td>
<td>0x4A000018</td>
<td>R/W</td>
<td>Indicate the interrupt request status. 0 = The interrupt has not been requested. 1 = The interrupt source has asserted the interrupt request.</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUBSRCPND</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[31:11]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>INT_ADC</td>
<td>[10]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TC</td>
<td>[9]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_ERR2</td>
<td>[8]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TXD2</td>
<td>[7]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_RXD2</td>
<td>[6]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_ERR1</td>
<td>[5]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TXD1</td>
<td>[4]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_RXD1</td>
<td>[3]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_ERR0</td>
<td>[2]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_TXD0</td>
<td>[1]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
<tr>
<td>INT_RXD0</td>
<td>[0]</td>
<td>0 = Not requested, 1 = Requested</td>
<td>0</td>
</tr>
</tbody>
</table>
**INTERRUPT SUB MASK (INTSUBMSK) REGISTER**

This register has 11 bits each of which is related to an interrupt source. If a specific bit is set to 1, the interrupt request from the corresponding interrupt source is not serviced by the CPU (note that even in such a case, the corresponding bit of the SUBSRCMPND register is set to 1). If the mask bit is 0, the interrupt request can be serviced.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTSUBMSK</td>
<td>0x4A00001C</td>
<td>R/W</td>
<td>Determine which interrupt source is masked. The masked interrupt source will not be serviced. 0 = Interrupt service is available, 1 = Interrupt service is masked.</td>
<td>0x7FF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INTSUBMSK</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[31:11]</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>INT_ADC</td>
<td>[10]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_TC</td>
<td>[9]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_ERR2</td>
<td>[8]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_TXD2</td>
<td>[7]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_RXD2</td>
<td>[6]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_ERR1</td>
<td>[5]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_TXD1</td>
<td>[4]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_RXD1</td>
<td>[3]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_ERR0</td>
<td>[2]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_TXD0</td>
<td>[1]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
<tr>
<td>INT_RXD0</td>
<td>[0]</td>
<td>0 = Service available, 1 = Masked</td>
<td>1</td>
</tr>
</tbody>
</table>
OVERVIEW

The LCD controller in the S3C2410X consists of the logic for transferring LCD image data from a video buffer located in system memory to an external LCD driver.

The LCD controller supports monochrome, 2-bit per pixel (4-level gray scale) or 4-bit per pixel (16-level gray scale) mode on a monochrome LCD, using a time-based dithering algorithm and Frame Rate Control (FRC) method and it can be interfaced with a color LCD panel at 8-bit per pixel (256-level color) and 12-bit per pixel (4096-level color) for interfacing with STN LCD. It can support 1-bit per pixel, 2-bit per pixel, 4-bit per pixel, and 8-bit per pixel for interfacing with the palettized TFT color LCD panel, and 16-bit per pixel and 24-bit per pixel for non-palettized true-color display.

The LCD controller can be programmed to support different requirements on the screen related to the number of horizontal and vertical pixels, data line width for the data interface, interface timing, and refresh rate.

FEATURES

STN LCD displays:
— Supports 3 types of LCD panels: 4-bit dual scan, 4-bit single scan, and 8-bit single scan display type
— Supports the monochrome, 4 gray levels, and 16 gray levels
— Supports 256 colors and 4096 colors for color STN LCD panel
— Supports multiple screen size
  Typical actual screen size: 640×480, 320×240, 160×160, and others
  Maximum virtual screen size is 4Mbytes.
  Maximum virtual screen size in 256 color mode: 4096×1024, 2048×2048, 1024×4096, and others

TFT LCD displays:
— Supports 1, 2, 4 or 8-bpp (bit per pixel) palettized color displays for TFT
— Supports 16-bpp non-palettized true-color displays for color TFT
— Supports 24-bpp non-palettized true-color displays for color TFT
— Supports maximum 16M color TFT at 24bit per pixel mode
— Supports multiple screen size
  Typical actual screen size: 640×480, 320×240, 160×160, and others
  Maximum virtual screen size is 4Mbytes.
  Maximum virtual screen size in 64K color mode: 2048×1024 and others
COMMON FEATURES

The LCD controller has a dedicated DMA that supports to fetch the image data from video buffer located in system memory. Its features also include:

- Dedicated interrupt functions (INT_FrSyn and INT_FiCnt)
- The system memory is used as the display memory.
- Supports Multiple Virtual Display Screen (Supports Hardware Horizontal/Vertical Scrolling)
- Programmable timing control for different display panels
- Supports little and big-endian byte ordering, as well as WinCE data formats
- Supports SEC TFT LCD panel (SAMSUNG 3.5” Portrait/256K Color/Reflective a-Si TFT LCD)
  LTS350Q1-PD1: TFT LCD panel with touch panel and front light unit
  LTS350Q1-PD2: TFT LCD panel only

NOTE

WinCE doesn't support the 12-bit packed data format. Please check if WinCE can support the 12-bit color-mode.

EXTERNAL INTERFACE SIGNAL

VFRAME/VSYNC/STV : Frame synchronous signal (STN)/vertical synchronous signal (TFT)/SEC TFT signal
VLINE/HSYNC/CPV : Line synchronous pulse signal (STN)/horizontal sync signal (TFT)/SEC TFT signal
VCLK/LCD_HCLK : Pixel clock signal (STN/TFT)/SEC TFT signal
VD[23:0] : LCD pixel data output ports (STN/TFT/SEC TFT)
VM/VDEN/TP : AC bias signal for the LCD driver (STN)/data enable signal (TFT)/SEC TFT signal
LEND/STH : Line end signal (TFT)/SEC TFT signal
LCD_PWREN : LCD panel power enable control signal
LCDVF0 : SEC TFT Signal OE
LCDVF1 : SEC TFT Signal REV
LCDVF2 : SEC TFT Signal REV B

The 33 output ports in total includes 24 data bits and 9 control bits
The S3C2410X LCD controller is used to transfer the video data and to generate the necessary control signals, such as VFRAME, VLINE, VCLK, VM, and so on. In addition to the control signals, the S3C2410X has the data ports for video data, which are VD[23:0] as shown in Figure 15-1. The LCD controller consists of a REGBANK, LCDCDMA, VIDPRCS, TIMEGEN, and LPC3600 (See the Figure 15-1 LCD Controller Block Diagram). The REGBANK has 17 programmable register sets and 256x16 palette memory which are used to configure the LCD controller. The LCDCDMA is a dedicated DMA, which can transfer the video data in frame memory to LCD driver automatically. By using this special DMA, the video data can be displayed on the screen without CPU intervention. The VIDPRCS receives the video data from the LCDCDMA and sends the video data through the VD[23:0] data ports to the LCD driver after changing them into a suitable data format, for example 4/8-bit single scan or 4-bit dual scan display mode. The TIMEGEN consists of programmable logic to support the variable requirements of interface timing and rates commonly found in different LCD drivers. The TIMEGEN block generates VFRAME, VLINE, VCLK, VM, and so on.

The description of data flow is as follows:

FIFO memory is present in the LCDCDMA. When FIFO is empty or partially empty, the LCDCDMA requests data fetching from the frame memory based on the burst memory transfer mode (consecutive memory fetching of 4 words (16 bytes) per one burst request without allowing the bus mastership to another bus master during the bus transfer). When the transfer request is accepted by bus arbitrator in the memory controller, there will be four successive word data transfers from system memory to internal FIFO. The total size of FIFO is 28 words, which consists of 12 words FIFOL and 16 words FIFOH, respectively. The S3C2410X has two FIFOs to support the dual scan display mode. In case of single scan mode, one of the FIFOs (FIFOH) can only be used.
STN LCD CONTROLLER OPERATION

TIMING GENERATOR (TIMEGEN)

The TIMEGEN generates the control signals for the LCD driver, such as VFRAME, VLINE, VCLK, and VM. These control signals are closely related to the configuration on the LCDCON1/2/3/4/5 registers in the REGBANK. Based on these programmable configurations on the LCD control registers in the REGBANK, the TIMEGEN can generate the programmable control signals suitable to support many different types of LCD drivers.

The VFRAME pulse is asserted for the duration of the entire first line at a frequency of once per frame. The VFRAME signal is asserted to bring the LCD’s line pointer to the top of the display to start over.

The VM signal helps the LCD driver alternate the polarity of the row and column voltages, which are used to turn the pixel on and off. The toggling rate of VM signals depends on the MMODE bit of the LCDCON1 register and MVAL field of the LCDCON4 register. If the MMODE bit is 0, the VM signal is configured to toggle on every frame. If the MMODE bit is 1, the VM signal is configured to toggle on the every event of the elapse of the specified number of VLINE by the MVAL[7:0] value. Figure 15-4 shows an example for MMODE=0 and for MMODE=1 with the value of MVAL[7:0]=0x2. When MMODE=1, the VM rate is related to MVAL[7:0], as shown below:

\[
\text{VM Rate} = \frac{\text{VLINE Rate}}{2 \times \text{MVAL}}
\]

The VFRAME and VLINE pulse generation relies on the configurations of the HOZVAL field and the LINEVAL field in the LCDCON2/3 register. Each field is related to the LCD size and display mode. In other words, the HOZVAL and LINEVAL can be determined by the size of the LCD panel and the display mode according to the following equation:

\[
\text{HOZVAL} = \frac{\text{Horizontal display size} - \text{Number of the valid VD data line}}{2}
\]

In color mode: Horizontal display size = 3 * Number of Horizontal Pixel

In the 4-bit single scan display mode, the Number of valid VD data line should be 4. In case of 4-bit dual scan display, the Number of valid VD data line should also be 4 while in case of 8-bit single scan display mode, the Number of valid VD data line should be 8.

\[
\text{LINEVAL} = \frac{\text{Vertical display size} - 1}{2} : \text{In case of single scan display type}
\]

\[
\text{LINEVAL} = \frac{\text{Vertical display size}}{2} - 1 : \text{In case of dual scan display type}
\]

The rate of VCLK signal depends on the configuration of the CLKVAL field in the LCDCON1 register. Table 15-1 defines the relationship of VCLK and CLKVAL. The minimum value of CLKVAL is 2.

\[
\text{VCLK(Hz)} = \frac{\text{HCLK}}{\text{CLKVAL} \times 2}
\]

The frame rate is the VFRAM signal frequency. The frame rate is closely related to the field of WLH[1:0][VLINE pulse width] WDLY[1:0] (the delay width of VCLK after VLINE pulse), HOZVAL, LINEBLANK, and LINEVAL in the LCDCON1/2/3/4 registers as well as VCLK and HCLK. Most LCD drivers need their own adequate frame rate. The frame rate is calculated as follows:

\[
\text{frame_rate(Hz)} = \frac{1}{\left\{ \left( \frac{1}{\text{VCLK}} \times (\text{HOZVAL}+1) + \frac{1}{\text{HCLK}} \times (A+B+(\text{LINEBLANK} \times 8)) \right) \times (\text{LINEVAL}+1) \right\}}
\]

\[
A = 2^{4+\text{WLH}}, \quad B = 2^{4+\text{WDLY}}
\]
Table 15-1. Relation Between VCLK and CLKVAL (STN, HCLK = 60 MHz)

<table>
<thead>
<tr>
<th>CLKVAL</th>
<th>60 MHz/X</th>
<th>VCLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>60 MHz/4</td>
<td>15.0 MHz</td>
</tr>
<tr>
<td>3</td>
<td>60 MHz/6</td>
<td>10.0 MHz</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>1023</td>
<td>60 MHz/2046</td>
<td>29.3 kHz</td>
</tr>
</tbody>
</table>

VIDEO OPERATION

The S3C2410X LCD controller supports 8-bit color mode (256 color mode), 12-bit color mode (4096 color mode), 4 level gray scale mode, 16 level gray scale mode as well as the monochrome mode. For the gray or color mode, it is required to implement the shades of gray level or color according to time-based dithering algorithm and Frame Rate Control (FRC) method. The selection can be made following a programmable lookup table, which will be explained later. The monochrome mode bypasses these modules (FRC and lookup table) and basically serializes the data in FIFOH (and FIFOL if a dual scan display type is used) into 4-bit (or 8-bit if a 4-bit dual scan or 8-bit single scan display type is used) streams by shifting the video data to the LCD driver.

The following sections describe the operation on the gray and color mode in terms of the lookup table and FRC.

Lookup Table

The S3C2410X can support the lookup table for various selection of color or gray level mapping, ensuring flexible operation for users. The lookup table is the palette which allows the selection on the level of color or gray (Selection on 4-gray levels among 16 gray levels in case of 4 gray mode, selection on 8 red levels among 16 levels, 8 green levels among 16 levels and 4 blue levels among 16 levels in case of 256 color mode). In other words, users can select 4 gray levels among 16 gray levels by using the lookup table in the 4 gray level mode. The gray levels cannot be selected in the 16 gray level mode; all 16 gray levels must be chosen among the possible 16 gray levels. In case of 256 color mode, 3 bits are allocated for red, 3 bits for green and 2 bits for blue. The 256 colors mean that the colors are formed from the combination of 8 red, 8 green and 4 blue levels (8x8x4 = 256). In the color mode, the lookup table can be used for suitable selections. Eight red levels can be selected among 16 possible red levels, 8 green levels among 16 green levels, and 4 blue levels among 16 blue levels. In case of 4096 color mode, there is no selection as in the 256 color mode.

Gray Mode Operation

The S3C2410X LCD controller supports two gray modes: 2-bit per pixel gray (4 level gray scale) and 4-bit per pixel gray (16 level gray scale). The 2-bit per pixel gray mode uses a lookup table (BLUELUT), which allows selection on 4 gray levels among 16 possible gray levels. The 2-bit per pixel gray lookup table uses the BULEVAL[15:0] in Blue Lookup Table (BLUELUT) register as same as blue lookup table in color mode. The gray level 0 will be denoted by BLUELVAL[3:0] value. If BLUELVAL[3:0] is 9, level 0 will be represented by gray level 9 among 16 gray levels. If BLUELVAL[3:0] is 15, level 0 will be represented by gray level 15 among 16 gray levels, and so on. Following the same method as above, level 1 will also be denoted by BLUELVAL[7:4], the level 2 by BLUELVAL[11:8], and the level 3 by BLUELVAL[15:12]. These four groups among BLUELVAL[15:0] will represent level 0, level 1, level 2, and level 3. In 16 gray levels, there is no selection as in the 16 gray levels.
256 Level Color Mode Operation

The S3C2410X LCD controller can support an 8-bit per pixel 256 color display mode. The color display mode can generate 256 levels of color using the dithering algorithm and FRC. The 8-bit per pixel are encoded into 3-bits for red, 3-bits for green, and 2-bits for blue. The color display mode uses separate lookup tables for red, green, and blue. Each lookup table uses the REDVAL[31:0] of REDLUT register, GREENVAL[31:0] of GREENLUT register, and BLUEVAL[15:0] of BLUELUT register as the programmable lookup table entries.

Similar to the gray level display, 8 group or field of 4 bits in the REDLUR register, i.e., REDVAL[31:28], REDLUT[27:24], REDLUT[23:20], REDLUT[19:16], REDLUT[15:12], REDLUT[11:8], REDLUT[7:4], and REDLUT[3:0], are assigned to each red level. The possible combination of 4 bits (each field) is 16, and each red level should be assigned to one level among possible 16 cases. In other words, the user can select the suitable red level by using this type of lookup table. For green color, the GREENVAL[31:0] of the GREENLUT register is assigned as the lookup table, as was done in the case of red color. Similarly, the BLUEVAL[15:0] of the BLUELUT register is also assigned as a lookup table. For blue color, 2 bits are allocated for 4 blue levels, different from the 8 red or green levels.

4096 Level Color Mode Operation

The S3C2410X LCD controller can support a 12-bit per pixel 4096 color display mode. The color display mode can generate 4096 levels of color using the dithering algorithm and FRC. The 12-bit per pixel are encoded into 4-bits for red, 4-bits for green, and 4-bits for blue. The 4096 color display mode does not use lookup tables.
DITHERING AND FRAME RATE CONTROL

For STN LCD displays (except monochrome), video data must be processed by a dithering algorithm. The DITHFRC block has two functions, such as Time-based Dithering Algorithm for reducing flicker and Frame Rate Control (FRC) for displaying gray and color level on the STN panel. The main principle of gray and color level display on the STN panel based on FRC is described. For example, to display the third gray (3/16) level from a total of 16 levels, the 3 times pixel should be on and 13 times pixel off. In other words, 3 frames should be selected among the 16 frames, of which 3 frames should have a pixel-on on a specific pixel while the remaining 13 frames should have a pixel-off on a specific pixel. These 16 frames should be displayed periodically. This is basic principle on how to display the gray level on the screen, so-called gray level display by FRC. The actual example is shown in Table 15-2. To represent the 14th gray level in the table, we should have a 6/7 duty cycle, which mean that there are 6 times pixel-on and one time pixel-off. The other cases for all gray levels are also shown in Table 15-2.

In the STN LCD display, we should be reminded of one item, i.e., Flicker Noise due to the simultaneous pixel-on and -off on adjacent frames. For example, if all pixels on first frame are turned on and all pixels on next frame are turned off, the Flicker Noise will be maximized. To reduce the Flicker Noise on the screen, the average probability of pixel-on and -off between frames should be the same. In order to realize this, the Time-based Dithering Algorithm, which varies the pattern of adjacent pixels on every frame, should be used. This is explained in detail. For the 16 gray level, FRC should have the following relationship between gray level and FRC. The 15th gray level should always have pixel-on, and the 14th gray level should have 6 times pixel-on and one times pixel-off, , , , , , and the 0th gray level should always have pixel-off as shown in Table 15-2.

<table>
<thead>
<tr>
<th>Pre-Dithered Data (gray level number)</th>
<th>Duty Cycle</th>
<th>Pre-Dithered Data (gray level number)</th>
<th>Duty Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1/2</td>
<td>15</td>
<td>1/2</td>
</tr>
<tr>
<td>14</td>
<td>6/7</td>
<td>14</td>
<td>6/7</td>
</tr>
<tr>
<td>13</td>
<td>4/5</td>
<td>13</td>
<td>4/5</td>
</tr>
<tr>
<td>12</td>
<td>3/4</td>
<td>12</td>
<td>3/4</td>
</tr>
<tr>
<td>11</td>
<td>5/7</td>
<td>11</td>
<td>5/7</td>
</tr>
<tr>
<td>10</td>
<td>2/3</td>
<td>10</td>
<td>2/3</td>
</tr>
<tr>
<td>9</td>
<td>3/5</td>
<td>9</td>
<td>3/5</td>
</tr>
<tr>
<td>8</td>
<td>4/7</td>
<td>8</td>
<td>4/7</td>
</tr>
</tbody>
</table>

**Table 15-2. Dither Duty Cycle Examples**
Display Types
The LCD controller supports 3 types of LCD drivers: 4-bit dual scan, 4-bit single scan, and 8-bit single scan display mode. Figure 15-2 shows these 3 different display types for monochrome displays, and Figure 15-3 show these 3 different display types for color displays.

4-bit Dual Scan Display Type
A 4-bit dual scan display uses 8 parallel data lines to shift data to both the upper and lower halves of the display at the same time. The 4 bits of data in the 8 parallel data lines are shifted to the upper half and 4 bits of data is shifted to the lower half, as shown in Figure 15-2. The end of frame is reached when each half of the display has been shifted and transferred. The 8 pins (VD[7:0]) for the LCD output from the LCD controller can be directly connected to the LCD driver.

4-bit Single Scan Display Type
A 4-bit single scan display uses 4 parallel data lines to shift data to successive single horizontal lines of the display at a time, until the entire frame has been shifted and transferred. The 4 pins (VD[3:0]) for the LCD output from the LCD controller can be directly connected to the LCD driver, and the 4 pins (VD[7:4]) for the LCD output are not used.

8-bit Single Scan Display Type
An 8-bit single scan display uses 8 parallel data lines to shift data to successive single horizontal lines of the display at a time, until the entire frame has been shifted and transferred. The 8 pins (VD[7:0]) for the LCD output from the LCD controller can be directly connected to the LCD driver.

256 Color Displays
Color displays require 3 bits (Red, Green, and Blue) of image data per pixel, and so the number of horizontal shift registers for each horizontal line corresponds to three times the number of pixels of one horizontal line, resulting in a horizontal shift register of length 3 times the number of pixels per horizontal line. This RGB is shifted to the LCD driver as consecutive bits via the parallel data lines. Figure 15-3 shows the RGB and order of the pixels in the parallel data lines for the 3 types of color displays.

4096 Color Displays
Color displays require 3 bits (Red, Green, and Blue) of image data per pixel, and so the number of horizontal shift registers for each horizontal line corresponds to three times the number of pixels of one horizontal line. This RGB is shifted to the LCD driver as consecutive bits via the parallel data lines. This RGB order is determined by the sequence of video data in video buffers.
MEMORY DATA FORMAT (STN, BSWP=0)

Mono 4-bit Dual Scan Display:

Video Buffer Memory:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>A[31:0]</td>
</tr>
<tr>
<td>0004H</td>
<td>B[31:0]</td>
</tr>
</tbody>
</table>

LCD Panel


1000H L[31:0]
1004H M[31:0]

Mono 4-bit Single Scan Display & 8-bit Single Scan Display:

Video Buffer Memory:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>A[31:0]</td>
</tr>
<tr>
<td>0004H</td>
<td>B[31:0]</td>
</tr>
<tr>
<td>0008H</td>
<td>C[31:0]</td>
</tr>
</tbody>
</table>

LCD Panel

MEMORY DATA FORMAT (STN, BSWP=0) (CONTINUED)

In 4-level gray mode, 2 bits of video data correspond to 1 pixel.

In 16-level gray mode, 4 bits of video data correspond to 1 pixel.

In 256 level color mode, 8 bits (3 bits of red, 3 bits of green, and 2 bits of blue) of video data correspond to 1 pixel. The color data format in a byte is as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
</tbody>
</table>

In 4096 level color mode, 12 bits (4 bits of red, 4 bits of green, 4 bits of blue) of video data correspond to 1 pixel. The following table shows color data format in words: (Video data must reside at 3 word boundaries (8 pixel), as follows)

**RGB Order**

<table>
<thead>
<tr>
<th>DATA</th>
<th>[31:28]</th>
<th>[27:24]</th>
<th>[23:20]</th>
<th>[19:16]</th>
<th>[15:12]</th>
<th>[11:8]</th>
<th>[7:4]</th>
<th>[3:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word #1</td>
<td>Red(1)</td>
<td>Green(1)</td>
<td>Blue(1)</td>
<td>Red(2)</td>
<td>Green(2)</td>
<td>Blue(2)</td>
<td>Red(3)</td>
<td>Green(3)</td>
</tr>
<tr>
<td>Word #2</td>
<td>Blue(3)</td>
<td>Red(4)</td>
<td>Green(4)</td>
<td>Blue(4)</td>
<td>Red(5)</td>
<td>Green(5)</td>
<td>Blue(5)</td>
<td>Red(6)</td>
</tr>
<tr>
<td>Word #3</td>
<td>Green(6)</td>
<td>Blue(6)</td>
<td>Red(7)</td>
<td>Green(7)</td>
<td>Blue(7)</td>
<td>Red(8)</td>
<td>Green(8)</td>
<td>Blue(8)</td>
</tr>
<tr>
<td>VD3</td>
<td>VD2</td>
<td>VD1</td>
<td>VD0</td>
<td>VD3</td>
<td>VD2</td>
<td>VD1</td>
<td>VD0</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**4-bit Dual Scan Display**

<table>
<thead>
<tr>
<th>VD3</th>
<th>VD2</th>
<th>VD1</th>
<th>VD0</th>
<th>VD3</th>
<th>VD2</th>
<th>VD1</th>
<th>VD0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**4-bit Single Scan Display**

<table>
<thead>
<tr>
<th>VD7</th>
<th>VD6</th>
<th>VD5</th>
<th>VD4</th>
<th>VD3</th>
<th>VD2</th>
<th>VD1</th>
<th>VD0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**8-bit Single Scan Display**

Figure 15-2. Monochrome Display Types (STN)
Figure 15-3. Color Display Types (STN)
Timing Requirements

Image data should be transferred from the memory to the LCD driver using the VD[7:0] signal. VCLK signal is used to clock the data into the LCD driver's shift register. After each horizontal line of data has been shifted into the LCD driver's shift register, the VLINE signal is asserted to display the line on the panel.

The VM signal provides an AC signal for the display. The LCD uses the signal to alternate the polarity of the row and column voltages, which are used to turn the pixels on and off, because the LCD plasma tends to deteriorate whenever subjected to a DC voltage. It can be configured to toggle on every frame or to toggle every programmable number of VLINE signals.

Figure 15-4 shows the timing requirements for the LCD driver interface.
Figure 15-4. 8-bit Single Scan Display Type STN LCD Timing
TFT LCD CONTROLLER OPERATION

The TIMEGEN generates the control signals for LCD driver, such as VSYNC, HSYNC, VCLK, VDEN, and LEND signal. These control signals are highly related with the configurations on the LCDCON1/2/3/4/5 registers in the REGBANK. Base on these programmable configurations on the LCD control registers in the REGBANK, the TIMEGEN can generate the programmable control signals suitable for the support of many different types of LCD drivers.

The VSYNC signal is asserted to cause the LCD's line pointer to start over at the top of the display.

The VSYNC and HSYNC pulse generation depends on the configurations of both the HOZVAL field and the LINEVAL field in the LCDCON2/3 registers. The HOZVAL and LINEVAL can be determined by the size of the LCD panel according to the following equations:

\[
\begin{align*}
    \text{HOZVAL} & = (\text{Horizontal display size}) -1 \\
    \text{LINEVAL} & = (\text{Vertical display size}) -1
\end{align*}
\]

The rate of VCLK signal depends on the CLKVAL field in the LCDCON1 register. Table 15-3 defines the relationship of VCLK and CLKVAL. The minimum value of CLKVAL is 0.

\[
VCLK(\text{Hz}) = \frac{HCLK}{(\text{CLKVAL}+1) \times 2}
\]

The frame rate is VSYNC signal frequency. The frame rate is related with the field of VSYNC, VBPD, VFPD, LINEVAL, HSYNC, HBPD, HFPD, HOZVAL, and CLKVAL in LCDCON1 and LCDCON2/3/4 registers. Most LCD drivers need their own adequate frame rate. The frame rate is calculated as follows:

\[
\text{Frame Rate} = \frac{1}{[(VSPW+1) + (VBPD+1) + (LINEVAL + 1) + (VFPD+1)] \times [(HSPW+1) + (HBPD +1) + (HFPD+1) + (HOZVAL + 1)] \times 2 \times (\text{CLKVAL}+1) / (HCLK)}
\]

<table>
<thead>
<tr>
<th>CLKVAL</th>
<th>60 MHz/X</th>
<th>VCLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60 MHz/4</td>
<td>15.0 MHz</td>
</tr>
<tr>
<td>2</td>
<td>60 MHz/6</td>
<td>10.0 MHz</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>1023</td>
<td>60 MHz/2048</td>
<td>30.0 kHz</td>
</tr>
</tbody>
</table>

VIDEO OPERATION

The TFT LCD controller within the S3C2410X supports 1, 2, 4 or 8 bpp (bit per pixel) palettized color displays and 16 or 24 bpp non-palettized true-color displays.

256 Color Palette

The S3C2410X can support the 256 color palette for various selection of color mapping, providing flexible operation for users.
MEMORY DATA FORMAT (TFT)

This section includes some examples of each display mode.

24BPP Display

(BSWP = 0, HWSWP = 0, BPP24BL = 0)

<table>
<thead>
<tr>
<th></th>
<th>D[31:24]</th>
<th>D[23:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>Dummy Bit</td>
<td>P1</td>
</tr>
<tr>
<td>004H</td>
<td>Dummy Bit</td>
<td>P2</td>
</tr>
<tr>
<td>008H</td>
<td>Dummy Bit</td>
<td>P3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(BSWP = 0, HWSWP = 0, BPP24BL = 1)

<table>
<thead>
<tr>
<th></th>
<th>D[31:8]</th>
<th>D[7:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P1</td>
<td>Dummy Bit</td>
</tr>
<tr>
<td>004H</td>
<td>P2</td>
<td>Dummy Bit</td>
</tr>
<tr>
<td>008H</td>
<td>P3</td>
<td>Dummy Bit</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

VD Pin Descriptions at 24BPP

<table>
<thead>
<tr>
<th>VD</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GREEN</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
16BPP Display
(BSWP = 0, HWSWP = 0)

<table>
<thead>
<tr>
<th></th>
<th>D[31:16]</th>
<th>D[15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>004H</td>
<td>P3</td>
<td>P4</td>
</tr>
<tr>
<td>008H</td>
<td>P5</td>
<td>P6</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(BSWP = 0, HWSWP = 1)

<table>
<thead>
<tr>
<th></th>
<th>D[31:16]</th>
<th>D[15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P2</td>
<td>P1</td>
</tr>
<tr>
<td>004H</td>
<td>P4</td>
<td>P3</td>
</tr>
<tr>
<td>008H</td>
<td>P6</td>
<td>P5</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VD Pin Connections at 16BPP
(5:6:5)

| VD  | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RED | 4  | 3  | 2  | 1  | 0  | NC |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| GREEN|    |    |    |    |    |    | 5  | 4  | 3  | 2  | 1  | 0  | NC |    |    |    |    |    |    |    |    |    |    |    |    |    |
| BLUE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | 4  | 3  | 2  | 1  |

(5:5:5:I)

| VD  | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RED | 4  | 3  | 2  | 1  | 0  | I  | NC |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| GREEN|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | 4  | 3  | 2  | 1  |
| BLUE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**NOTE:** The unused VD pins can be used as GPIO.
### 8BPP Display

**(BSWP = 0, HWSWP = 0)**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
</tr>
<tr>
<td>004H</td>
<td>P5</td>
<td>P6</td>
<td>P7</td>
<td>P8</td>
</tr>
<tr>
<td>008H</td>
<td>P9</td>
<td>P10</td>
<td>P11</td>
<td>P12</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**(BSWP = 1, HWSWP = 0)**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P4</td>
<td>P3</td>
<td>P2</td>
<td>P1</td>
</tr>
<tr>
<td>004H</td>
<td>P8</td>
<td>P7</td>
<td>P6</td>
<td>P5</td>
</tr>
<tr>
<td>008H</td>
<td>P12</td>
<td>P11</td>
<td>P10</td>
<td>P9</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 ......**

**LCD Panel**
### 4BPP Display

(BSWP = 0, HWSWP = 0)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
<td>P5</td>
<td>P6</td>
<td>P7</td>
</tr>
<tr>
<td>004H</td>
<td>P9</td>
<td>P10</td>
<td>P11</td>
<td>P12</td>
<td>P13</td>
<td>P14</td>
<td>P15</td>
</tr>
<tr>
<td>008H</td>
<td>P17</td>
<td>P18</td>
<td>P19</td>
<td>P20</td>
<td>P21</td>
<td>P22</td>
<td>P23</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(BSWP = 1, HWSWP = 0)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P7</td>
<td>P8</td>
<td>P5</td>
<td>P6</td>
<td>P3</td>
<td>P4</td>
<td>P1</td>
</tr>
<tr>
<td>004H</td>
<td>P15</td>
<td>P16</td>
<td>P13</td>
<td>P14</td>
<td>P11</td>
<td>P12</td>
<td>P9</td>
</tr>
<tr>
<td>008H</td>
<td>P23</td>
<td>P24</td>
<td>P21</td>
<td>P22</td>
<td>P19</td>
<td>P20</td>
<td>P17</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2BPP Display

(BSWP = 0, HWSWP = 0)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
<td>P5</td>
<td>P6</td>
<td>P7</td>
</tr>
<tr>
<td>004H</td>
<td>P17</td>
<td>P18</td>
<td>P19</td>
<td>P20</td>
<td>P21</td>
<td>P22</td>
<td>P23</td>
</tr>
<tr>
<td>008H</td>
<td>P33</td>
<td>P34</td>
<td>P35</td>
<td>P36</td>
<td>P37</td>
<td>P38</td>
<td>P39</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>P9</td>
<td>P10</td>
<td>P11</td>
<td>P12</td>
<td>P13</td>
<td>P14</td>
<td>P15</td>
</tr>
<tr>
<td>004H</td>
<td>P25</td>
<td>P26</td>
<td>P27</td>
<td>P28</td>
<td>P29</td>
<td>P30</td>
<td>P31</td>
</tr>
<tr>
<td>008H</td>
<td>P41</td>
<td>P42</td>
<td>P43</td>
<td>P44</td>
<td>P45</td>
<td>P46</td>
<td>P47</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
256 PALETTE USAGE (TFT)

Palette Configuration and Format Control

The S3C2410X provides 256 color palette for TFT LCD Control.
The user can select 256 colors from the 64K colors in these two formats.
The 256 color palette consists of the 256 (depth) × 16-bit SPSRAM. The palette supports 5:6:5 (R:G:B) format and 5:5:5:1 (R:G:B:I) format.

When the user uses 5:5:5:1 format, the intensity data (I) is used as a common LSB bit of each RGB data. So, 5:5:5:1 format is the same as R(5+I):G(5+I):B(5+I) format.
In 5:5:5:1 format, for example, the user can write the palette as in Table 15-5 and then connect VD pin to TFT LCD panel (R(5+I)=VD[23:19]+VD[18], VD[10] or VD[2], G(5+I)=VD[15:11]+VD[18], VD[10] or VD[2], B(5+I)=VD[7:3]+VD[18], VD[10] or VD[2]), and set FRM565 of LCDCON5 register to 0.

Table 15-4. 5:6:5 Format

<table>
<thead>
<tr>
<th>INDEX/Bit Pos.</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>G5</td>
<td>G4</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td>G0</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
<td>0X4D000400 (note1)</td>
</tr>
<tr>
<td>01H</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>G5</td>
<td>G4</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td>G0</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
<td>0X4D000404</td>
</tr>
<tr>
<td>FFH</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>G4</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td>G0</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
<td>0X4D0007FC</td>
<td></td>
</tr>
</tbody>
</table>

Table 15-5. 5:5:5:1 Format

<table>
<thead>
<tr>
<th>INDEX/Bit Pos.</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td>G0</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
<td>I</td>
<td>0X4D000400</td>
<td></td>
</tr>
<tr>
<td>01H</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td>G0</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
<td>I</td>
<td>0X4D000404</td>
<td></td>
</tr>
<tr>
<td>FFH</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td>G0</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
<td>I</td>
<td>0X4D0007FC</td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
1. 0x4D000400 is Palette start address.
2. VD18, VD10 and VD2 have the same output value, I.
3. DATA[31:16] is invalid.

Palette Read/Write

When the user performs Read/Write operation on the palette, HSTATUS and VSTATUS of LCDCON5 register must be checked, for Read/Write operation is prohibited during the ACTIVE status of HSTATUS and VSTATUS.

Temporary Palette Configuration

The S3C2410X allows the user to fill a frame with one color without complex modification to fill the one color to the frame buffer or palette. The one colored frame can be displayed by the writing a value of the color which is displayed on LCD panel to TPALVAL of TPAL register and enable TPALEN.
Figure 15-5. 16BPP Display Types (TFT)
Figure 15-6. TFT LCD Timing Example
SAMSUNG TFT LCD PANEL (3.5” PORTRAIT / 256K COLOR /REFLECTIVE A-SI TFT LCD)

The S3C2410X supports SEC TFT LCD panel (SAMSUNG 3.5” Portrait / 256K Color /Reflective a-Si TFT LCD).

LTS350Q1-PD1: TFT LCD panel with touch panel and front light unit
LTS350Q1-PD2: TFT LCD panel only

The S3C2410X provides timing signals as follows to use LTS350Q1-PD1 and PD2:

- STH: Horizontal Start Pulse
- TP: Source Driver Data Load Pulse
- INV: Digital Data Inversion
- LCD_HCLK: Horizontal Sampling Clock
- CPV: Vertical Shift Clock
- STV: Vertical Start Pulse
- OE: Gate On Enable
- REV: Inversion Signal
- REVb: Inversion Signal

So, LTS350Q1-PD1 and PD2 can be connected with the S3C2410X without using the additional timing control logic.

But the user should additionally apply Vcom generator circuit, various voltages, INV signal and Gray scale voltage generator circuit, which is recommended by PRODUCT INFORMATION (SPEC) of LTS350Q1-PD1 and PD2.

Detailed timing diagram is also described in PRODUCT INFORMATION (SPEC) of LTS350Q1-PD1 and PD2.

Refer to the documentation (PRODUCT INFORMATION of LTS350Q1-PD1 and PD2), which is prepared by AMLCD Technical Customer Center of Samsung Electronics Co., LTD.

Caution:

The S3C2410X has HCLK, working as the clock of AHB bus.

Accidentally, SEC TFT LCD panel (LTS350Q1-PD1 and PD2) has Horizontal Sampling Clock (HCLK). These two HCLKs may cause a confusion. So, note that HCLK of the S3C2410X is HCLK and other HCLK of the LTS350 is LCD_HCLK.

Check that the HCLK of SEC TFT LCD panel (LTS350Q1-PD1 and PD2) is changed to LCD_HCLK.
VIRTUAL DISPLAY (TFT/STN)

The S3C2410X supports hardware horizontal or vertical scrolling. If the screen is scrolled, the fields of LCDBASEU and LCDBASEL in LCDSADDR1/2 registers need to be changed (see Figure 15-8), except the values of PAGEWIDTH and OFFSIZE.

The video buffer in which the image is stored should be larger than the LCD panel screen in size.

Figure 15-7. Example of Scrolling in Virtual Display (Single Scan)
LCD POWER ENABLE (STN/TFT)

The S3C2410X provides Power enable (PWREN) function. When PWREN is set to make PWREN signal enabled, the output value of LCD_PWREN pin is controlled by ENVID. In other words, if LCD_PWREN pin is connected to the power on/off control pin of the LCD panel, the power of LCD panel is controlled by the setting of ENVID automatically.

The S3C2410X also supports INVPWREN bit to invert polarity of the PWREN signal.

This function is available only when LCD panel has its own power on/off control port and when port is connected to LCD_PWREN pin.

Figure 15-8. Example of PWREN Function (PWREN = 1, INVPWREN = 0)
## LCD CONTROL SPECIAL REGISTERS

### LCD Control 1 Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDCON1</td>
<td>0x4D000000</td>
<td>R/W</td>
<td>LCD control 1 register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCDCON1</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINECNT (read only)</td>
<td>[27:18]</td>
<td>Provide the status of the line counter. Down count from LINEVAL to 0</td>
<td>0000000000</td>
</tr>
<tr>
<td>CLKVAL</td>
<td>[17:8]</td>
<td>Determine the rates of VCLK and CLKVAL[9:0]. <strong>STN</strong>: VCLK = HCLK / (CLKVAL × 2) (CLKVAL ≥ 2) <strong>TFT</strong>: VCLK = HCLK / [(CLKVAL+1) × 2] (CLKVAL ≥ 0)</td>
<td>0000000000</td>
</tr>
<tr>
<td>MMODE</td>
<td>[7]</td>
<td>Determine the toggle rate of the VM. 0 = Each Frame, 1 = The rate defined by the MVAL</td>
<td>0</td>
</tr>
<tr>
<td>PNRMODE</td>
<td>[6:5]</td>
<td>Select the display mode. 00 = 4-bit dual scan display mode (STN) 01 = 4-bit single scan display mode (STN) 10 = 8-bit single scan display mode (STN) 11 = TFT LCD panel</td>
<td>00</td>
</tr>
<tr>
<td>BPPMODE</td>
<td>[4:1]</td>
<td>Select the BPP (Bits Per Pixel) mode. 0000 = 1 bpp for STN, Monochrome mode 0001 = 2 bpp for STN, 4-level gray mode 0010 = 4 bpp for STN, 16-level gray mode 0011 = 8 bpp for STN, color mode 0100 = 12 bpp for STN, color mode 1000 = 1 bpp for TFT 1001 = 2 bpp for TFT 1010 = 4 bpp for TFT 1011 = 8 bpp for TFT 1100 = 16 bpp for TFT 1101 = 24 bpp for TFT</td>
<td>0000</td>
</tr>
<tr>
<td>ENVID</td>
<td>[0]</td>
<td>LCD video output and the logic enable/disable. 0 = Disable the video output and the LCD control signal. 1 = Enable the video output and the LCD control signal.</td>
<td>0</td>
</tr>
</tbody>
</table>
### LCD Control 2 Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDCON2</td>
<td>0X4D000004</td>
<td>R/W</td>
<td>LCD control 2 register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCDCON2</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBPD</td>
<td>[31:24]</td>
<td><strong>TFT:</strong> Vertical back porch is the number of inactive lines at the start of a frame, after vertical synchronization period. <strong>STN:</strong> These bits should be set to zero on STN LCD.</td>
<td>0x00</td>
</tr>
<tr>
<td>LINEVAL</td>
<td>[23:14]</td>
<td><strong>TFT/STN:</strong> These bits determine the vertical size of LCD panel.</td>
<td>00000000000</td>
</tr>
<tr>
<td>VFPD</td>
<td>[13:6]</td>
<td><strong>TFT:</strong> Vertical front porch is the number of inactive lines at the end of a frame, before vertical synchronization period. <strong>STN:</strong> These bits should be set to zero on STN LCD.</td>
<td>00000000</td>
</tr>
<tr>
<td>VSPW</td>
<td>[5:0]</td>
<td><strong>TFT:</strong> Vertical sync pulse width determines the VSYNC pulse's high level width by counting the number of inactive lines. <strong>STN:</strong> These bits should be set to zero on STN LCD.</td>
<td>000000</td>
</tr>
</tbody>
</table>
## LCD Control 3 Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDCON3</td>
<td>0X4D000008</td>
<td>R/W</td>
<td>LCD control 3 register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### Bit Description

- **HBPD (TFT) [25:19]**
  - **TFT**: Horizontal back porch is the number of VCLK periods between the falling edge of HSYNC and the start of active data.
  - **STN**: WDLY[1:0] bits determine the delay between VLINE and VCLK by counting the number of the HCLK. WDLY[7:2] are reserved.  
    - 00 = 16 HCLK, 01 = 32 HCLK, 10 = 48 HCLK, 11 = 64 HCLK

- **WDLY (STN)**
  - **STN**: WDLY[1:0] bits determine the delay between VLINE and VCLK by counting the number of the HCLK. WDLY[7:2] are reserved.  
    - 00 = 16 HCLK, 01 = 32 HCLK, 10 = 48 HCLK, 11 = 64 HCLK

- **HOZVAL [18:8]**
  - **TFT/STN**: These bits determine the horizontal size of LCD panel.  
    - HOZVAL has to be determined to meet the condition that total bytes of 1 line are 4n bytes. If the x size of LCD is 120 dot in mono mode, x=120 cannot be supported because 1 line consists of 15 bytes.  
    - Instead, x=128 in mono mode can be supported because 1 line is composed of 16 bytes (2n). LCD panel driver will discard the additional 8 dot.

- **HFPD (TFT) [7:0]**
  - **TFT**: Horizontal front porch is the number of VCLK periods between the end of active data and the rising edge of HSYNC.

- **LINEBLANK (STN)**
  - **STN**: These bits indicate the blank time in one horizontal line duration time. These bits adjust the rate of the VLINE finely.  
    - The unit of LINEBLANK is HCLK x 8.  
    - Ex) If the value of LINEBLANK is 10, the blank time is inserted to VCLK during 80 HCLK.
### LCD Control 4 Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDCON4</td>
<td>0X4D00000C</td>
<td>R/W</td>
<td>LCD control 4 register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCDCON4</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVAL</td>
<td>[15:8]</td>
<td><strong>STN</strong>: These bit define the rate at which the VM signal will toggle if the MMODE bit is set to logic ‘1’.</td>
<td>0X00</td>
</tr>
<tr>
<td>HSPW(TFT)</td>
<td>[7:0]</td>
<td><strong>TFT</strong>: Horizontal sync pulse width determines the HSYNC pulse's high level width by counting the number of the VCLK.</td>
<td>0X00</td>
</tr>
<tr>
<td>WLH(STN)</td>
<td></td>
<td><strong>STN</strong>: WLH[1:0] bits determine the VLINE pulse's high level width by counting the number of the HCLK. WLH[7:2] are reserved.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = 16 HCLK, 01 = 32 HCLK, 10 = 48 HCLK, 11 = 64 HCLK</td>
<td></td>
</tr>
</tbody>
</table>
### LCD Control 5 Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDCON5</td>
<td>0x4d000010</td>
<td>R/W</td>
<td>LCD control 5 register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

#### LCDCON5 Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[31:17] This bit is reserved and the value should be '0'.</td>
<td>0</td>
</tr>
<tr>
<td>VSTATUS</td>
<td>[16:15] TFT: Vertical Status (read only). 00 = VSYNC, 01 = BACK Porch, 10 = ACTIVE, 11 = FRONT Porch</td>
<td>00</td>
</tr>
<tr>
<td>HSTATUS</td>
<td>[14:13] TFT: Horizontal Status (read only). 00 = HSYNC, 01 = BACK Porch, 10 = ACTIVE, 11 = FRONT Porch</td>
<td>00</td>
</tr>
<tr>
<td>BPP24BL</td>
<td>[12] TFT: This bit determines the order of 24 bpp video memory. 0 = LSB valid, 1 = MSB Valid</td>
<td>0</td>
</tr>
<tr>
<td>FRM565</td>
<td>[11] TFT: This bit selects the format of 16 bpp output video data. 0 = 5:5:5:1 Format, 1 = 5:6:5 Format</td>
<td>0</td>
</tr>
<tr>
<td>INVVCLK</td>
<td>[10] STN/TFT: This bit controls the polarity of the VCLK active edge. 0 = The video data is fetched at VCLK falling edge, 1 = The video data is fetched at VCLK rising edge</td>
<td>0</td>
</tr>
<tr>
<td>INVVLINE</td>
<td>[9] STN/TFT: This bit indicates the VLINE/HSYNC pulse polarity. 0 = Normal, 1 = Inverted</td>
<td>0</td>
</tr>
<tr>
<td>INVVFRAME</td>
<td>[8] STN/TFT: This bit indicates the VFRAME/VSYNC pulse polarity. 0 = Normal, 1 = Inverted</td>
<td>0</td>
</tr>
<tr>
<td>INVVD</td>
<td>[7] STN/TFT: This bit indicates the VD (video data) pulse polarity. 0 = Normal, 1 = VD is inverted.</td>
<td>0</td>
</tr>
</tbody>
</table>
## LCD Control 5 Register (Continued)

<table>
<thead>
<tr>
<th>LCDCON5</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
</table>
| INVVDEN | [6] | TFT: This bit indicates the VDEN signal polarity.  
0 = Normal  
1 = Inverted | 0 |
| INVPWREN | [5] | STN/TFT: This bit indicates the PWREN signal polarity.  
0 = Normal  
1 = Inverted | 0 |
| INVLEND | [4] | TFT: This bit indicates the LEND signal polarity.  
0 = Normal  
1 = Inverted | 0 |
0 = Disable PWREN signal  
1 = Enable PWREN signal | 0 |
0 = Disable LEND signal  
1 = Enable LEND signal | 0 |
0 = Swap Disable  
1 = Swap Enable | 0 |
0 = Swap Disable  
1 = Swap Enable | 0 |
**FRAME BUFFER START ADDRESS 1 REGISTER**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDSADDR1</td>
<td>0x4D000014</td>
<td>R/W</td>
<td><strong>STN/TFT</strong>: Frame buffer start address 1 register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>LCDBANK</td>
<td>[29:21]</td>
<td></td>
<td>These bits indicate A[30:22] of the bank location for the</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>video buffer in the system memory. LCDBANK value cannot be</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>changed even when moving the view port. LCD frame buffer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>should be within aligned 4MB region, which ensures that</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LCDBANK value will not be changed when moving the view port.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>So, care should be taken to use the malloc() function.</td>
<td></td>
</tr>
<tr>
<td>LCDBASEU</td>
<td>[20:0]</td>
<td></td>
<td>For dual-scan LCD: These bits indicate A[21:1] of the start</td>
<td>0x00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>address of the upper address counter, which is for the upper</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>frame memory of dual scan LCD or the frame memory of single</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scan LCD. For single-scan LCD: These bits indicate A[21:1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>of the start address of the LCD frame buffer.</td>
<td></td>
</tr>
</tbody>
</table>

**FRAME Buffer Start Address 2 Register**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDSADDR2</td>
<td>0x4D000018</td>
<td>R/W</td>
<td><strong>STN/TFT</strong>: Frame buffer start address 2 register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>LCDBASEL</td>
<td>[20:0]</td>
<td></td>
<td>For dual-scan LCD: These bits indicate A[21:1] of the start</td>
<td>0x0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>address of the lower address counter, which is used for the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lower frame memory of dual scan LCD. For single scan LCD:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>These bits indicate A[21:1] of the end address of the LCD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>frame buffer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LCDBASEL = ((the frame end address) &gt;&gt;1) + 1 +</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(PAGEWIDTH+OFFSIZE) x (LINEVAL+1)</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** Users can change the LCDBASEU and LCDBASEL values for scrolling while the LCD controller is turned on. But, users must not change the value of the LCDBASEU and LCDBASEL registers at the end of FRAME by referring to the LINECNT field in LCDCON1 register, for the LCD FIFO fetches the next frame data prior to the change in the frame. So, if you change the frame, the pre-fetched FIFO data will be obsolete and LCD controller will display an incorrect screen. To check the LINECNT, interrupts should be masked. If any interrupt is executed just after reading LINECNT, the read LINECNT value may be obsolete because of the execution time of Interrupt Service Routine (ISR).
### FRAME Buffer Start Address 3 Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDSADDR3</td>
<td>0x4D000001C</td>
<td>R/W</td>
<td>STN/TFT: Virtual screen address set</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCDSADDR3 Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSIZE [21:11]</td>
<td>Virtual screen offset size (the number of half words). This value defines the difference between the address of the last half word displayed on the previous LCD line and the address of the first half word to be displayed in the new LCD line.</td>
<td>00000000000</td>
</tr>
<tr>
<td>PAGEWIDTH [10:0]</td>
<td>Virtual screen page width (the number of half words). This value defines the width of the view port in the frame.</td>
<td>0000000000</td>
</tr>
</tbody>
</table>

**NOTE:** The values of PAGEWIDTH and OFFSIZE must be changed when ENVID bit is 0.

Example 1. LCD panel = 320*240, 16gray, single scan
Frame start address = 0xc500000
Offset dot number = 2048 dots (512 half words)

\[
\begin{align*}
\text{LINEVAL} &= 240 \times 1 = \text{0xef} \\
\text{PAGEWIDTH} &= 320 \times 4 / 16 = 0\times50 \\
\text{OFFSIZE} &= 512 \times 0 = \text{0x200} \\
\text{LCDBANK} &= 0xc500000 >> 22 = 0\times31 \\
\text{LCDBASEU} &= 0x100000 >> 1 = 0\times80000 \\
\text{LCDBASEL} &= 0x80000 + (0\times50 + 0\times200) \times (0\timesef + 1) = 0\text{xa2b00}
\end{align*}
\]

Example 2. LCD panel = 320*240, 16gray, dual scan
Frame start address = 0xc500000
Offset dot number = 2048 dots (512 half words)

\[
\begin{align*}
\text{LINEVAL} &= 120 \times 1 = \text{0x77} \\
\text{PAGEWIDTH} &= 320 \times 4 / 16 = 0\times50 \\
\text{OFFSIZE} &= 512 \times 0 = \text{0x200} \\
\text{LCDBANK} &= 0xc500000 >> 22 = 0\times31 \\
\text{LCDBASEU} &= 0x100000 >> 1 = 0\times80000 \\
\text{LCDBASEL} &= 0x80000 + (0\times50 + 0\times200) \times (0\times77 + 1) = 0\text{x91580}
\end{align*}
\]

Example 3. LCD panel = 320*240, color, single scan
Frame start address = 0xc500000
Offset dot number = 1024 dots (512 half words)

\[
\begin{align*}
\text{LINEVAL} &= 240 \times 1 = \text{0xef} \\
\text{PAGEWIDTH} &= 320 \times 8 / 16 = 0\xa0 \\
\text{OFFSIZE} &= 512 \times 0 = \text{0x200} \\
\text{LCDBANK} &= 0xc500000 >> 22 = 0\times31 \\
\text{LCDBASEU} &= 0x100000 >> 1 = 0\times80000 \\
\text{LCDBASEL} &= 0x80000 + (0\xa0 + 0\times200) \times (0\text{xef} + 1) = 0\text{x7600}
\end{align*}
\]
## RED Lookup Table Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REDLUT</td>
<td>0x4D000020</td>
<td>R/W</td>
<td>STN: Red lookup table register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### REDLUT Bit Description

**REDVAL [31:0]** These bits define which of the 16 shades will be chosen by each of the 8 possible red combinations.

- 000 = REDVAL[3:0], 001 = REDVAL[7:4]
- 010 = REDVAL[11:8], 011 = REDVAL[15:12]
- 100 = REDVAL[19:16], 101 = REDVAL[23:20]
- 110 = REDVAL[27:24], 111 = REDVAL[31:28]

### GREEN Lookup Table Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREENLUT</td>
<td>0x4D000024</td>
<td>R/W</td>
<td>STN: Green lookup table register</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### GREENLUT Bit Description

**GREENVAL [31:0]** These bits define which of the 16 shades will be chosen by each of the 8 possible green combinations.

- 000 = GREENVAL[3:0], 001 = GREENVAL[7:4]
- 010 = GREENVAL[11:8], 011 = GREENVAL[15:12]
- 100 = GREENVAL[19:16], 101 = GREENVAL[23:20]
- 110 = GREENVAL[27:24], 111 = GREENVAL[31:28]

### BLUE Lookup Table Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUELUT</td>
<td>0x4D000028</td>
<td>R/W</td>
<td>STN: Blue lookup table register</td>
<td>0x0000</td>
</tr>
</tbody>
</table>

### BLUELUT Bit Description

**BLUEVAL [15:0]** These bits define which of the 16 shades will be chosen by each of the 4 possible blue combinations.

- 00 = BLUEVAL[3:0], 01 = BLUEVAL[7:4]
- 10 = BLUEVAL[11:8], 11 = BLUEVAL[15:12]

**NOTE:** Address from 0x14A0002C to 0x14A00048 should not be used. This area is reserved for Test mode.
## Dithering Mode Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DITHMODE</td>
<td>0X4D00004C</td>
<td>R/W</td>
<td>STN: Dithering mode register. This register reset value is 0x00000 But, user can change this value to 0x12210. (Refer to a sample program source for the latest value of this register.)</td>
<td>0x00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DITHMODE</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>DITHMODE</td>
<td>[18:0]</td>
<td>Use one of following value for your LCD: 0x00000 or 0x12210</td>
<td>0x00000</td>
</tr>
</tbody>
</table>
## Temp Palette Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPAL</td>
<td>0X4D000050</td>
<td>R/W</td>
<td>TFT: Temporary palette register. This register value will be video data at next frame.</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TPAL</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPALEN</td>
<td>[24]</td>
<td>Temporary palette register enable bit. 0 = Disable 1 = Enable</td>
<td>0</td>
</tr>
</tbody>
</table>
### LCD Interrupt Pending Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCINTPND</td>
<td>0X4D000054</td>
<td>R/W</td>
<td>Indicate the LCD interrupt pending register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCINTPND Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_FrSyn</td>
<td>LCD frame synchronized interrupt pending bit.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = The interrupt has not been requested.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = The frame has asserted the interrupt request.</td>
<td></td>
</tr>
<tr>
<td>INT_FiCnt</td>
<td>LCD FIFO interrupt pending bit.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = The interrupt has not been requested.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = LCD FIFO interrupt is requested when LCD FIFO reaches trigger level.</td>
<td></td>
</tr>
</tbody>
</table>

### LCD Source Pending Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDSRCMPND</td>
<td>0X4D000058</td>
<td>R/W</td>
<td>Indicate the LCD interrupt source pending register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCDSRCMPND Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_FrSyn</td>
<td>LCD frame synchronized interrupt source pending bit.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = The interrupt has not been requested.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = The frame has asserted the interrupt request.</td>
<td></td>
</tr>
<tr>
<td>INT_FiCnt</td>
<td>LCD FIFO interrupt source pending bit.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = The interrupt has not been requested.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = LCD FIFO interrupt is requested when LCD FIFO reaches trigger level.</td>
<td></td>
</tr>
</tbody>
</table>
### LCD Interrupt Mask Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDINTMSK</td>
<td>0X4D00005C</td>
<td>R/W</td>
<td>Determine which interrupt source is masked. The masked interrupt source will not be serviced.</td>
<td>0x3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCDINTMSK</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIWSEL</td>
<td>[2]</td>
<td>Determine the trigger level of LCD FIFO. 0 = 4 words 1 = 8 words</td>
<td></td>
</tr>
<tr>
<td>INT_FrSyn</td>
<td>[1]</td>
<td>Mask LCD frame synchronized interrupt. 0 = The interrupt service is available. 1 = The interrupt service is masked.</td>
<td>1</td>
</tr>
<tr>
<td>INT_FiCnt</td>
<td>[0]</td>
<td>Mask LCD FIFO interrupt. 0 = The interrupt service is available. 1 = The interrupt service is masked.</td>
<td>1</td>
</tr>
</tbody>
</table>

### LPC3600 Control Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPCSEL</td>
<td>0X4D000060</td>
<td>R/W</td>
<td>This register controls the LPC3600 modes.</td>
<td>0x4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LPCSEL</th>
<th>Bit</th>
<th>Description</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES_SEL</td>
<td>[1]</td>
<td>1 = 240×320</td>
<td>0</td>
</tr>
<tr>
<td>LPC_EN</td>
<td>[0]</td>
<td>Determine LPC3600 Enable/Disable. 0 = LPC3600 Disable 1 = LPC3600 Enable</td>
<td>0</td>
</tr>
</tbody>
</table>
Register Setting Guide (STN)

The LCD controller supports multiple screen sizes by special register setting.

The CLKVAL value determines the frequency of VCLK. This value has to be determined such that the VCLK value is greater than data transmission rate. The data transmission rate for the VD port of the LCD controller is used to determine the value of CLKVAL register.

The data transmission rate is given by the following equation:

\[
\text{Data transmission rate} = HS \times VS \times FR \times MV
\]

- **HS**: Horizontal LCD size
- **VS**: Vertical LCD size
- **FR**: Frame rate
- **MV**: Mode dependent value

<table>
<thead>
<tr>
<th>Mode</th>
<th>MV Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mono, 4-bit single scan display</td>
<td>1/4</td>
</tr>
<tr>
<td>Mono, 8-bit single scan display or 4-bit dual scan display</td>
<td>1/8</td>
</tr>
<tr>
<td>4 level gray, 4-bit single scan display</td>
<td>1/4</td>
</tr>
<tr>
<td>4 level gray, 8-bit single scan display or 4-bit dual scan display</td>
<td>1/8</td>
</tr>
<tr>
<td>16 level gray, 4-bit single scan display</td>
<td>1/4</td>
</tr>
<tr>
<td>16 level gray, 8-bit single scan display or 4-bit dual scan display</td>
<td>1/8</td>
</tr>
<tr>
<td>Color, 4-bit single scan display</td>
<td>3/4</td>
</tr>
<tr>
<td>Color, 8-bit single scan display or 4-bit dual scan display</td>
<td>3/8</td>
</tr>
</tbody>
</table>

The LCDBASEU register value is the first address value of the frame buffer. The lowest 4 bits must be eliminated for burst 4 word access. The LCDBASEL register value depends on LCD size and LCDBASEU. The LCDBASEL value is given by the following equation:

\[
\text{LCDBASEL} = \text{LCDBASEU} + \text{LCDBASEL offset}
\]
Example 1:
160 x 160, 4-level gray, 80 frame/sec, 4-bit single scan display, HCLK frequency is 60 MHz WLH = 1, WDLY = 1.

Data transmission rate = $160 \times 160 \times 80 \times \frac{1}{4} = 512 \text{ kHz}$

CLKVAL = 58, VCLK = 517 kHz
HOZVAL = 39, LINEVAL = 159

LINEBLANK = 10

LCDBASEL = LCDBASEU + 3200

NOTE: The higher the system load is, the lower the cpu performance is.

Example 2 (Virtual Screen Register):
4-level gray, Virtual screen size = 1024 x 1024, LCD size = 320 x 240, LCDBASEU = 0x64, 4-bit dual scan.

1 halfword = 8 pixels (4-level gray),
Virtual screen 1 line = 128 halfword = 1024 pixels,
LCD 1 line = 320 pixels = 40 halfword,
OFFSIZE = 128 - 40 = 88 = 0x58,
PAGEWIDTH = 40 = 0x28

LCDBASEL = LCDBASEU + (PAGEWIDTH + OFFSIZE) \times (LINEVAL +1) = 100 + (40 + 88) \times 120 = 0 \times 3C64
Gray Level Selection Guide

The S3C2410X LCD controller can generate 16 gray level using Frame Rate Control (FRC). The FRC characteristics may cause unexpected patterns in gray level. These unwanted erroneous patterns may be shown in fast response LCD or at lower frame rates.

Because the quality of LCD gray levels depends on LCD's own characteristics, the user has to select an appropriate gray level after viewing all gray levels on user's own LCD.

Select the gray level quality through the following procedures:
1. Get the latest dithering pattern register value from SAMSUNG.
2. Display 16 gray bar in LCD.
3. Change the frame rate into an optimal value.
4. Change the VM alternating period to get the best quality.
5. As viewing 16 gray bars, select a good gray level, which is displayed well on your LCD.
6. Use only the good gray levels for quality.

LCD Refresh Bus Bandwidth Calculation Guide

The S3C2410X LCD controller can support various LCD display sizes. To select a suitable size (for the flicker free LCD system application), the user have to consider the LCD refresh bus bandwidth determined by the LCD display size, bit per pixel (bpp), frame rate, memory bus width, memory type, and so on.

$$\text{LCD Data Rate (Byte/s)} = \text{bpp} \times (\text{Horizontal display size}) \times (\text{Vertical display size}) \times (\text{Frame rate}) / 8$$

$$\text{LCD DMA Burst Count (Times/s)} = \text{LCD Data Rate (Byte/s)} / 16(\text{Byte}) ; \text{LCD DMA using 4words(16Byte) burst}$$

Pdma means LCD DMA access period. In other words, the value of Pdma indicates the period of four-beat burst (4-words burst) for video data fetch. So, Pdma depends on memory type and memory setting.

Eventually, LCD System Load is determined by LCD DMA Burst Count and Pdma.

$$\text{LCD System Load} = \text{LCD DMA Burst Count} \times \text{Pdma}$$

Example 3:
640 \times 480, 8bpp, 60 frame/sec, 16-bit data bus width, SDRAM (Trp=2HCLK / Trcd=2HCLK / CL=2HCLK) and HCLK frequency is 60 MHz

$$\text{LCD Data Rate} = 8 \times 640 \times 480 \times 60 / 8 = 18.432 \text{Mbyte/s}$$

$$\text{LCD DMA Burst Count} = 18.432 / 16 = 1.152 \text{M/s}$$

$$\text{Pdma} = (\text{Trp}+\text{Trcd}+\text{CL}+(2 \times 4)+1) \times (1/60 \text{ MHz}) = 0.250 \text{ms}$$

$$\text{LCD System Load} = 1.152 \times 250 = 0.288$$

System Bus Occupation Rate = (0.288/1) \times 100 = 28.8\%
Register Setting Guide (TFT LCD)

The CLKVAL register value determines the frequency of VCLK and frame rate.

Frame Rate = \(1/\left[\left(\text{VSPW} + 1 \right) + \left(\text{VBPD} + 1 \right) + \left(\text{LINEVAL} + 1 \right) + \left(\text{VFPD} + 1 \right)\right] \times \left[\left(\text{HSPW} + 1 \right) + \left(\text{HBPD} + 1 \right) + \left(\text{HOZVAL} + 1 \right)\right] \times \left(\frac{2 \times (\text{CLKVAL} + 1)}{\text{HCLK}}\right)\]

For applications, the system timing must be considered to avoid under-run condition of the fifo of the lcd controller caused by memory bandwidth contention.

**Example 4:**

TFT Resolution: 240 × 240,

VSPW = 2, VBPD = 14, LINEVAL = 239, VFPD = 4

HSPW = 25, HBPD = 15, HOZVAL = 239, HFPD = 1

CLKVAL = 5

HCLK = 60M (Hz)

The parameters below must be referenced by LCD size and driver specifications:

VSPW, VBPD, LINEVAL, VFPD, HSPW, HBPD, HOZVAL, and HFPD

If target frame rate is 60–70Hz, then CLKVAL should be 5.

So, Frame Rate = 67Hz

**Known Problems**

Problem: In a MDS, such as Multi-ICE, some of the LCD controller registers may be displayed incorrectly in the memory view window of the ARM debugger.

Solution: The LCD controller register will be displayed correctly unless the memory view window is used. Instead, use 'pr' command in the debugger console window.
16 ADC & TOUCH SCREEN INTERFACE

OVERVIEW

The 10-bit CMOS analog to digital converter (ADC) of the S3C2410X is a recycling typed device with 8-channel analog inputs. It converts the analog input signal into 10-bit binary digital codes at a maximum conversion rate of 500KSPS with 2.5 MHz A/D converter clock. The A/D converter operates with on-chip, sample-and-hold function and power down mode is supported.

The S3C2410X supports Touch Screen Interface, which consists of a touch screen panel, four external transistors, an external voltage source, AIN[7] and AIN[5] (see Figure 16-2). Touch Screen Interface controls and selects control signals (nYPON, YMON, nXPON and XMON) and analog pads (AIN[7], AIN[5]) which are connected with pads of touch screen panel and the external transistor for X-position conversion and Y-position conversion. Touch Screen Interface contains an external transistor control logic and an ADC interface logic with an interrupt generation logic.

FEATURES

- Resolution: 10-bit
- Differential Linearity Error: ±1.5 LSB
- Integral Linearity Error: ±2.0 LSB
- Maximum Conversion Rate: 500 KSPS
- Low Power Consumption
- Power Supply Voltage: 3.3V
- Analog Input Range: 0 ~ 3.3V
- On-chip Sample-and-hold Function
- Normal Conversion Mode
- Separate X/Y position conversion Mode
- Auto (Sequential) X/Y Position Conversion Mode
- Waiting for Interrupt Mode
ADC & TOUCH SCREEN INTERFACE OPERATION

BLOCK DIAGRAM

Figure 16-1 shows the functional block diagram of the S3C2410X A/D converter and Touch Screen Interface. Note that the A/D converter is a recycling type. A pull-up resister is attached to AIN[7] on VDDA_ADC. So, XP pad of the touch screen panel should be connected with AIN[7] of the S3C2410X and YP pad of the touch screen panel should be connected with AIN[5].

Figure 16-1. ADC and Touch Screen Interface Functional Block Diagram
EXAMPLE FOR TOUCH SCREEN

In this example, AIN[7] is connected with XP and AIN[5] is connected with YP pad of the touch screen panel. To control pads of the touch screen panel (XP, XM, YP and YM), four external transistors are applied and control signals including nYPON, YMON, nXPON and XMON are connected with four external transistors.

<table>
<thead>
<tr>
<th>External Transistor Control</th>
<th>Touch Panel</th>
<th>External Voltage Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>nYPON</td>
<td>XM</td>
<td>S3C2410X01</td>
</tr>
<tr>
<td>YMON</td>
<td>YP</td>
<td></td>
</tr>
<tr>
<td>nXPON</td>
<td>YM</td>
<td></td>
</tr>
<tr>
<td>XMON</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 16-2. Example of ADC and Touch Screen Interface

The following procedure is suggested:

1. Connect pads of the touch screen panel to the S3C2410X using external transistor (see Figure 16-2).
2. Select Separate X/Y Position Conversion Mode or Auto (Sequential) X/Y Position Conversion Mode to get X/Y position.
3. Set Touch Screen Interface to Waiting Interrupt Mode.
4. If interrupt occurs, then appropriate conversion (Separate X/Y Position Conversion Mode or Auto (Sequential) X/Y Position Conversion Mode) is activated.
5. After get the proper value about X/Y position, return to Waiting for Interrupt Mode.

NOTES:
1. External voltage source should be 3.3 V.
2. Internal resistance of the external transistor should be under 5 ohm.
FUNCTION DESCRIPTIONS

A/D Conversion Time
When the PCLK frequency is 50 MHz and the prescaler value is 49, total 10-bit conversion time is given:

\[
\text{A/D converter freq.} = \frac{50 \text{ MHz}}{49+1} = 1 \text{ MHz}
\]
\[
\text{Conversion time} = \frac{1}{1 \text{ MHz} / 5\text{cycles}} = \frac{1}{200 \text{ kHz}} = 5 \text{ us}
\]

NOTE
This A/D converter is designed to operate at maximum 2.5 MHz clock, so the conversion rate can go up to 500 KSPS.

Touch Screen Interface Mode
1. Normal Conversion Mode
   Normal Conversion Mode (AUTO_PST = 0, XY_PST = 0) is generally used for General Purpose ADC Conversion. This mode can be initialized by setting the ADCCON and ADCTSC and completed with a read the XPDATA (Normal ADC) value of ADCDAT0 (ADC Data Register 0).

2. Separate X/Y Position Conversion Mode
   Separate X/Y Position Conversion Mode is consist of two Conversion Modes: X-Position Mode and Y-Position Mode.
   The first mode is operated in the following way:
   X-Position Mode (AUTO_PST = 0 and XY_PST = 1) writes X-position conversion data to XPDATA of ADCDAT0 register. After conversion, The Touch Screen Interface generates the Interrupt source (INT_ADC) to Interrupt Controller.
   Y-Position Mode (AUTO_PST = 0 and XY_PST = 2) writes Y-position conversion data to YPDATA of ADCDAT1. After the conversion, the Touch Screen Interface also generates the Interrupt source (INT_ADC) to Interrupt Controller.

Table 16-1. Condition of Touch Screen Panel Pads in Separate X/Y Position Conversion Mode

<table>
<thead>
<tr>
<th></th>
<th>XP</th>
<th>XM</th>
<th>YP</th>
<th>YM</th>
</tr>
</thead>
</table>
3. Auto (Sequential) X/Y Position Conversion Mode.

Auto (Sequential) X/Y Position Conversion Mode (AUTO_PST = 1 and XY_PST = 0) is operated in the following way:

The Touch Screen Controller automatically converts X-position and Y-position. The Touch Screen Controller writes X-measurement data to XPDATA of ADCDAT0, and then writes Y-measurement data to YPDATA of ADCDAT1. After Auto (Sequential) Position Conversion, The Touch Screen Controller generates Interrupt source (INT_ADC) to Interrupt Controller.

Table 16-2. Condition of Touch Screen Panel Pads in Auto (Sequential) X/Y Position Conversion Mode.

<table>
<thead>
<tr>
<th></th>
<th>XP</th>
<th>XM</th>
<th>YP</th>
<th>YM</th>
</tr>
</thead>
</table>

4. Waiting for Interrupt Mode.

When Touch Screen Controller is in Waiting for Interrupt Mode, it waits for Stylus down. The controller generates Interrupt (INT_TC) signals when the Stylus is down on Touch Screen Panel. After an interrupt occurs, X and Y position can be read by the proper conversion mode (Separate X/Y position conversion Mode or Auto X/Y Position Conversion Mode).

Table 16-3. Condition of Touch Screen Panel Pads in Waiting for Interrupt Mode.

<table>
<thead>
<tr>
<th></th>
<th>XP</th>
<th>XM</th>
<th>YP</th>
<th>YM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waiting for Interrupt Mode</td>
<td>Pull-up</td>
<td>Hi-Z</td>
<td>AIN[5]</td>
<td>GND</td>
</tr>
</tbody>
</table>

Standby Mode

Standby mode is activated when STDBM of ADCCON register is set to ‘1’. In this mode, A/D conversion operation is halted and XPDATA (Normal ADC) of ADCDAT0 and YPDATA of ADCDAT1 contain the previous converted data.
Programming Notes

1. The A/D converted data can be accessed by means of interrupt or polling method. With interrupt method, the overall conversion time - from A/D converter start to converted data read - may be delayed because of the return time of interrupt service routine and data access time. With polling method, by checking the ADCCON[15] - end of conversion flag-bit, the read time from ADCDAT register can be determined.

2. A/D conversion can be activated in different way: After ADCCON[1] - A/D conversion start-by-read mode-is set to 1, A/D conversion starts simultaneously whenever converted data is read.

\[
\begin{align*}
A &= D \times \left(1/X\cdot\text{Tal Clock}\right) \quad \text{or} \quad A = D \times \left(1/\text{External Clock}\right) \\
B &= D \times \left(1/\text{PCLK}\right) \\
C &= D \times \left(1/\text{PCLK}\right) \\
D &= \text{DELAY value of ADCDLY Register}
\end{align*}
\]

Figure 16-3 Timing Diagram in Auto (Sequential) X/Y Position Conversion Mode
## ADC AND TOUCH SCREEN INTERFACE SPECIAL REGISTERS

### ADC CONTROL (ADCCON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCCON</td>
<td>0x58000000</td>
<td>R/W</td>
<td>ADC control register</td>
<td>0x3FC4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADCCON Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ECFLG [15]</strong></td>
<td>End of conversion flag (read only). 0 = A/D conversion in process 1 = End of A/D conversion</td>
<td>0</td>
</tr>
<tr>
<td><strong>PRSCEN [14]</strong></td>
<td>A/D converter prescaler enable. 0 = Disable 1 = Enable</td>
<td>0</td>
</tr>
<tr>
<td><strong>PRSCVL [13:6]</strong></td>
<td>A/D converter prescaler value. Data value: 1 ~ 255 Note that division factor is (N+1) when the prescaler value is N.</td>
<td>0xFF</td>
</tr>
<tr>
<td><strong>SEL_MUX [5:3]</strong></td>
<td>Analog input channel select. 000 = AIN 0 001 = AIN 1 010 = AIN 2 011 = AIN 3 100 = AIN 4 101 = AIN 5 110 = AIN 6 111 = AIN 7 (XP)</td>
<td>0</td>
</tr>
<tr>
<td><strong>STDBM [2]</strong></td>
<td>Standby mode select. 0 = Normal operation mode 1 = Standby mode</td>
<td>1</td>
</tr>
<tr>
<td><strong>READ_START [1]</strong></td>
<td>A/D conversion start by read. 0 = Disable start by read operation 1 = Enable start by read operation</td>
<td>0</td>
</tr>
<tr>
<td><strong>ENABLE_START [0]</strong></td>
<td>A/D conversion starts by setting this bit. If READ_START is enabled, this value is not valid. 0 = No operation 1 = A/D conversion starts and this bit is cleared after the start-up.</td>
<td>0</td>
</tr>
</tbody>
</table>
ADC TOUCH SCREEN CONTROL (ADCTSC) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCTSC</td>
<td>0x58000004</td>
<td>R/W</td>
<td>ADC touch screen control register</td>
<td>0x058</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADCTSC</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[8]</td>
<td>This bit should be zero.</td>
<td>0</td>
</tr>
<tr>
<td>YMSEN</td>
<td>[7]</td>
<td>Select output value of YMON.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = YMON output is 0 (YM = Hi-Z).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = YMON output is 1 (YM = GND).</td>
<td></td>
</tr>
<tr>
<td>YPSEN</td>
<td>[6]</td>
<td>Select output value of nYPON.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = nYPON output is 0 (YP = External voltage).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = nYPON output is 1 (YP is connected with AIN[5]).</td>
<td></td>
</tr>
<tr>
<td>XMSEN</td>
<td>[5]</td>
<td>Select output value of XMON.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = XMON output is 0 (XM = Hi-Z).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = XMON output is 1 (XM = GND).</td>
<td></td>
</tr>
<tr>
<td>XPSEN</td>
<td>[4]</td>
<td>Select output value of nXPON.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = nXPON output is 0 (XP = External voltage).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = nXPON output is 1 (XP is connected with AIN[7]).</td>
<td></td>
</tr>
<tr>
<td>PULL_UP</td>
<td>[3]</td>
<td>Pull-up switch enable.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = XP pull-up enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = XP pull-up disable</td>
<td></td>
</tr>
<tr>
<td>AUTO_PST</td>
<td>[2]</td>
<td>Automatically sequencing conversion of X-position and Y-position</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Normal ADC conversion</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Auto (Sequential) X/Y Position Conversion Mode</td>
<td></td>
</tr>
<tr>
<td>XY_PST</td>
<td>[1:0]</td>
<td>Manual measurement of X-position or Y-position.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = No operation mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = X-position measurement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = Y-position measurement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = Waiting for Interrupt Mode</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: In Auto mode, ADCTSC register should be reconfigured before starting read operation.
ADC START DELAY (ADCDLY) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCDLY</td>
<td>0x580000008</td>
<td>R/W</td>
<td>ADC start or interval delay register</td>
<td>0x00ff</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADCDLY Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELAY [15:0]</td>
<td>1) Normal Conversion Mode, Separate X/Y Position Conversion Mode, and Auto (Sequential) X/Y Position Conversion Mode. → X/Y Position Conversion Delay Value. 2) Waiting for Interrupt Mode. When Stylus down occurs in Waiting for Interrupt Mode, this register generates Interrupt signal (INT_TC) at intervals of several ms for Auto X/Y Position conversion. <strong>NOTE:</strong> Do not use Zero value (0x0000)</td>
<td>00ff</td>
</tr>
</tbody>
</table>

**NOTES:**
1. Before ADC conversion, Touch screen uses X-tal clock or EXTCLK (Waiting for Interrupt Mode).
2. During ADC conversion, PCLK is used.
## ADC CONVERSION DATA (ADCDAT0) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCDAT0</td>
<td>0x5800000C</td>
<td>R</td>
<td>ADC conversion data register</td>
<td>-</td>
</tr>
</tbody>
</table>

### Description of ADCDAT0 Bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
</table>
| UPDOWN[15] | Up or down state of Stylus at Waiting for Interrupt Mode.  
0 = Stylus down state  
1 = Stylus up state | -             |
0 = Normal ADC conversion  
1 = Sequencing measurement of X-position, Y-position | -             |
00 = No operation mode  
01 = X-position measurement  
10 = Y-position measurement  
11 = Waiting for Interrupt Mode | -             |
| Reserved[11:10] | Reserved | -             |
| XPDATA (Normal ADC)[9:0] | X-position conversion data value. (include Normal ADC conversion data value)  
Data value: 0 ~ 3FF | -             |
### ADC CONVERSION DATA (ADCDAT1) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCDAT1</td>
<td>0x58000010</td>
<td>R</td>
<td>ADC conversion data register</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADCDAT1 Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDOWN [15]</td>
<td>Up or down state of Stylus at Waiting for Interrupt Mode.</td>
<td>-</td>
</tr>
<tr>
<td>AUTO_PST [14]</td>
<td>Automatically sequencing conversion of X-position and Y-position.</td>
<td>-</td>
</tr>
<tr>
<td>XY_PST [13:12]</td>
<td>Manual measurement of X-position or Y-position.</td>
<td>-</td>
</tr>
<tr>
<td>Reserved [11:10]</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>YPDATA [9:0]</td>
<td>Y-position conversion data value</td>
<td>-</td>
</tr>
</tbody>
</table>

Data value: 0 ~ 3FF
REAL TIME CLOCK (RTC)

OVERVIEW

The Real Time Clock (RTC) unit can be operated by the backup battery while the system power is off. The RTC can transmit 8-bit data to CPU as Binary Coded Decimal (BCD) values using the STRB/LDRB ARM operation. The data include the time by second, minute, hour, date, day, month, and year. The RTC unit works with an external 32.768 kHz crystal and also can perform the alarm function.

FEATURES

— BCD number: second, minute, hour, date, day, month, and year
— Leap year generator
— Alarm function: alarm interrupt or wake-up from power-off mode
— Year 2000 problem is removed.
— Independent power pin (RTCVDD)
— Supports millisecond tick time interrupt for RTOS kernel time tick.
— Round reset function
REAL TIME CLOCK OPERATION

Figure 17-1. Real Time Clock Block Diagram

LEAP YEAR GENERATOR

The leap year generator can determine the last date of each month out of 28, 29, 30, or 31, based on data from BCDDATE, BCDMON, and BCDYEAR. This block considers leap year in deciding on the last date. An 8-bit counter can only represent 2 BCD digits, so it cannot decide whether “00” year (the year with its last two digits zeros) is a leap year or not. For example, it cannot discriminate between 1900 and 2000. To solve this problem, the RTC block in S3C2410X has hard-wired logic to support the leap year in 2000. Note 1900 is not leap year while 2000 is leap year. Therefore, two digits of 00 in S3C2410X denote 2000, not 1900.
READ/WRITE REGISTERS

Bit 0 of the RTCCON register must be set high in order to write the BCD register in RTC block. To display the second, minute, hour, date, month, and year, the CPU should read the data in BCDSEC, BCDMIN, BCDHOUR, BCDDAY, BCDDATE, BCDMON, and BCDYEAR registers, respectively, in the RTC block. However, a one second deviation may exist because multiple registers are read. For example, when the user reads the registers from BCDYEAR to BCDMIN, the result is assumed to be 2059 (Year), 12 (Month), 31 (Date), 23 (Hour) and 59 (Minute). When the user read the BCDSEC register and the value ranges from 1 to 59 (Second), there is no problem, but, if the value is 0 sec., the year, month, date, hour, and minute may be changed to 2060 (Year), 1 (Month), 1 (Date), 0 (Hour) and 0 (Minute) because of the one second deviation that was mentioned. In this case, the user should re-read from BCDYEAR to BCDSEC if BCDSEC is zero.

BACKUP BATTERY OPERATION

The RTC logic can be driven by the backup battery, which supplies the power through the RTCVDD pin into the RTC block, even if the system power is off. When the system is off, the interfaces of the CPU and RTC logic should be blocked, and the backup battery only drives the oscillation circuit and the BCD counters to minimize power dissipation.

ALARM FUNCTION

The RTC generates an alarm signal at a specified time in the power-off mode or normal operation mode. In normal operation mode, the alarm interrupt (ALMINT) is activated. In the power-off mode, the power management wakeup (PMWKUP) signal is activated as well as the ALMINT. The RTC alarm register (RTCALM) determines the alarm enable/disable status and the condition of the alarm time setting.

TICK TIME INTERRUPT

The RTC tick time is used for interrupt request. The TICNT register has an interrupt enable bit and the count value for the interrupt. The count value reaches '0' when the tick time interrupt occurs. Then the period of interrupt is as follows:

\[
\text{Period} = \frac{n+1}{128} \text{second}
\]

n: Tick time count value (1~127)

This RTC time tick may be used for real time operating system (RTOS) kernel time tick. If time tick is generated by the RTC time tick, the time related function of RTOS will always synchronized in real time.

ROUND RESET FUNCTION

The round reset function can be performed by the RTC round reset register (RTCRST). The round boundary (30, 40, or 50 sec.) of the second carry generation can be selected, and the second value is rounded to zero in the round reset. For example, when the current time is 23:37:47 and the round boundary is selected to 40 sec, the round reset changes the current time to 23:38:00.

NOTE

All RTC registers have to be accessed for each byte unit using the STRB and LDRB instructions or char type pointer.
32.768KHZ X-TAL CONNECTION EXAMPLE

The Figure 17-2 shows a circuit of the RTC unit oscillation at 32.768 kHz.

Figure 17-2. Main Oscillator Circuit Example
REAL TIME CLOCK SPECIAL REGISTERS

REAL TIME CLOCK CONTROL (RTCCON) REGISTER

The RTCCON register consists of 4 bits such as the RTCEN, which controls the read/write enable of the BCD registers, CLKSEL, CNTSEL, and CLKRST for testing.

RTCEN bit can control all interfaces between the CPU and the RTC, so it should be set to 1 in an RTC control routine to enable data read/write after a system reset. Also before power off, the RTCEN bit should be cleared to 0 to prevent inadvertent writing into RTC registers.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTCCON</td>
<td>0x57000040(L) 0x57000043(B)</td>
<td>R/W (by byte)</td>
<td>RTC control register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RTCCON Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKRST [3]</td>
<td>RTC clock count reset.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = No reset, 1 = Reset</td>
<td></td>
</tr>
<tr>
<td>CNTSEL [2]</td>
<td>BCD count select.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = Merge BCD counters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Reserved (Separate BCD counters)</td>
<td></td>
</tr>
<tr>
<td>CLKSEL [1]</td>
<td>BCD clock select.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = XTAL 1/2^15 divided clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Reserved (XTAL clock only for test)</td>
<td></td>
</tr>
<tr>
<td>RTCEN [0]</td>
<td>RTC control enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: Only BCD time count and read operation can be performed.

NOTES:
1. All RTC registers have to be accessed for each byte unit using STRB and LDRB instructions or char type pointer.
2. (L): Little endian.
   (B): Big endian.

TICK TIME COUNT (TICNT) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICNT</td>
<td>0x57000044(L) 0x57000047(B)</td>
<td>R/W (by byte)</td>
<td>Tick time count register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TICNT</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICNT</td>
<td>[7]</td>
<td>Tick time interrupt enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>TICNT</td>
<td>[6:0]</td>
<td>Tick time count value (1~127).</td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This counter value decreases internally, and users cannot read this counter value in working.</td>
<td></td>
</tr>
</tbody>
</table>
RTC ALARM CONTROL (RTCALM) REGISTER

The RTCALM register determines the alarm enable and the alarm time. Note that the RTCALM register generates the alarm signal through both ALMINT and PMWKUP in power down mode, but only through ALMINT in the normal operation mode.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTCALM</td>
<td>0x57000050(L)</td>
<td>R/W</td>
<td>RTC alarm control register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x57000053(B)</td>
<td>(by byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RTCALM</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7]</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ALMEN</td>
<td>[6]</td>
<td>Alarm global enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>YEAREN</td>
<td>[5]</td>
<td>Year alarm enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>MONREN</td>
<td>[4]</td>
<td>Month alarm enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>DATEEN</td>
<td>[3]</td>
<td>Date alarm enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>HOUREN</td>
<td>[2]</td>
<td>Hour alarm enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>MINEN</td>
<td>[1]</td>
<td>Minute alarm enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>SECEN</td>
<td>[0]</td>
<td>Second alarm enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
</tbody>
</table>
### ALARM SECOND DATA (ALMSEC) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMSEC</td>
<td>0x57000054(L)</td>
<td>R/W (by byte)</td>
<td>Alarm second data register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x57000057(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALMSEC Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7]</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td>[6:4]</td>
<td>SECDATA BCD value for alarm second.</td>
<td>000</td>
</tr>
<tr>
<td>[3:0]</td>
<td>0 ~ 9</td>
<td>0000</td>
</tr>
</tbody>
</table>

### ALARM MIN DATA (ALMMIN) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMMIN</td>
<td>0x57000058(L)</td>
<td>R/W (by byte)</td>
<td>Alarm minute data register</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x5700005B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALMMIN Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7]</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td>[6:4]</td>
<td>MINDATA BCD value for alarm minute.</td>
<td>000</td>
</tr>
<tr>
<td>[3:0]</td>
<td>0 ~ 9</td>
<td>0000</td>
</tr>
</tbody>
</table>

### ALARM HOUR DATA (ALMHOURL) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMHOURL</td>
<td>0x5700005C(L)</td>
<td>R/W (by byte)</td>
<td>Alarm hour data register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x5700005F(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALMHOURL Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:6]</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>[5:4]</td>
<td>HOURDATA BCD value for alarm hour.</td>
<td>00</td>
</tr>
<tr>
<td>[3:0]</td>
<td>0 ~ 9</td>
<td>0000</td>
</tr>
</tbody>
</table>
# REAL TIME CLOCK (RTC) S3C2410X

## ALARM DATE DATA (ALMDATE) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMDATE</td>
<td>0x57000060(L)</td>
<td>R/W (by byte)</td>
<td>Alarm date data register</td>
<td>0x01</td>
</tr>
<tr>
<td></td>
<td>0x57000063(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:6]</td>
<td>00</td>
</tr>
<tr>
<td>DATEDATA</td>
<td>[5:4]</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>[3:0]</td>
<td>0001</td>
</tr>
</tbody>
</table>

## ALARM MON DATA (ALMMON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMMON</td>
<td>0x57000064(L)</td>
<td>R/W</td>
<td>Alarm month data register</td>
<td>0x01</td>
</tr>
<tr>
<td></td>
<td>0x57000067(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:5]</td>
<td>00</td>
</tr>
<tr>
<td>MONDATA</td>
<td>[4]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>[3:0]</td>
<td>0001</td>
</tr>
</tbody>
</table>

## ALARM YEAR DATA (ALMYEAR) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMYEAR</td>
<td>0x57000068(L)</td>
<td>R/W</td>
<td>Alarm year data register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x5700006B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEARDATA</td>
<td>[7:0]</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:6]</td>
<td>00</td>
</tr>
<tr>
<td>DATEDATA</td>
<td>[5:4]</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>[3:0]</td>
<td>0001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:5]</td>
<td>00</td>
</tr>
<tr>
<td>MONDATA</td>
<td>[4]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>[3:0]</td>
<td>0001</td>
</tr>
</tbody>
</table>
**RTC ROUND RESET (RTCRST) REGISTER**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTCRST</td>
<td>0x5700006C(L)</td>
<td>R/W</td>
<td>RTC round reset register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x5700006F(B)</td>
<td>(by byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RTCRST</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRSTEN</td>
<td>[3]</td>
<td>Round second reset enable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Disable, 1 = Enable</td>
<td></td>
</tr>
<tr>
<td>SECCR</td>
<td>[2:0]</td>
<td>Round boundary for second carry generation.</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>011 = over than 30 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>100 = over than 40 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>101 = over than 50 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NOTE:</strong> If other values (0, 1, 2, 6, or 7) are set, no second carry is generated. But second value can be reset.</td>
<td></td>
</tr>
</tbody>
</table>

**BCD SECOND (BCDSEC) REGISTER**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDSEC</td>
<td>0x57000070(L)</td>
<td>R/W</td>
<td>BCD second register</td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td>0x57000073(B)</td>
<td>(by byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCDSEC</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECDATA</td>
<td>[6:4]</td>
<td>BCD value for second.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 ~ 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3:0]</td>
<td>0 ~ 9</td>
<td></td>
</tr>
</tbody>
</table>

**BCD MINUTE (BCDMIN) REGISTER**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDMIN</td>
<td>0x57000074(L)</td>
<td>R/W</td>
<td>BCD minute register</td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td>0x57000077(B)</td>
<td>(by byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCDMIN</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINDATA</td>
<td>[6:4]</td>
<td>BCD value for minute.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 ~ 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3:0]</td>
<td>0 ~ 9</td>
<td></td>
</tr>
</tbody>
</table>
# BCD HOUR (BCDHOUR) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDHOUR</td>
<td>0x57000078(L)</td>
<td>R/W</td>
<td>BCD hour register</td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td>0x5700007B(B)</td>
<td>(by byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCDHOUR Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7:6]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOURDATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5:4]</td>
<td>BCD value for hour.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0 ~ 2</td>
<td></td>
</tr>
<tr>
<td>[3:0]</td>
<td>0 ~ 9</td>
<td></td>
</tr>
</tbody>
</table>

# BCD DATE (BCDDATE) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDDATE</td>
<td>0x5700007C(L)</td>
<td>R/W</td>
<td>BCD date register</td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td>0x5700007F(B)</td>
<td>(by byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCDDATE Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7:6]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATEDATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5:4]</td>
<td>BCD value for date.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>0 ~ 3</td>
<td></td>
</tr>
<tr>
<td>[3:0]</td>
<td>0 ~ 9</td>
<td></td>
</tr>
</tbody>
</table>

# BCD DAY (BCDDAY) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDDAY</td>
<td>0x57000080(L)</td>
<td>R/W</td>
<td>BCD a day of the week register</td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td>0x57000083(B)</td>
<td>(by byte)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCDDAY Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7:3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAYDATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2:0]</td>
<td>BCD value for a day of the week.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1 ~ 7</td>
<td></td>
</tr>
</tbody>
</table>
## BCD MONTH (BCDMON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDMON</td>
<td>0x57000084(L)</td>
<td>R/W (by byte)</td>
<td>BCD month register</td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td>0x57000087(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCDMON Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:5]</td>
<td></td>
</tr>
<tr>
<td>MONDATA</td>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 ~ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3:0]</td>
<td>0 ~ 9</td>
</tr>
</tbody>
</table>

## BCD YEAR (BCDYEAR) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCDYEAR</td>
<td>0x57000088(L)</td>
<td>R/W (by byte)</td>
<td>BCD year register</td>
<td>Undefined</td>
</tr>
<tr>
<td></td>
<td>0x5700008B(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCDYEAR Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEARDATA</td>
<td>[7:0]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00 ~ 99</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCD value for year.</td>
<td>-</td>
</tr>
</tbody>
</table>
OVERVIEW

The S3C2410X watchdog timer is used to resume the controller operation whenever it is disturbed by malfunctions such as noise and system errors. It can be used as a normal 16-bit interval timer to request interrupt service. The watchdog timer generates the reset signal for 128 PCLK cycles.

FEATURES

— Normal interval timer mode with interrupt request
— Internal reset signal is activated for 128 PCLK cycles when the timer count value reaches 0 (time-out).
WATCHDOG TIMER OPERATION

Figure 18-1 shows the functional block diagram of the watchdog timer. The watchdog timer uses only PCLK as its source clock. The PCLK frequency is prescaled to generate the corresponding watchdog timer clock, and the resulting frequency is divided again.

![Watchdog Timer Block Diagram](image)

The prescaler value and the frequency division factor are specified in the watchdog timer control (WTCON) register. Valid prescaler values range from 0 to $2^8 - 1$. The frequency division factor can be selected as 16, 32, 64, or 128.

Use the following equation to calculate the watchdog timer clock frequency and the duration of each timer clock cycle:

$$t_{\text{watchdog}} = \frac{1}{PCLK \times \frac{\text{Prescaler value} + 1}{\text{Division factor}}}$$

WTDAT & WTCNT

Once the watchdog timer is enabled, the value of watchdog timer data (WTDAT) register cannot be automatically reloaded into the timer counter (WTCNT). In this reason, an initial value must be written to the watchdog timer count (WTCNT) register, before the watchdog timer starts.

CONSIDERATION OF DEBUGGING ENVIRONMENT

When the S3C2410X is in debug mode using Embedded ICE, the watchdog timer must not operate.

The watchdog timer can determine whether or not it is currently in the debug mode from the CPU core signal (DBGACK signal). Once the DBGACK signal is asserted, the reset output of the watchdog timer is not activated as the watchdog timer is expired.
WATCHDOG TIMER SPECIAL REGISTERS

WATCHDOG TIMER CONTROL (WTCON) REGISTER

The WTCON register allows the user to enable/disable the watchdog timer, select the clock signal from 4 different sources, enable/disable interrupts, and enable/disable the watchdog timer output. The Watchdog timer is used to resume the S3C2410X restart on mal-function after its power on; if controller restart is not desired, the Watchdog timer should be disabled.

If the user wants to use the normal timer provided by the Watchdog timer, enable the interrupt and disable the Watchdog timer.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTCON</td>
<td>0x53000000</td>
<td>R/W</td>
<td>Watchdog timer control register</td>
<td>0x8021</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WTCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescaler Value</td>
<td>[15:8]</td>
<td>Prescaler value. The valid range is from 0 to (2^8-1).</td>
<td>0x80</td>
</tr>
<tr>
<td>Reserved</td>
<td>[7:6]</td>
<td>Reserved. These two bits must be 00 in normal operation.</td>
<td>00</td>
</tr>
<tr>
<td>Watchdog Timer</td>
<td>[5]</td>
<td>Enable or disable bit of Watchdog timer. 0 = Disable 1 = Enable</td>
<td>1</td>
</tr>
<tr>
<td>Clock Select</td>
<td>[4:3]</td>
<td>Determine the clock division factor. 00: 16 01: 32 10: 64 11: 128</td>
<td>00</td>
</tr>
<tr>
<td>Interrupt Generation</td>
<td>[2]</td>
<td>Enable or disable bit of the interrupt. 0 = Disable 1 = Enable</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[1]</td>
<td>Reserved. This bit must be 0 in normal operation.</td>
<td>0</td>
</tr>
<tr>
<td>Reset Enable/Disable</td>
<td>[0]</td>
<td>Enable or disable bit of Watchdog timer output for reset signal. 1: Assert reset signal of the S3C2410X at watchdog time-out 0: Disable the reset function of the watchdog timer.</td>
<td>1</td>
</tr>
</tbody>
</table>
WATCHDOG TIMER DATA (WTDAT) REGISTER

The WTDAT register is used to specify the time-out duration. The content of WTDAT cannot be automatically loaded into the timer counter at initial watchdog timer operation. However, using 0x8000 (initial value) will drive the first time-out. In this case, the value of WTDAT will be automatically reloaded into WTCNT.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTDAT</td>
<td>0x53000004</td>
<td>R/W</td>
<td>Watchdog timer data register</td>
<td>0x8000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WTDAT Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count Reload Value</td>
<td>Watchdog timer count value for reload.</td>
<td>0x8000</td>
</tr>
</tbody>
</table>

WATCHDOG TIMER COUNT (WTCNT) REGISTER

The WTCNT register contains the current count values for the watchdog timer during normal operation. Note that the content of the WTDAT register cannot be automatically loaded into the timer count register when the watchdog timer is enabled initially, so the WTCNT register must be set to an initial value before enabling it.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTCNT</td>
<td>0x53000008</td>
<td>R/W</td>
<td>Watchdog timer count register</td>
<td>0x8000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WTCNT Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count Value</td>
<td>The current count value of the watchdog timer</td>
<td>0x8000</td>
</tr>
</tbody>
</table>
SECURE DIGITAL INTERFACE FOR SDIO(SDI)

OVERVIEW

The S3C2410X Secure Digital Interface (SDI) can interface for SD memory card, SDIO device and Multi-Media Card (MMC).

FEATURES

— SD Memory Card Spec. (ver. 1.0) / MMC Spec. (2.11) compatible
— SDIO Card Spec (ver. 1.0) compatible
— 16 words (64 bytes) FIFO (depth 16) for data Tx/Rx
— 40-bit Command Register (SDICARG[31:0]+SDICCON[7:0])
— 136-bit Response Register (SDIRSPn[127:0]+SDICSTA[7:0])
— 8-bit Prescaler logic (Freq. = System Clock / (2(P + 1)))
— CRC7 & CRC16 Generator
— Normal, and DMA Data Transfer Mode (byte or word transfer)
— 1bit / 4bit (wide bus) Mode & Block / Stream Mode Switch support
— Supports up to 25 MHz in data transfer mode for SDI
— Supports up to 10 MHz in data transfer mode for MMC
Figure 19-1. Block Diagram
SDI OPERATION

A serial clock line is synchronized with the five data lines for shifting and sampling of the information. Making the appropriate bit settings to the SDIPRE register depends on the transmission frequency. You can modify its frequency to adjust the baud rate data register value.

Programming Procedure (common)

SDI modules can be programmed, following these basic steps:

1. Set SDICON to configure properly with clock and interrupt.
2. Set SDIPRE to configure with a proper value.
3. Wait 74 SDCLK clock cycle in order to initialize the card.

CMD Path Programming

1. Write command argument (32-bit) to SDICARG register.
2. Determine command types and start command by setting SDICCON[8].
3. Confirm the end of SDI command operation when the specific flag of SDICSTA is set.
   - If the type of command is no-response, the flag is SDICSTA[11].
   - If the type of command is with-response, the flag is SDICSTA[9].
4. Clear the corresponding flag of the SDICSTA register by writing one to the flag bit.

DAT Path Programming

1. Write timeout period to SDITIMER register.
2. Write block size (block length) to SDIBSIZE register (normally 0x200 byte).
3. Determine the mode of block, wide bus, DMA, etc. and start data transfer with setting SDIDCON register.
4. Write Tx-data to SDIDAT register while Tx FIFO is available by checking SDIFSTA (available, half or empty) register.
5. Read Rx-data from SDIDAT register while Rx FIFO is available by checking SDIFSTA (available, half or be last data) register.
6. Confirm the end of SDI data operation when the flag of data transfer finish (SDIDSTA[4]) is set.
7. Clear the corresponding flag of SDIDSTA register by writing one to the flag bit.

NOTES

1. In case of MMC, the maximum clock for data transfer is 10 MHz.
2. In MMC write mode, the CRC error is occurred in spite of correct writing. If user wants to transmit data reliably, read the data and compare it after writing.
3. In case of long response, the CRC error should be detected after receiving exact response data from SD device. User should check the CRC of received response by software.
SDIO OPERATION

There are two functions of the SDIO operation: SDIO Interrupt receiving and Read Wait Request generation. These two functions can operate when RcvIOInt bit and RwaitEn bit of SDICON register is activated respectively. Detailed steps and conditions for the two functions are described below.

SDIO Interrupt

In SD 1bit mode, the interrupt is received through all ranges from SDDAT1 pin.

In SD 4bit mode, SDDAT1 pin is shared between to receive data and interrupts. When interrupt detection ranges (Interrupt Period) are:

1. Single Block: the time between A and B
   - A: 2clocks after the completion of a data packet
   - B: The completion of sending the end bit of the next with-data command

2. Multi Block, SDIDCON[21] = 0: the time between A and B, restart interrupt detection range at C
   - A: 2clocks after the completion of a data packet
   - B: 2clocks after A
   - C: 2clocks after the end bit of the abort command response

3. Multi Block, SDIDCON[21] = 1: the time between A and B, restart at A
   - A: 2clocks after the completion of a data packet
   - B: 2clocks after A

In case of last block, interrupt period begins at last A, but it does not end at B (CMD53 case).

Read Wait Request

Regardless of 1bit or 4bit mode, Read Wait Request signal transmits to SDDAT2 pin in the condition below.

- In read multiple operation, request signal transmission begins at 2clocks after the end of the data block.
- Transmission ends when the user writes one to SDIDSTA[10].
### SDI SPECIAL REGISTERS

#### SDI Control (SDICON) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDICON</td>
<td>0x5A000000</td>
<td>R/W</td>
<td>SDI control register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

#### SDICON Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>Determine byte order type when you read (write) data from (to)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SD host FIFO with word boundary.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Type A,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Type B</td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>Determine whether SD host receives SDIO Interrupt from the card or not</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(for SDIO).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = ignore,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = receive SDIO Interrupt</td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>Determine read wait request signal generate when SD host</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>waits the next block in multiple block read mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This bit needs to delay the next block to be transmitted from the card</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(for SDIO).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = disable (no generate),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Read wait enable (use SDIO)</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>Reset FIFO value. This bit is automatically cleared.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = normal mode,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = FIFO reset</td>
<td></td>
</tr>
<tr>
<td>[0]</td>
<td>Determine whether SDCLK Out enable or not.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 = disable (prescaler off),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = clock enable</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:**
- Byte Order Type

#### SDI Baud Rate Prescaler (SDIPRE) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIPRE</td>
<td>0x5A000004</td>
<td>R/W</td>
<td>SDI baud rate prescaler register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

#### SDIPRE Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:0]</td>
<td>Determine SDI clock (SDCLK) rate as above equation.</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>Baud rate = PCLK / 2 / (Prescaler value + 1)</td>
<td></td>
</tr>
</tbody>
</table>
### SDI Command Argument Register (SDICARG)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDICARG</td>
<td>0x5A000008</td>
<td>R/W</td>
<td>SDI command argument register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDICARG</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CmdArg</td>
<td>[31:0]</td>
<td>Command Argument</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### SDI Command Control (SDICCON) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDICCON</td>
<td>0x5A00000C</td>
<td>R/W</td>
<td>SDI command control register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDICCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort Cmd (AbortCmd)</td>
<td>[12]</td>
<td>Determine whether command type is for abort (for SDIO). 0 = normal command, 1 = abort command (CMD12, CMD52)</td>
<td>0</td>
</tr>
<tr>
<td>Command with Data (WithData)</td>
<td>[11]</td>
<td>Determine whether command type is with data (for SDIO). 0 = without data, 1 = with data</td>
<td>0</td>
</tr>
<tr>
<td>LongRsp</td>
<td>[10]</td>
<td>Determine whether host receives a 136-bit long response or not. 0 = short response, 1 = long response</td>
<td>0</td>
</tr>
<tr>
<td>WaitRsp</td>
<td>[9]</td>
<td>Determine whether host waits for a response or not. 0 = no response, 1 = wait response</td>
<td>0</td>
</tr>
<tr>
<td>Command Start(CMST)</td>
<td>[8]</td>
<td>Determine whether command operation starts or not. 0 = command ready, 1 = command start</td>
<td>0</td>
</tr>
<tr>
<td>CmdIndex</td>
<td>[7:0]</td>
<td>Command index with start 2bit (8bit)</td>
<td>0x00</td>
</tr>
</tbody>
</table>
### SDI Command Status (SDICSTA) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDICSTA</td>
<td>0x5A000010</td>
<td>R/(W)</td>
<td>SDI command status register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDICSTA</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response CRC</td>
<td>[12] R/W</td>
<td>CRC check failed when command response received. This flag is cleared by setting one to this bit. 0 = not detect, 1 = crc fail</td>
<td>0</td>
</tr>
<tr>
<td>Fail(RspCrc)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command Sent</td>
<td>[11] R/W</td>
<td>Command sent (not concerned with response). This flag is cleared by setting one to this bit. 0 = not detect, 1 = command end</td>
<td>0</td>
</tr>
<tr>
<td>(CmdSent)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command Time</td>
<td>[10] R/W</td>
<td>Command response timeout (64clk). This flag is cleared by setting one to this bit. 0 = not detect, 1 = timeout</td>
<td>0</td>
</tr>
<tr>
<td>Out (CmdTout)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>[9] R/W</td>
<td>Command response received. This flag is cleared by setting one to this bit. 0 = not detect, 1 = response end</td>
<td>0</td>
</tr>
<tr>
<td>Receive</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End (RspFin)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMD line</td>
<td>[8] R</td>
<td>Command transfer in progress. 0 = not detect, 1 = in progress</td>
<td>0</td>
</tr>
<tr>
<td>progress</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On (CmdOn)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RspIndex</td>
<td>[7:0] R</td>
<td>Response index 6bit with start 2bit (8bit)</td>
<td>0x00</td>
</tr>
</tbody>
</table>

---

*Note: The register addresses and descriptions are based on the S3C2410X Secure Digital Interface for SDIO (SDI) specifications.*
### SDI Response Register 0 (SDIRSP0)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIRSP0</td>
<td>0x5A000014</td>
<td>R</td>
<td>SDI response register 0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIRSP0 Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response0</td>
<td>Card status<a href="short">31:0</a>, card status<a href="long">127:96</a></td>
<td>0x000000000</td>
</tr>
</tbody>
</table>

### SDI Response Register 1 (SDIRSP1)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIRSP1</td>
<td>0x5A000018</td>
<td>R</td>
<td>SDI response register 1</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIRSP1 Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCRC7</td>
<td>CRC7 (with end bit, short), card status<a href="long">95:88</a></td>
<td>0x00</td>
</tr>
<tr>
<td>Response1</td>
<td>Unused (short), card status<a href="long">87:64</a></td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### SDI Response Register 2 (SDIRSP2)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIRSP2</td>
<td>0x5A00001C</td>
<td>R</td>
<td>SDI response register 2</td>
<td>0x000y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIRSP2 Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response2</td>
<td>Unused (short), card status<a href="long">63:32</a></td>
<td>0x000000000</td>
</tr>
</tbody>
</table>

### SDI Response Register 3 (SDIRSP3)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIRSP3</td>
<td>0x5A000020</td>
<td>R</td>
<td>SDI response register 3</td>
<td>0x0y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIRSP3 Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response3</td>
<td>Unused (short), card status<a href="long">31:0</a></td>
<td>0x000000000</td>
</tr>
</tbody>
</table>
### SDI Data / Busy Timer (SDIDTIMER) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIDTIMER</td>
<td>0x5A000024</td>
<td>R/W</td>
<td>SDI data / busy timer register</td>
<td>0x2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIDTIMER</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTimer</td>
<td>[15:0]</td>
<td>Data / busy timeout period (0~65535 cycle)</td>
<td>0x2000</td>
</tr>
</tbody>
</table>

### SDI Block Size (SDIBSIZE) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIBSIZE</td>
<td>0x5A000028</td>
<td>R/W</td>
<td>SDI block size register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIBSIZE</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlkSize</td>
<td>[11:0]</td>
<td>Block size value (0~4095 byte). Do not care when stream mode</td>
<td>0x000</td>
</tr>
</tbody>
</table>

**NOTE:** In Case of multi block, BlkSize must be aligned to word(4byte) size.(BlkSize[1:0] = 00)
## SDI Data Control (SDIDCON) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIDCON</td>
<td>0x5A00002C</td>
<td>R/W</td>
<td>SDI data control register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIDCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIO Interrupt</td>
<td>[21]</td>
<td>Period Type (PrdType) determines whether SDIO Interrupt period is 2 cycle or</td>
<td>0</td>
</tr>
<tr>
<td>Period Type (PrdType)</td>
<td></td>
<td>extend more cycle when last data block is transferred (for SDIO). 0 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>exactly 2 cycle, 1 = more cycle (likely single block)</td>
<td></td>
</tr>
<tr>
<td>Transmit After</td>
<td>[20]</td>
<td>Response (TARSP) determines when data transmit start after response receive</td>
<td>0</td>
</tr>
<tr>
<td>Response (TARSP)</td>
<td></td>
<td>or not. 0 = directly after DatMode set, 1 = after response</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>receive (assume DatMode sets to 2'b11)</td>
<td></td>
</tr>
<tr>
<td>Receive After</td>
<td>[19]</td>
<td>Command (RACMD) determines when data receive start after command sent or</td>
<td>0</td>
</tr>
<tr>
<td>Command (RACMD)</td>
<td></td>
<td>not. 0 = directly after DatMode set, 1 = after command sent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(assume DatMode sets to 2'b10)</td>
<td></td>
</tr>
<tr>
<td>Busy After</td>
<td>[18]</td>
<td>Command (BACMD) determines when busy receive start after command sent or</td>
<td>0</td>
</tr>
<tr>
<td>Command (BACMD)</td>
<td></td>
<td>not. 0 = directly after DatMode set, 1 = after command sent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(assume DatMode sets to 2'b01)</td>
<td></td>
</tr>
<tr>
<td>Block mode</td>
<td>[17]</td>
<td>(BlkMode) determines the direction of data transfer. 0 = ready, 1 = block</td>
<td>0</td>
</tr>
<tr>
<td>(BlkMode)</td>
<td></td>
<td>data transfer, 1 = block data transfer</td>
<td></td>
</tr>
<tr>
<td>Wide bus enable</td>
<td>[16]</td>
<td>(WideBus) determines enable wide bus mode. 0 = standard bus mode (only</td>
<td>0</td>
</tr>
<tr>
<td>(WideBus)</td>
<td></td>
<td>SDIDAT[0] used; 1 = wide bus mode (SDIDAT[3:0] used)</td>
<td></td>
</tr>
<tr>
<td>DMA Enable</td>
<td>[15]</td>
<td>(EnDMA) determines whether DMA is enabled. 0 = disable (polling), 1 =</td>
<td>0</td>
</tr>
<tr>
<td>(EnDMA)</td>
<td></td>
<td>dma enable</td>
<td></td>
</tr>
<tr>
<td>Stop by force</td>
<td>[14]</td>
<td>(STOP) determines whether data transfer stop by force or not. 0 = normal,</td>
<td>0</td>
</tr>
<tr>
<td>(STOP)</td>
<td></td>
<td>1 = stop by force</td>
<td></td>
</tr>
<tr>
<td>Data Transfer</td>
<td>[13:12]</td>
<td>Mode (DatMode) determines the direction of data transfer. 00 = ready,</td>
<td>00</td>
</tr>
<tr>
<td>Mode (DatMode)</td>
<td></td>
<td>01 = only busy check start, 10 = data receive start, 11 = data transmit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>start</td>
<td></td>
</tr>
<tr>
<td>BlkNum</td>
<td>[11:0]</td>
<td>Block Number (0~4095). Do not care when stream mode.</td>
<td>0x00</td>
</tr>
</tbody>
</table>

**NOTES:**
1. If you want one of TARSP, RACMD and BACMD bits (SDIDCON[20:18]) to "1", you need to write on SDIDCON register head of on SDICCON register (always need for SDIO).
2. When DMA operation is completed, DMA Enable[15] bit of SDIDCON register should be disabled.
SDI Data Remain Counter (SDIDCNT) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIDCNT</td>
<td>0x5A000030</td>
<td>R</td>
<td>SDI data remain counter register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIDCNT</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlkNumCnt</td>
<td>[23:12]</td>
<td>Remaining block number</td>
<td>0x000</td>
</tr>
<tr>
<td>BlkCnt</td>
<td>[11:0]</td>
<td>Remaining data byte of 1 block</td>
<td>0x000</td>
</tr>
</tbody>
</table>
SDI Data Status (SDIDSTA) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIDSTA</td>
<td>0x5A000034</td>
<td>R/(W)</td>
<td>SDI data status register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIDSTA</th>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Wait Request Occur (RWaitReq)</td>
<td>[10]</td>
<td>R/W</td>
<td>Read wait request signal transmits to SD card. The request signal is stopped and this flag is cleared by setting one to this bit. (for SDIO)&lt;br&gt;0 = not occur, 1 = Read wait request occur</td>
</tr>
<tr>
<td>SDIO Interrupt Detect (IOIntDet)</td>
<td>[9]</td>
<td>R/W</td>
<td>SDIO interrupt detects. This flag is cleared by setting one to this bit. (for SDIO)&lt;br&gt;0 = not detect, 1 = SDIO interrupt detect</td>
</tr>
<tr>
<td>FIFO Fail error (FFfail)</td>
<td>[8]</td>
<td>R/W</td>
<td>FIFO fail error when FIFO occurs overrun / underrun / misaligned data saving. This flag is cleared by setting one to this bit.&lt;br&gt;0 = not detect, 1 = FIFO fail</td>
</tr>
<tr>
<td>CRC Status Fail (CrcSta)</td>
<td>[7]</td>
<td>R/W</td>
<td>CRC Status error when data block sent (CRC check failed - returned from card). This flag is cleared by setting one to this bit.&lt;br&gt;0 = not detect, 1 = crc status fail</td>
</tr>
<tr>
<td>Data Receive CRC Fail (DatCrc)</td>
<td>[6]</td>
<td>R/W</td>
<td>Data block received error (CRC check failed - calculated by host). This flag is cleared by setting one to this bit.&lt;br&gt;0 = not detect, 1 = receive crc fail</td>
</tr>
<tr>
<td>Data Time Out (DatTout)</td>
<td>[5]</td>
<td>R/W</td>
<td>Data / Busy receive timeout. This flag is cleared by setting one to this bit.&lt;br&gt;0 = not detect, 1 = timeout</td>
</tr>
<tr>
<td>Data Transfer Finish (DatFin)</td>
<td>[4]</td>
<td>R/W</td>
<td>Data transfer completes (data counter is zero). This flag is cleared by setting one to this bit.&lt;br&gt;0 = not detect, 1 = data finish detect</td>
</tr>
<tr>
<td>Busy Finish (BusyFin)</td>
<td>[3]</td>
<td>R/W</td>
<td>Only busy check finish. This flag is cleared by setting one to this bit.&lt;br&gt;0 = not detect, 1 = busy finish detect</td>
</tr>
<tr>
<td>Start Bit Error (SbitErr)</td>
<td>[2]</td>
<td>R/W</td>
<td>Start bit is not detected on all data signals in wide bus mode. This flag is cleared by setting one to this bit.&lt;br&gt;0 = not detect, 1 = command end</td>
</tr>
<tr>
<td>Tx Data progress On (TxDatOn)</td>
<td>[1]</td>
<td>R</td>
<td>Data transmit in progress.&lt;br&gt;0 = not active, 1 = data Tx in progress</td>
</tr>
<tr>
<td>Rx Data Progress On (RxDatOn)</td>
<td>[0]</td>
<td>R</td>
<td>Data receive in progress.&lt;br&gt;0 = not active, 1 = data Rx in progress</td>
</tr>
</tbody>
</table>
## SDI FIFO Status (SDIFSTA) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIFSTA</td>
<td>0x5A000038</td>
<td>R</td>
<td>SDI FIFO status register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SDIFSTA</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO available Detect for Tx (TFDET)</td>
<td>[13]</td>
<td>Indicate that FIFO data is available for transmission when DatMode (SDIDCON[12]) is data transmit mode. If DMA mode is enable, SD host requests DMA operation. 0 = not detect (FIFO full), 1 = detect (0 ≤ FIFO ≤ 63)</td>
<td>0</td>
</tr>
<tr>
<td>FIFO available Detect for Rx (RFDET)</td>
<td>[12]</td>
<td>Indicate that FIFO data is available for reception when DatMode (SDIDCON[12]) is data receive mode. If DMA mode is enable, SD host requests DMA operation. 0 = not detect(FIFO empty), 1 = detect(1 ≤ FIFO ≤ 64)</td>
<td>0</td>
</tr>
<tr>
<td>Tx FIFO Half Full (TFHalf)</td>
<td>[11]</td>
<td>Set to 1 whenever Tx FIFO is less than 33byte. 0 = 33 ≤ Tx FIFO ≤ 64, 1 = 0 ≤ Tx FIFO ≤ 32</td>
<td>0</td>
</tr>
<tr>
<td>Tx FIFO Empty (TFEmpty)</td>
<td>[10]</td>
<td>Set to 1 whenever Tx FIFO is empty. 0 = 1 ≤ Tx FIFO ≤ 64, 1 = Empty(0byte)</td>
<td>0</td>
</tr>
<tr>
<td>Rx FIFO Last Data Ready (RFLast)</td>
<td>[9]</td>
<td>Set to 1 whenever Rx FIFO has last data of all block. 0 = not received yet, 1 = Last data ready</td>
<td>0</td>
</tr>
<tr>
<td>Rx FIFO Full (RFFull)</td>
<td>[8]</td>
<td>Set to 1 whenever Rx FIFO is full. 0 = 0 ≤ Rx FIFO ≤ 63, 1 = Full(64byte)</td>
<td>0</td>
</tr>
<tr>
<td>Rx FIFO Half Full (RFHalf)</td>
<td>[7]</td>
<td>Set to 1 whenever Rx FIFO is more than 31byte. 0 = 0 ≤ Rx FIFO ≤ 31, 1 = 32 ≤ Rx FIFO ≤ 64</td>
<td>0</td>
</tr>
<tr>
<td>FIFO Count (FFCNT)</td>
<td>[6:0]</td>
<td>Number of data (byte) in FIFO</td>
<td>0000000</td>
</tr>
</tbody>
</table>
## SDI Data (SDIDAT) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIDAT</td>
<td>0x5A00003C(Li/W, Li/B, Bi/W)</td>
<td>R/W</td>
<td>SDI data register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x5A00003F(Bi/B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SDIDAT Bit Description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>This field contains the data to be transmitted or received over the SDI channel.</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### NOTES:
1. (Li/W, Li/B): Access by Word/Byte unit when endian mode is Little
2. (Bi/W): Access by Word unit when endian mode is Big
3. (Bi/B): Access by Byte unit when endian mode is Big
## SDI Interrupt Mask (SDIIMSK) Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIIMSK</td>
<td>0x5A000040</td>
<td>R/W</td>
<td>SDI interrupt mask register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
</table>
| [17]| RspCrc Interrupt Enable
Response CRC error interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [16]| CmdSent Interrupt Enable
Command sent(without response) interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [15]| CmdTout Interrupt Enable
Command response timeout interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [14]| RspEnd Interrupt Enable
Command response received interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [13]| RWaItReq Interrupt Enable
Read wait request interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [12]| IOlntDet Interrupt Enable
SD host receives SDIO Interrupt from the card (for SDIO).<br>0 = disable, 1 = interrupt enable | 0             |
FIFO fail error interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [10]| CrcSta Interrupt Enable
CRC status errors interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [9]| DatCrc Interrupt Enable
Data CRC fail interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [8]| DatTout Interrupt Enable
Data timeout interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [7]| DatFin Interrupt Enable
Data counter zero interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [6]| BusyFin Interrupt Enable
Busy checks complete interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [5]| SBitErr Interrupt Enable
Start bit error interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [4]| TFHalf Interrupt Enable
Tx FIFO half interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [3]| TFEmpty Interrupt Enable
Tx FIFO empty interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [2]| RFlast Interrupt Enable
Rx FIFO has last data interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [1]| RFFull Interrupt Enable
Rx FIFO full interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
| [0]| RFHalf Interrupt Enable
Rx FIFO half interrupt.<br>0 = disable, 1 = interrupt enable | 0             |
SDI Data/Busy Timer Register

SDI data/busy timer register has 16-bit counter. In case of 25MHz operation, the countable maximum time is 2.6ms (40ns * 0x10000). But, some cards have very long access time (TAAC), their TAAC are up to 100ms. In this case the SDI generates data timeout error state. To solve this problem follow the below flow chart.

Start

Read (multil or single block)

Is timeout occurs?

Yes

25MHz or 600KHz ?

25MHz

Send stop command

No

Send stop command if multi block

25MHz

Change SDI clock to 25MHz if 600KHz

600KHz

Real-timeout error

Return

Change SDI clock to 600KHz

Retry

Return error
OVERVIEW

The S3C2410X RISC microprocessor can support a multi-master IIC-bus serial interface. A dedicated serial data line (SDA) and a serial clock line (SCL) carry information between bus masters and peripheral devices which are connected to the IIC-bus. The SDA and SCL lines are bi-directional.

In multi-master IIC-bus mode, multiple S3C2410X RISC microprocessors can receive or transmit serial data to or from slave devices. The master S3C2410X can initiate and terminate a data transfer over the IIC-bus. The IIC-bus in the S3C2410X uses Standard bus arbitration procedure.

To control multi-master IIC-bus operations, values must be written to the following registers:

— Multi-master IIC-bus control register, IICCON
— Multi-master IIC-bus control/status register, IICSTAT
— Multi-master IIC-bus Tx/Rx data shift register, IICDS
— Multi-master IIC-bus address register, IICADD

When the IIC-bus is free, the SDA and SCL lines should be both at High level. A High-to-Low transition of SDA can initiate a Start condition. A Low-to-High transition of SDA can initiate a Stop condition while SCL remains steady at High Level.

The Start and Stop conditions can always be generated by the master devices. A 7-bit address value in the first data byte, which is put onto the bus after the Start condition has been initiated, can determine the slave device which the bus master device has selected. The 8th bit determines the direction of the transfer (read or write).

Every data byte put onto the SDA line should be eight bits in total. The bytes can be unlimitedly sent or received during the bus transfer operation. Data is always sent from most-significant bit (MSB) first, and every byte should be immediately followed by an acknowledge (ACK) bit.
Figure 20-1. IIC-Bus Block Diagram

NOTE: IIC Data Hold Time
The IIC data hold time (tSDAH) is minimum 0ns.
(IIC data hold time is minimum 0ns for standard/fast bus mode in IIC specification v2.1.)
Please check the data hold time of your IIC device if it's 0 nS or not.
The IIC controller supports only IIC bus device (standard/fast bus mode), not C bus device.
IIC-BUS INTERFACE

The S3C2410X IIC-bus interface has four operation modes:

— Master transmitter mode
— Master receive mode
— Slave transmitter mode
— Slave receive mode

Functional relationships among these operating modes are described below.

START AND STOP CONDITIONS

When the IIC-bus interface is inactive, it is usually in Slave mode. In other words, the interface should be in Slave mode before detecting a Start condition on the SDA line (a Start condition can be initiated with a High-to-Low transition of the SDA line while the clock signal of SCL is High). When the interface state is changed to Master mode, a data transfer on the SDA line can be initiated and SCL signal generated.

A Start condition can transfer a one-byte serial data over the SDA line, and a Stop condition can terminate the data transfer. A Stop condition is a Low-to-High transition of the SDA line while SCL is High. Start and Stop conditions are always generated by the master. The IIC-bus gets busy when a Start condition is generated. A Stop condition will make the IIC-bus free.

When a master initiates a Start condition, it should send a slave address to notify the slave device. One byte of address field consists of a 7-bit address and a 1-bit transfer direction indicator (showing write or read). If bit 8 is 0, it indicates a write operation (transmit operation); if bit 8 is 1, it indicates a request for data read (receive operation).

The master will finish the transfer operation by transmitting a Stop condition. If the master wants to continue the data transmission to the bus, it should generate another Start condition as well as a slave address. In this way, the read-write operation can be performed in various formats.

![Figure 20-2. Start and Stop Condition](image-url)
DATA TRANSFER FORMAT

Every byte placed on the SDA line should be eight bits in length. The bytes can be unlimitedly transmitted per transfer. The first byte following a Start condition should have the address field. The address field can be transmitted by the master when the IIC-bus is operating in Master mode. Each byte should be followed by an acknowledgement (ACK) bit. The MSB bit of the serial data and addresses are always sent first.

**Write Mode Format with 7-bit Addresses**

```
S | Slave Address 7bits | R/W | A | DATA(1Byte) | A | P
```

*"0" (Write)*

Data Transferred (Data + Acknowledge)

**Write Mode Format with 10-bit Addresses**

```
S | Slave Address 1st 7 bits | R/W | A | Slave Address 2nd Byte | A | DATA | A | P
```

*1110XX *"0" (Write)*

Data Transferred (Data + Acknowledge)

**Read Mode Format with 7-bit Addresses**

```
S | Slave Address 7 bits | R/W | A | DATA | A | P
```

*"1" (Read)*

Data Transferred (Data + Acknowledge)

**Read Mode Format with 10-bit Addresses**

```
S | Slave Address 1st 7 bits | R/W | A | Slave Address 2nd Byte | A | rS | Slave Address 1st 7 Bits | R/W | A | DATA | A | P
```

*1110XX *"1" (Read)*

Data Transferred (Data + Acknowledge)

**NOTES:**

1. S : Start, rS: Repeat Start, P : Stop, A: Acknowledge
2. : From Master to Slave, : From Slave to Master

---

Figure 20-3. IIC-Bus Interface Data Format
ACK SIGNAL TRANSMISSION

To complete a one-byte transfer operation, the receiver should send an ACK bit to the transmitter. The ACK pulse should occur at the ninth clock of the SCL line. Eight clocks are required for the one-byte data transfer. The master should generate the clock pulse required to transmit the ACK bit.

The transmitter should release the SDA line by making the SDA line High when the ACK clock pulse is received. The receiver should also drive the SDA line Low during the ACK clock pulse so that the SDA keeps Low during the High period of the ninth SCL pulse.

The ACK bit transmit function can be enabled or disabled by software (IICSTAT). However, the ACK pulse on the ninth clock of SCL is required to complete the one-byte data transfer operation.
READ-WRITE OPERATION

In Transmitter mode, when the data is transferred, the IIC-bus interface will wait until IIC-bus Data Shift (IICDS) register receives a new data. Before the new data is written into the register, the SCL line will be held low, and then released after it is written. The S3C2410X should hold the interrupt to identify the completion of current data transfer. After the CPU receives the interrupt request, it should write a new data into the IICDS register, again.

In Receive mode, when a data is received, the IIC-bus interface will wait until IICDS register is read. Before the new data is read out, the SCL line will be held low and then released after it is read. The S3C2410X should hold the interrupt to identify the completion of the new data reception. After the CPU receives the interrupt request, it should read the data from the IICDS register.

BUS ARBITRATION PROCEDURES

Arbitration takes place on the SDA line to prevent the contention on the bus between two masters. If a master with a SDA High level detects the other master with a SDA active Low level, it will not initiate a data transfer because the current level on the bus does not correspond to its own. The arbitration procedure will be extended until the SDA line turns High.

However, when the masters simultaneously lower the SDA line, each master should evaluate whether or not the mastership is allocated to itself. For the purpose of evaluation, each master should detect the address bits. While each master generates the slaver address, it should also detect the address bit on the SDA line because the SDA line is likely to get Low rather than to keep High. Assume that one master generates a Low as first address bit, while the other master is maintaining High. In this case, both masters will detect Low on the bus because the Low status is superior to the High status in power. When this happens, Low (as the first bit of address) generating master will get the mastership while High (as the first bit of address) generating master should withdraw the mastership. If both masters generate Low as the first bit of address, there should be an arbitration for the second address bit, again. This arbitration will continue to the end of last address bit.

ABORT CONDITIONS

If a slave receiver cannot acknowledge the confirmation of the slave address, it should hold the level of the SDA line High. In this case, the master should generate a Stop condition and to abort the transfer.

If a master receiver is involved in the aborted transfer, it should signal the end of the slave transmit operation by canceling the generation of an ACK after the last data byte received from the slave. The slave transmitter should then release the SDA to allow a master to generate a Stop condition.

CONFIGURING IIC-BUS

To control the frequency of the serial clock (SCL), the 4-bit prescaler value can be programmed in the IICCON register. The IIC-bus interface address is stored in the IIC-bus address (IICADD) register. (By default, the IIC-bus interface address has an unknown value.)
FLOWCHARTS OF OPERATIONS IN EACH MODE

The following steps must be executed before any IIC Tx/Rx operations.

1) Write own slave address on IICADD register, if needed.
2) Set IICCON register.
   a) Enable interrupt
   b) Define SCL period
3) Set IICSTAT to enable Serial Output

START

Master Tx mode has been configured.

Write slave address to IICDS.

Write 0xF0 (M/T Start) to IICSTAT.

The data of the IICDS is transmitted.

ACK period and then interrupt is pending.

Stop?

Y

Write 0xD0 (M/T Stop) to IICSTAT.

Clear pending bit.

Wait until the stop condition takes effect.

END

N

Write new data transmitted to IICDS.

Clear pending bit to resume.

The data of the IICDS is shifted to SDA.

Figure 20-6. Operations for Master/Transmitter Mode
Write slave address to IICDS.

Write 0xB0 (M/R Start) to IICSTAT.

The data of the IICDS (slave address) is transmitted.

ACK period and then interrupt is pending.

Stop?

Y

Read a new data from IICDS.

N

Write 0x90 (M/R Stop) to IICSTAT.

Clear pending bit.

Clear pending bit.

Wait until the stop condition takes effect.

SDA is shifted to IICDS.

Figure 20-7. Operations for Master/Receiver Mode
START
Slave Tx mode has been configured.

IIC detects start signal. and, IICDS receives data.

IIC compares IICADD and IICDS (the received slave address).

Matched?

N

The IIC address match interrupt is generated.

Write data to IICDS.

Clear pending bit to resume.

Stop?

Y

END

N

The data of the IICDS is shifted to SDA.

Interrupt is pending.

Figure 20-8. Operations for Slave/Transmitter Mode
IIC detects start signal, and, IICDS receives data.

IIC compares IICADD and IICDS (the received slave address).

Matched?

The IIC address match interrupt is generated.

Read data to IICDS.

Clear pending bit to resume.

Stop?

SDA is shifted to IICDS.

Interrupt is pending.

Figure 20-9. Operations for Slave/Receiver Mode
IIC-BUS INTERFACE SPECIAL REGISTERS

MULTI-MASTER IIC-BUS CONTROL (IICCON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IICCON</td>
<td>0x54000000</td>
<td>R/W</td>
<td>IIC-Bus control register</td>
<td>0x0X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IICCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledge generation (note 1)</td>
<td>[7]</td>
<td>IIC-bus acknowledge enable bit. 0 = Disable, 1 = Enable</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In Tx mode, the IICSDA is free in the ack time. In Rx mode, the IICSDA is L in the ack time.</td>
<td></td>
</tr>
<tr>
<td>Tx clock source selection</td>
<td>[6]</td>
<td>Source clock of IIC-bus transmit clock prescaler selection bit. 0 = IICCLK = f_PCLK/16, 1 = IICCLK = f_PCLK/512</td>
<td>0</td>
</tr>
<tr>
<td>Tx/Rx Interrupt (note 5)</td>
<td>[5]</td>
<td>IIC-Bus Tx/Rx interrupt enable/disable bit. 0 = Disable, 1 = Enable</td>
<td>0</td>
</tr>
<tr>
<td>Interrupt pending flag (note 2), (note 3)</td>
<td>[4]</td>
<td>IIC-bus Tx/Rx interrupt pending flag. This bit cannot be written to 1. When this bit is read as 1, the IICSCL is tied to L and the IIC is stopped. To resume the operation, clear this bit as 0. 0 = 1) No interrupt pending (when read), 2) Clear pending condition &amp; Resume the operation (when write). 1 = 1) Interrupt is pending (when read) 2) N/A (when write)</td>
<td>0</td>
</tr>
<tr>
<td>Transmit clock value (note 4)</td>
<td>[3:0]</td>
<td>IIC-Bus transmit clock prescaler. IIC-Bus transmit clock frequency is determined by this 4-bit prescaler value, according to the following formula: Tx clock = IICCLK/(IICCON[3:0]+1).</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

NOTES:
1. Interfacing with EEPROM, the ack generation may be disabled before reading the last data in order to generate the STOP condition in Rx mode.
2. An IIC-bus interrupt occurs 1) when a 1-byte transmit or receive operation is completed, 2) when a general call or a slave address match occurs, or 3) if bus arbitration fails.
3. To adjust the setup time of IICSDA before IISSCL rising edge, IICDS has to be written before clearing the IIC interrupt pending bit.
4. IICCLK is determined by IICCON[6]. Tx clock can vary by SCL transition time. When IICCON[6]=0, IICCON[3:0]=0x0 or 0x1 is not available.
## MULTI-MASTER IIC-BUS CONTROL/STATUS (IICSTAT) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IICSTAT</td>
<td>0x54000004</td>
<td>R/W</td>
<td>IIC-Bus control/status register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

### IICSTAT Bit Description

**Mode selection**
  - 00: Slave receive mode
  - 01: Slave transmit mode
  - 10: Master receive mode
  - 11: Master transmit mode

**Busy signal status / START STOP condition**
  - 0 = read) Not busy (when read)
  - write) STOP signal generation
  - 1 = read) Busy (when read)
  - write) START signal generation.
  - The data in IICDS will be transferred automatically just after the start signal.

**Serial output**
  - 0 = Disable Rx/Tx
  - 1 = Enable Rx/Tx

**Arbitration status flag**
  - 0 = Bus arbitration successful
  - 1 = Bus arbitration failed during serial I/O

**Address-as-slave status flag**
  - 0 = Cleared when START/STOP condition was detected
  - 1 = Received slave address matches the address value in the IICADD

**Address zero status flag**
  - 0 = Cleared when START/STOP condition was detected.
  - 1 = Received slave address is 00000000b.

**Last-received bit status flag**
- [0] IIC-bus last-received bit status flag bit.
  - 0 = Last-received bit is 0 (ACK was received).
  - 1 = Last-received bit is 1 (ACK was not received).
MULTI-MASTER IIC-BUS ADDRESS (IICADD) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IICADD</td>
<td>0x54000008</td>
<td>R/W</td>
<td>IIC-Bus address register</td>
<td>0xXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IICADD</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave address</td>
<td>[7:0]</td>
<td>7-bit slave address, latched from the IIC-bus. When serial output enable = 0 in the IICSTAT, IICADD is write-enabled. The IICADD value can be read any time, regardless of the current serial output enable bit (IICSTAT) setting. Slave address = [7:1] Not mapped = [0]</td>
<td>XXXXXXXXXX</td>
</tr>
</tbody>
</table>

MULTI-MASTER IIC-BUS TRANSMIT/RECEIVE DATA SHIFT (IICDS) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IICDS</td>
<td>0x5400000C</td>
<td>R/W</td>
<td>IIC-Bus transmit/receive data shift register</td>
<td>0xXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IICDS</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data shift</td>
<td>[7:0]</td>
<td>8-bit data shift register for IIC-bus Tx/Rx operation. When serial output enable = 1 in the IICSTAT, IICDS is write-enabled. The IICDS value can be read any time, regardless of the current serial output enable bit (IICSTAT) setting.</td>
<td>XXXXXXXXXX</td>
</tr>
</tbody>
</table>
21 IIS-BUS INTERFACE

OVERVIEW

Currently, many digital audio systems are attracting the consumers on the market, in the form of compact discs, digital audio tapes, digital sound processors, and digital TV sound. The S3C2410X Inter-IC Sound (IIS) bus interface can be used to implement a CODEC interface to an external 8/16-bit stereo audio CODEC IC for mini-disc and portable applications. The IIS bus interface supports both IIS bus data format and MSB-justified data format. The interface provides DMA transfer mode for FIFO access instead of an interrupt. It can transmit and receive data simultaneously as well as transmit or receive data alternatively at a time.
FUNCTIONAL DESCRIPTIONS

Bus interface, register bank, and state machine (BRFC): Bus interface logic and FIFO access are controlled by the state machine.

5-bit dual prescaler (IPSR): One prescaler is used as the master clock generator of the IIS bus interface and the other is used as the external CODEC clock generator.

64-byte FIFOs (Tx_FIFO and Rx_FIFO): In transmit data transfer, data are written to Tx_FIFO, and, in the receive data transfer, data are read from Rx_FIFO.

Master IISCLK generator (SCLKG): In master mode, serial bit clock is generated from the master clock.

Channel generator and state machine (CHNC): IISCLK and IISLRCK are generated and controlled by the channel state machine.

16-bit shift register (SFTR): Parallel data is shifted to serial data output in the transmit mode, and serial data input is shifted to parallel data in the receive mode.

TRANSMIT OR RECEIVE ONLY MODE

Normal Transfer

IIS control register has FIFO ready flag bits for transmit and receive FIFOs. When FIFO is ready to transmit data, the FIFO ready flag is set to ‘1’ if transmit FIFO is not empty.

If transmit FIFO is empty, FIFO ready flag is set to ‘0’. When receive FIFO is not full, the FIFO ready flag for receive FIFO is set to ‘1’; it indicates that FIFO is ready to receive data. If receive FIFO is full, FIFO ready flag is set to ‘0’. These flags can determine the time that CPU is to write or read FIFOs. Serial data can be transmitted or received while the CPU is accessing transmit and receive FIFOs in this way.
DMA Transfer
In this mode, transmit or receive FIFO is accessible by the DMA controller. DMA service request in transmit or receive mode is made by the FIFO ready flag automatically.

Transmit and Receive Mode
In this mode, IIS bus interface can transmit and receive data simultaneously.

AUDIO SERIAL INTERFACE FORMAT

IIS-BUS FORMAT
The IIS bus has four lines including serial data input (IISDI), serial data output (IISDO), left/right channel select (IISLRCK), and serial bit clock (IISCLK); the device generating IISLRCK and IISCLK is the master.

Serial data is transmitted in 2's complement with the MSB first. The MSB is transmitted first because the transmitter and receiver may have different word lengths. The transmitter does not have to know how many bits the receiver can handle, nor does the receiver need to know how many bits are being transmitted.

When the system word length is greater than the transmitter word length, the word is truncated (least significant data bits are set to '0') for data transmission. If the receiver gets more bits than its word length, the bits after the LSB are ignored. On the other hand, if the receiver gets fewer bits than its word length, the missing bits are set to zero internally. And therefore, the MSB has a fixed position, whereas the position of the LSB depends on the word length. The transmitter sends the MSB of the next word at one clock period whenever the IISLRCK is changed.

Serial data sent by the transmitter may be synchronized with either the trailing (HIGH to LOW) or the leading (LOW to HIGH) edge of the clock signal. However, the serial data must be latched into the receiver on the leading edge of the serial clock signal, and so there are some restrictions when transmitting data that is synchronized with the leading edge.

The LR channel select line indicates the channel being transmitted. IISLRCK may be changed either on a trailing or leading edge of the serial clock, but it does not need to be symmetrical. In the slave, this signal is latched on the leading edge of the clock signal. The IISLRCK line changes one clock period before the MSB is transmitted. This allows the slave transmitter to derive synchronous timing of the serial data that will be set up for transmission. Furthermore, it enables the receiver to store the previous word and clear the input for the next word.

MSB (LEFT) JUSTIFIED
MSB / left justified bus format is the same as IIS bus format architecturally. Only, different from the IIS bus format, the MSB justified format realizes that the transmitter always sends the MSB of the next word whenever the IISLRCK is changed.
IIS-BUS INTERFACE

**SMALLING FREQUENCY AND MASTER CLOCK**

Master clock frequency (PCLK) can be selected by sampling frequency as shown in Table 21-1. Because PCLK is made by IIS prescaler, the prescaler value and PCLK type (256 or 384fs) should be determined properly. Serial bit clock frequency type (16/32/48fs) can be selected by the serial bit per channel and PCLK as shown in Table 21-2.

![Diagram of IIS-Bus and MSB (Left)-justified Data Interface Formats](image)

![Diagram of MSB-justified Format](image)

**Table 21-1. CODEC clock (CODECLK = 256 or 384fs)**

<table>
<thead>
<tr>
<th>IISLRCK (fs)</th>
<th>8.000 kHz</th>
<th>11.025 kHz</th>
<th>16.000 kHz</th>
<th>22.050 kHz</th>
<th>32.000 kHz</th>
<th>44.100 kHz</th>
<th>48.000 kHz</th>
<th>64.000 kHz</th>
<th>88.200 kHz</th>
<th>96.000 kHz</th>
</tr>
</thead>
</table>

**Figure 21-2. IIS-Bus and MSB (Left)-justified Data Interface Formats**
### Table 21-2 Usable Serial Bit Clock Frequency (IISCLK = 16 or 32 or 48fs)

<table>
<thead>
<tr>
<th>Serial bit per channel</th>
<th>8-bit</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial clock frequency (IISCLK)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@CODECLK = 256fs</td>
<td>16fs, 32fs</td>
<td>32fs</td>
</tr>
<tr>
<td>@CODECLK = 384fs</td>
<td>16fs, 32fs, 48fs</td>
<td>32fs, 48fs</td>
</tr>
</tbody>
</table>

### IIS-BUS INTERFACE SPECIAL REGISTERS

#### IIS CONTROL (IISCON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IISCON</td>
<td>0x55000000 (Li/HW, Li/W, Bi/W)</td>
<td>R/W</td>
<td>IIS control register</td>
<td>0x100</td>
</tr>
<tr>
<td></td>
<td>0x55000002 (Bi/HW)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IISCON</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left/Right channel index (Read only)</td>
<td>[8] 0 = Left, 1 = Right</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Transmit FIFO ready flag (Read only)</td>
<td>[7] 0 = empty, 1 = not empty</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Receive FIFO ready flag (Read only)</td>
<td>[6] 0 = full, 1 = not full</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Transmit DMA service request</td>
<td>[5] 0 = Disable, 1 = Enable</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Receive DMA service request</td>
<td>[4] 0 = Disable, 1 = Enable</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Transmit channel idle command</td>
<td>[3] In Idle state the IISLRCK is inactive (Pause Tx), 0 = Not idle, 1 = Idle</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Receive channel idle command</td>
<td>[2] In Idle state the IISLRCK is inactive (Pause Rx), 0 = Not idle, 1 = Idle</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>IIS prescaler</td>
<td>[1] 0 = Disable, 1 = Enable</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>IIS interface</td>
<td>[0] 0 = Disable (stop), 1 = Enable (start)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### NOTES:
1. The IISCON register is accessible for each byte, halfword and word unit using STRB/STRH/STR and LDRB/LDRH/LDR instructions or char/short int/int type pointer in Little/Big endian mode.
2. (Li/HW/W): Little/HalfWord/Word
   (Bi/HW/W): Big/HalfWord/Word
**IIS MODE REGISTER (IISMOD) REGISTER**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IISMOD</td>
<td>0x55000004 (Li/W, Li/HW, Bi/W)</td>
<td>R/W</td>
<td>IIS mode register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x55000006 (Bi/HW)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IISMOD</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master/slave mode select</td>
<td>[8]</td>
<td>0 = Master mode (IISLRCK and IISCLK are output mode). 1 = Slave mode (IISLRCK and IISCLK are input mode).</td>
<td>0</td>
</tr>
<tr>
<td>Transmit/receive mode select</td>
<td>[7:6]</td>
<td>00 = No transfer 01 = Receive mode 10 = Transmit mode 11 = Transmit and receive mode</td>
<td>00</td>
</tr>
<tr>
<td>Active level of left/right channel</td>
<td>[5]</td>
<td>0 = Low for left channel (High for right channel) 1 = High for left channel (Low for right channel)</td>
<td>0</td>
</tr>
<tr>
<td>Serial interface format</td>
<td>[4]</td>
<td>0 = IIS compatible format 1 = MSB (Left)-justified format</td>
<td>0</td>
</tr>
<tr>
<td>Serial data bit per channel</td>
<td>[3]</td>
<td>0 = 8-bit 1 = 16-bit</td>
<td>0</td>
</tr>
<tr>
<td>Master clock frequency select</td>
<td>[2]</td>
<td>0 = 256fs 1 = 384fs (fs: sampling frequency)</td>
<td>0</td>
</tr>
<tr>
<td>Serial bit clock frequency select</td>
<td>[1:0]</td>
<td>00 = 16fs 01 = 32fs 10 = 48fs 11 = N/A</td>
<td>00</td>
</tr>
</tbody>
</table>

**NOTES:**
1. The IISMOD register is accessible for each halfword and wordunit using STRH/STR and LDRH/LDR instructions or short int/int type pointer in Little/Big endian mode.
2. (Li/HW/W): Little/HalfWord/Word.
   (Bi/HW/W): Big/HalfWord/Word.
### IIS PRESCALER (IISPSR) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IISPSR</td>
<td>0x550000008 (Li/HW, Li/W, Bi/W)</td>
<td>R/W</td>
<td>IIS prescaler register</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>0x5500000A (Bi/HW)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### IISPSR Bit Description

<table>
<thead>
<tr>
<th>Prescaler control A</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[9:5]</td>
<td></td>
<td>Data value: 0 ~ 31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Prescaler A makes the master clock that is used the internal block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and division factor is N+1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initial State: 00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prescaler control B</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4:0]</td>
<td></td>
<td>Data value: 0 ~ 31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Prescaler B makes the master clock that is used the external block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and division factor is N+1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initial State: 00000</td>
</tr>
</tbody>
</table>

#### NOTES:

1. The IISPSR register is accessible for each byte, halfword and word unit using STRB/STRH/STR and LDRB/LDRH/LDR instructions or char/short int/int type pointer in Little/Big endian mode.
2. (Li/HW/W) : Little/HalfWord/Word. (Bi/HW/W) : Big/HalfWord/Word.
IIS FIFO CONTROL (IISFCON) REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IISFCON</td>
<td>0x55000000C (Li/HW, Li/W, Bi/W) 0x5500000E (Bi/HW)</td>
<td>R/W</td>
<td>IIS FIFO interface register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

IISFCON Bit Description Initial State

- **Transmit FIFO access mode select**
  - [15] 0 = Normal
  - 1 = DMA
  - Initial State: 0

- **Receive FIFO access mode select**
  - [14] 0 = Normal
  - 1 = DMA
  - Initial State: 0

- **Transmit FIFO**
  - [13] 0 = Disable
  - 1 = Enable
  - Initial State: 0

- **Receive FIFO**
  - [12] 0 = Disable
  - 1 = Enable
  - Initial State: 0

- **Transmit FIFO data count (Read only)**
  - [11:6] Data count value = 0 ~ 32
  - Initial State: 000000

- **Receive FIFO data count (Read only)**
  - [5:0] Data count value = 0 ~ 32
  - Initial State: 000000

NOTES:
1. The IISFCON register is accessible for each halfword and word unit using STRH/STR and LDRH/LDR instructions or short int/int type pointer in Little/Big endian mode.
2. (Li/HW/W): Little/HalfWord/Word.
   (Bi/HW/W): Big/HalfWord/Word.

IIS FIFO (IISFIFO) REGISTER

IIS bus interface contains two 64-byte FIFO for the transmit and receive mode. Each FIFO has 16-width and 32-depth form, which allows the FIFO to handles data for each halfword unit regardless of valid data size. Transmit and receive FIFO access is performed through FIFO entry; the address of FENTRY is 0x55000010.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IISFIFO</td>
<td>0x55000010(Li/HW) 0x55000012(Bi/HW)</td>
<td>R/W</td>
<td>IIS FIFO register</td>
<td>0x0</td>
</tr>
</tbody>
</table>

IISFIF Bit Description Initial State

- **FENTRY**
  - [15:0] Transmit/Receive data for IIS
  - Initial State: 0x0

NOTES:
1. The IISFIFO register is accessible for each halfword and word unit using STRH/STR and LDRH/LDR instructions or short int/int type pointer in Little/Big endian mode.
2. (Li/HW): Little/HalfWord.
   (Bi/HW): Big/HalfWord.
OVERVIEW

The S3C2410X Serial Peripheral Interface (SPI) can interface the serial data transfer. The S3C2410X includes two SPI, each of which has two 8-bit shift registers for transmission and receiving, respectively. During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). 8-bit serial data at a frequency is determined by its corresponding control register settings. If you only want to transmit, received data can be dummy. Otherwise, if you only want to receive, you should transmit dummy ‘1’ data.

There are 4 I/O pin signals associated with SPI transfers: the SCK (SPICLK0,1), the MISO (SPIMISO0,1) data line, the MOSI (SPIMOSI0,1) data line, and the active low /SS (nSS0,1) pin (input).

FEATURES

— SPI Protocol (ver. 2.11) compatible
— 8-bit Shift Register for transmit
— 8-bit Shift Register for receive
— 8-bit Prescaler logic
— Polling, Interrupt, and DMA transfer mode
Figure 22-1. SPI Block Diagram
SPI OPERATION

Using the SPI interface, the S3C2410X can send/receive 8-bit data simultaneously with an external device. A serial clock line is synchronized with the two data lines for shifting and sampling of the information. When the SPI is the master, transmission frequency can be controlled by setting the appropriate bit to SPPREn register. You can modify its frequency to adjust the baud rate data register value. When the SPI is a slave, other master supplies the clock. When the programmer writes byte data to SPTDATn register, SPI transmit/receive operation will start simultaneously. In some cases, nSS should be activated before writing byte data to SPTDATn.

Programming Procedure

When a byte data is written into the SPTDATn register, SPI starts to transmit if ENSCK and MSTR of SPCONn register are set. You can use a typical programming procedure to operate an SPI card.

To program the SPI modules, follow these basic steps:

1. Set Baud Rate Prescaler Register (SPPREn).
2. Set SPCONn to configure properly the SPI module.
3. Write data 0xFF to SPTDATn 10 times in order to initialize MMC or SD card.
4. Set a GPIO pin, which acts as nSS, to low to activate the MMC or SD card.
5. Tx data → Check the status of Transfer Ready flag (REDY=1), and then write data to SPTDATn.
6. Rx data(1): SPCONn's TAGD bit disable = normal mode
   → write 0xFF to SPTDATn, then confirm REDY to set, and then read data from Read Buffer.
7. Rx data(2): SPCONn's TAGD bit enable = Tx Auto Garbage Data mode
   → confirm REDY to set, and then read data from Read Buffer(then automatically start to transfer).
8. Set a GPIO pin, which acts as nSS, to high, to deactivate MMC or SD card.
### SPI Transfer Format

The S3C2410X supports 4 different formats to transfer the data. Figure 22-2 shows four waveforms for SPICLK.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>SPICLK</th>
<th>MOSI</th>
<th>MISO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MSB 6 5 4 3 2 1 LSB</td>
<td>MSB 6 5 4 3 2 1 LSB</td>
</tr>
<tr>
<td>CPOL = 0, CPHA = 0 (Format A)</td>
<td></td>
<td>* MSB of character just received</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle</th>
<th>SPICLK</th>
<th>MOSI</th>
<th>MISO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MSB 6 5 4 3 2 1 LSB</td>
<td>MSB 6 5 4 3 2 1 LSB</td>
</tr>
<tr>
<td>CPOL = 0, CPHA = 1 (Format B)</td>
<td></td>
<td>* LSB of previously transmitted character</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle</th>
<th>SPICLK</th>
<th>MOSI</th>
<th>MISO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MSB 6 5 4 3 2 1 LSB</td>
<td>MSB 6 5 4 3 2 1 LSB</td>
</tr>
<tr>
<td>CPOL = 1, CPHA = 0 (Format A)</td>
<td></td>
<td>* MSB of character just received</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle</th>
<th>SPICLK</th>
<th>MOSI</th>
<th>MISO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MSB 6 5 4 3 2 1 LSB</td>
<td>MSB 6 5 4 3 2 1 LSB</td>
</tr>
<tr>
<td>CPOL = 1, CPHA = 1 (Format B)</td>
<td></td>
<td>* LSB of previously transmitted character</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 22-2. SPI Transfer Format**
Transmitting Procedure by DMA
1. The SPI is configured as DMA mode.
2. DMA is configured properly.
3. The SPI requests DMA service.
4. DMA transmits 1byte data to the SPI.
5. The SPI transmits the data to card.
6. Return to Step 3 until DMA count becomes 0.
7. The SPI is configured as interrupt or polling mode with SMOD bits.

Receiving Procedure by DMA
1. The SPI is configured as DMA start with SMOD bits and setting TAGD bit.
2. DMA is configured properly.
3. The SPI receives 1byte data from card.
4. The SPI requests DMA service.
5. DMA receives the data from the SPI.
6. Write data 0xFF automatically to SPTDATn.
7. Return to Step 4 until DMA count becomes 0.
8. The SPI is configured as polling mode with SMOD bits and clearing TAGD bit.
9. If SPSTAn’s REDY flag is set, then read the last byte data.

NOTE: Total received data = DMA TC values + the last data in polling mode (Step 9).
      The first DMA received data is dummy and so the user can neglect it.

SPI Slave Rx Mode with Format B
If the SPI slave Rx mode is activated and SPI format is set to format B, then SPI operation will be failed:
The READY signal, one of internal signals, becomes high before the SPI_CNT reaches 0. Therefore, in DMA mode, DATA_READ signal is generated before the last data is latched.
**Guide**

1) DMA mode: This mode cannot be used at SPI slave Rx mode with format B.

2) Polling mode: DATA_READ signal should be delayed by 1 phase of SPICLK at SPI slave Rx mode with format B.

3) Interrupt mode: DATA_READ signal should be delayed 1 phase of SPICLK at SPI slave Rx mode with format B.

---

![Diagram of SPI Data Receive Function Error (Slave/Receive/Format B Mode)](image-url)
### SPI SPECIAL REGISTERS

#### SPI CONTROL REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPCON0</td>
<td>0x59000000</td>
<td>R/W</td>
<td>SPI channel 0 control register</td>
<td>0x00</td>
</tr>
<tr>
<td>SPCON1</td>
<td>0x59000020</td>
<td>R/W</td>
<td>SPI channel 1 control register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPCONn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
</table>
| SPI Mode Select (SMOD) | [6:5]     | Determine how and by what SPTDAT is read/written.  
00 = polling mode, 01 = interrupt mode  
10 = DMA mode, 11 = reserved | 00           |
| SCK Enable (ENSCK)   | [4]        | Determine whether you want SCK enable or not (for only master).  
0 = disable, 1 = enable | 0            |
| Master/Slave Select (MSTR) | [3]    | Determine the desired mode (master or slave).  
0 = slave, 1 = master  
**NOTE:** In slave mode, there should be set up time for master to initiate Tx/Rx. | 0            |
| Clock Polarity Select (CPOL) | [2]  | Determine an active high or active low clock.  
0 = active high, 1 = active low | 0            |
| Clock Phase Select (CPHA) | [1]    | Select one of two fundamentally different transfer formats.  
0 = format A, 1 = format B | 0            |
| Tx Auto Garbage Data mode enable (TAGD) | [0] | Decide whether the receiving data only needs or not.  
0 = normal mode, 1 = Tx auto garbage data mode  
**NOTE:** In normal mode, if you only want to receive data, you should transmit dummy 0xFF data. | 0            |
## SPI STATUS REGISTER

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPSTA0</td>
<td>0x59000004</td>
<td>R</td>
<td>SPI channel 0 status register</td>
<td>0x01</td>
</tr>
<tr>
<td>SPSTA1</td>
<td>0x59000024</td>
<td>R</td>
<td>SPI channel 1 status register</td>
<td>0x01</td>
</tr>
</tbody>
</table>

### Bit Descriptions

<table>
<thead>
<tr>
<th>SPSTAn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Collision Error Flag (DCOL)</td>
<td>[2]</td>
<td>This flag is set if the SPTDATn is written or the SPRDATn is read while a transfer is in progress and cleared by reading the SPSTAn. 0 = not detect, 1 = collision error detect</td>
<td>0</td>
</tr>
<tr>
<td>Multi Master Error Flag (MULF)</td>
<td>[1]</td>
<td>This flag is set if the nSS signal goes to active low while the SPI is configured as a master, and SPPINn's ENMUL bit is multi master errors detect mode. MULF is cleared by reading SPSTAn. 0 = not detect, 1 = multi master error detect</td>
<td>0</td>
</tr>
<tr>
<td>Transfer Ready Flag (REDY)</td>
<td>[0]</td>
<td>This bit indicates that SPTDATn or SPRDATn is ready to transmit or receive. This flag is automatically cleared by writing data to SPTDATn. 0 = not ready, 1 = data Tx/Rx ready</td>
<td>1</td>
</tr>
</tbody>
</table>
SPI PIN CONTROL REGISTER

When the SPI system is enabled, the direction of pins, except nSS pin, is controlled by MSTR bit of SCON0n register. The direction of nSS pin is always input.
When the SPI is a master, nSS pin is used to check multi-master error, provided the SPPIN’s ENMUL bit is active, and another GPIO should be used to select a slave.
If the SPI is configured as a slave, the nSS pin is used to select SPI as a slave by one master.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPPIN0</td>
<td>0x59000008</td>
<td>R/W</td>
<td>SPI channel 0 pin control register</td>
<td>0x02</td>
</tr>
<tr>
<td>SPPIN1</td>
<td>0x5900028</td>
<td>R/W</td>
<td>SPI channel 1 pin control register</td>
<td>0x02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPPINn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>[7:3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi Master error detect Enable (ENMUL)</td>
<td>[2]</td>
<td>The /SS pin is used as an input to detect multi master error when the SPI system is a master. 0 = disable (general purpose) 1 = multi master error detect enable</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>[1]</td>
<td>This bit should be ‘1’.</td>
<td>1</td>
</tr>
<tr>
<td>Master Out Keep (KEEP)</td>
<td>[0]</td>
<td>Determine MOSI drive or release when 1byte transmit is completed (only master). 0 = release, 1 = drive the previous level</td>
<td>0</td>
</tr>
</tbody>
</table>

The SPIMISO (MISO) and SPIMOSI (MOSI) data pins are used for transmitting and receiving serial data. When the SPI is configured as a master, SPIMISO (MISO) is the master data input line, SPIMOSI (MOSI) is the master data output line, and SPICLK (SCK) is the clock output line. When the SPI becomes a slave, these pins perform reversed roles. In a multiple-master system, SPICLK (SCK) pins, SPIMOSI (MOSI) pins, and SPIMISO (MISO) pins are tied to configure a group respectively.

A master SPI can experience a multi master error, when other SPI device working as a master selects the S3C2410 SPI as a slave. When this error is detected, the following actions are taken immediately. But you must previously set SPPINn’s ENMUL bit if you want to detect this error.

1. The SPCON0n’s MSTR bit is forced to 0 to operate slave mode.
2. The SPSTAn’s MULF flag is set, and an SPI interrupt is generated.
## SPI Interface S3C2410X

### SPI Baud Rate Prescaler Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPPRE0</td>
<td>0x5900000C</td>
<td>R/W</td>
<td>SPI channel 0 baud rate prescaler register</td>
<td>0x00</td>
</tr>
<tr>
<td>SPPRE1</td>
<td>0x5900002C</td>
<td>R/W</td>
<td>SPI channel 1 baud rate prescaler register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

### SPI Baud Rate Prescaler Register

<table>
<thead>
<tr>
<th>SPPREn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescaler Value</td>
<td>[7:0]</td>
<td>Determine SPI clock rate as above equation. Baud rate = PCLK / 2 / (Prescaler value + 1)</td>
<td>0x00</td>
</tr>
</tbody>
</table>

**NOTE:** Baud rate should be less than 25 MHz.

### SPI Tx Data Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPTDAT0</td>
<td>0x59000010</td>
<td>R/W</td>
<td>SPI channel 0 Tx data register</td>
<td>0x00</td>
</tr>
<tr>
<td>SPTDAT1</td>
<td>0x59000030</td>
<td>R/W</td>
<td>SPI channel 1 Tx data register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

### SPI Tx Data Register

<table>
<thead>
<tr>
<th>SPTDATn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx Data Register</td>
<td>[7:0]</td>
<td>This field contains the data to be transmitted over the SPI channel.</td>
<td>0x00</td>
</tr>
</tbody>
</table>

### SPI Rx Data Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPRDAT0</td>
<td>0x59000014</td>
<td>R</td>
<td>SPI channel 0 Rx data register</td>
<td>0x00</td>
</tr>
<tr>
<td>SPRDAT1</td>
<td>0x59000034</td>
<td>R</td>
<td>SPI channel 1 Rx data register</td>
<td>0x00</td>
</tr>
</tbody>
</table>

### SPI Rx Data Register

<table>
<thead>
<tr>
<th>SPRDATn</th>
<th>Bit</th>
<th>Description</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rx Data Register</td>
<td>[7:0]</td>
<td>This field contains the data to be received over the SPI channel.</td>
<td>0x00</td>
</tr>
</tbody>
</table>
OVERVIEW

The bus arbitration logic determines the priorities of bus masters. It supports a combination of rotation priority mode and fixed priority mode.

BUS PRIORITY MAP

The S3C2410X holds eleven bus masters including SDRAM refresh controller, LCD_DMA, DMA0, DMA1, DMA2, DMA3, USB_HOST_DMA, EXT_BUS_MASTER, Test interface controller (TIC), and ARM920T. The following list shows the priorities among these bus masters after a reset:

1. SDRAM refresh controller
2. LCD_DMA
3. DMA0
4. DMA1
5. DMA2
6. DMA3
7. USB host DMA
8. External bus master
9. TIC
10. ARM920T
11. Reserved

Among those bus masters, four DMAs operate under the rotation priority, while others run under the fixed priority.
ABSOLUTE MAXIMUM RATINGS

Table 24-1. Absolute Maximum Rating

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Rating</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Supply Voltage</td>
<td>V_{DDi}</td>
<td>1.8V V_{DD}</td>
<td>2.7</td>
</tr>
<tr>
<td></td>
<td>V_{DDIO}</td>
<td>3.3V V_{DD}</td>
<td>3.8</td>
</tr>
<tr>
<td>DC Input Voltage</td>
<td>V_{IN}</td>
<td>3.3V Input buffer</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.3V Interface / 5V Tolerant input buffer</td>
<td>6.5</td>
</tr>
<tr>
<td>DC Output Voltage</td>
<td>V_{OUT}</td>
<td>3.3V Output buffer</td>
<td>3.8</td>
</tr>
<tr>
<td>DC Input (Latch-up) Current</td>
<td>I_{IN}</td>
<td>± 200</td>
<td>mA</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>T_{STG}</td>
<td>– 65 to 150</td>
<td>°C</td>
</tr>
</tbody>
</table>

RECOMMENDED OPERATING CONDITIONS

Table 24-2. Recommended Operating Conditions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Rating</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Supply Voltage for Internal</td>
<td>V_{DDi}</td>
<td>1.8V V_{DD}</td>
<td>1.8 ± 0.15</td>
</tr>
<tr>
<td>DC Supply Voltage for I/O Block</td>
<td>V_{DDIO}</td>
<td>3.3V V_{DD}</td>
<td>3.3 ± 0.3</td>
</tr>
<tr>
<td>DC Supply Voltage for Analog Core</td>
<td>V_{DD}</td>
<td>3.3V V_{DD}</td>
<td>3.3 ± 5%</td>
</tr>
<tr>
<td>DC Input Voltage</td>
<td>V_{IN}</td>
<td>3.3V Input buffer</td>
<td>3.3 ± 0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.3V Interface / 5V Tolerant input buffer</td>
<td>3.0 – 5.25</td>
</tr>
<tr>
<td>DC Output Voltage</td>
<td>V_{OUT}</td>
<td>3.3V Output buffer</td>
<td>3.3 ± 0.3</td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>T_{OPR}</td>
<td>Commercial</td>
<td>0 to 70 °C</td>
</tr>
</tbody>
</table>
## D.C. ELECTRICAL CHARACTERISTICS

Table 24-3 and 24-4 define the DC electrical characteristics for the standard LVCMOS I/O buffers.

### Table 24-3. Normal I/O PAD DC Electrical Characteristics

\( V_{DD} = 3.3V \pm 0.3V, \ T_A = 0 \text{ to } 70 \ ^\circ C \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameters</th>
<th>Condition</th>
<th>Min</th>
<th>Type</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{IH} )</td>
<td>High level input voltage</td>
<td>LVCMOS interface</td>
<td>2.0</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>( V_{IL} )</td>
<td>Low level input voltage</td>
<td>LVCMOS interface</td>
<td>0.8</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>( V_T )</td>
<td>Switching threshold</td>
<td></td>
<td>1.4</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>( V_{T^+} )</td>
<td>Schmitt trigger, positive-going threshold</td>
<td>CMOS</td>
<td>2.0</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>( V_{T^-} )</td>
<td>Schmitt trigger, negative-going threshold</td>
<td>CMOS</td>
<td>0.8</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>( I_{IH} )</td>
<td>High level input current</td>
<td>Input buffer, ( V_{IN} = V_{DD} )</td>
<td>-10</td>
<td>10</td>
<td></td>
<td>( \mu A )</td>
</tr>
<tr>
<td>( I_{IL} )</td>
<td>Low level input current</td>
<td>Input buffer, ( V_{IN} = V_{SS} )</td>
<td>-10</td>
<td>10</td>
<td></td>
<td>( \mu A )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Input buffer with pull-up</td>
<td>-60</td>
<td>-33</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>( V_{OH} )</td>
<td>High level output voltage</td>
<td>Type B6 ( I_{OH} = -6 \ mA )</td>
<td></td>
<td></td>
<td>2.4</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type B8 ( I_{OH} = -8 \ mA )</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type B12 ( I_{OH} = -12 \ mA )</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>( V_{OL} )</td>
<td>Low level output voltage</td>
<td>Type B6 ( I_{OL} = 6 \ mA )</td>
<td></td>
<td></td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type B8 ( I_{OL} = 8 \ mA )</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type B12 ( I_{OL} = 12 \ mA )</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
</tbody>
</table>

### NOTES:

1. Type B6 means 6mA output driver cell.
2. Type B8 means 8mA output driver cell.
3. Type B12 means 12mA output driver cells.
Table 24-4. USB DC Electrical Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IH}$</td>
<td>High level input voltage</td>
<td></td>
<td>2.5</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL}$</td>
<td>Low level input voltage</td>
<td></td>
<td>0.8</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>$I_{IH}$</td>
<td>High level input current</td>
<td>$V_{in} = 3.3V$</td>
<td>-10</td>
<td>10</td>
<td>µA</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Low level input current</td>
<td>$V_{in} = 0.0V$</td>
<td>-10</td>
<td>10</td>
<td>µA</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>Static Output High</td>
<td>15KΩ to GND</td>
<td>2.8</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>Static Output Low</td>
<td>1.5KΩ to 3.6V</td>
<td>0.3</td>
<td></td>
<td>V</td>
</tr>
</tbody>
</table>

Table 24-5. S3C2410X Power Supply Voltage and Current

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical $V_{DDi} / V_{DDio}$</td>
<td>1.8 / 3.3</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Max. Operating frequency (FCLK)</td>
<td>203</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td>Max. Operating frequency (HCLK)</td>
<td>101.5</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td>Max. Operating frequency (PCLK)</td>
<td>50.7</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td>Typical normal mode power $^{\text{NOTE}(3)}$ (Total $V_{DDi} + V_{DDio}$)</td>
<td>297</td>
<td>mW</td>
<td>$^{\text{NOTE}(1)}$</td>
</tr>
<tr>
<td>Typical normal mode power $^{\text{NOTE}(3)}$ (Total $V_{DDi} + V_{DDio}$)</td>
<td>224</td>
<td>mW</td>
<td>$^{\text{NOTE}(2)}$</td>
</tr>
<tr>
<td>Typical idle mode power $^{\text{NOTE}(3)}$ (Total $V_{DDi} + V_{DDio}$)</td>
<td>122</td>
<td>mW</td>
<td>$^{\text{FCLK} = 203\text{MHz}}$ ($^{\text{F:H:P} = 1:2:4}$)</td>
</tr>
<tr>
<td>Typical slow mode power $^{\text{NOTE}(3)}$ (Total $V_{DDi} + V_{DDio}$)</td>
<td>33</td>
<td>mW</td>
<td>$^{\text{FCLK} = 12\text{MHz}}$ ($^{\text{F:H:P} = 1:1:1}$)</td>
</tr>
<tr>
<td>Maximum Power_OFF mode power</td>
<td>80</td>
<td>uA</td>
<td>$^{\text{At 1.95V/3.6V, Room temperature}}$</td>
</tr>
<tr>
<td>Typical Power_OFF mode power $^{\text{NOTE}(3)}$</td>
<td>8</td>
<td>uA</td>
<td>All other I/O static.</td>
</tr>
<tr>
<td>Maximum RTC power</td>
<td>63</td>
<td>uA</td>
<td>$^{\text{At 1.95V/3.6V, Room temperature}}$</td>
</tr>
<tr>
<td>Typical RTC power $^{\text{NOTE}(3)}$</td>
<td>3</td>
<td>uA</td>
<td>X-tal = 32.768KHz for RTC</td>
</tr>
</tbody>
</table>

NOTES:
1. I/D cache: ON, MMU: ON, Code on SRAM, FCLK:HCLK:PCLK = 203MHz:101.5MHz:50.7MHz
   LCD ON (320x240x16bppx60Hz, color TFT):Timer internal mode(5 Channel run)
   Audio(IIS&DMA,CDCLK=16.9MHz,LRCK=44.1KHz):Integer data quick sort(65536 EA)
2. WinCE 3.0 MPEG play.
3. Room temperature specification.
Table 24-6. Typical Current Decrease by CLKCON Register (FCLK@202.8MHz)  

(Unit: mA)

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>NFC</th>
<th>LCD</th>
<th>USBH</th>
<th>USBD</th>
<th>Timer</th>
<th>SDI</th>
<th>UART</th>
<th>RTC</th>
<th>ADC</th>
<th>IIC</th>
<th>IIS</th>
<th>SPI</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>2.9</td>
<td>5.8</td>
<td>0.4</td>
<td>2.9</td>
<td>0.5</td>
<td>1.9</td>
<td>3.6</td>
<td>0.4</td>
<td>0.4</td>
<td>0.6</td>
<td>0.5</td>
<td>0.5</td>
<td>20.4</td>
</tr>
</tbody>
</table>

NOTE: This table includes each power consumption of each peripherals. For example, If you do not use IIS and you turned off IIS block by CLKCON register, you can save the 0.5mA.

Figure 24-1. Typical Operating Voltage/Frequency Range  
(VDDIO=3.3V, @Room temperature & SMDK2410 board)
A.C. ELECTRICAL CHARACTERISTICS

**Figure 24-2. XTIpll Clock Timing**

**Figure 24-3. EXTCLK Clock Input Timing**

**Figure 24-4. EXTCLK/HCLK in case that EXTCLK is used without the PLL**
Figure 24-5. HCLK/CLKOUT/SCLK in case that EXTCLK is used

Figure 24-6. Manual Reset Input Timing
McU operates by XTipll or EXTCLK clock.

PLL is configured by S/W first time.

VCO is adapted to new clock frequency.

FCLK is new frequency.

Figure 24-7. Power-On Oscillation Setting Timing
Several slow clocks (XTIpll or EXTCLK)

Figure 24-8. Power_OFF Mode Return Oscillation Setting Timing
Figure 24-9. ROM/SRAM Burst READ Timing(I)
(Tacs=0, Tcos=0, Tacc=2, Toch=0, Tcah=0, PMC=0, ST=0, DW=16bit)
Figure 24-10. ROM/SRAM Burst READ Timing(II)
(Tacs=0, Tcos=0, Tacc=2, Toch=0, Tcah=0, PMC=0, ST=1, DW=16bit)
Figure 24-11. External Bus Request in ROM/SRAM Cycle
(Tacs=0, Tcos=0, Tacc=8, Toch=0, Tcah=0, PMC=0, ST=0)
Figure 24-12. ROM/SRAM READ Timing (I)
(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2, PMC=0, ST=0)
Figure 24-13. ROM/SRAM READ Timing (II)  
(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2cycle, PMC=0, ST=1)
Figure 24-14. ROM/SRAM WRITE Timing (I)
(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2, PMC=0, ST=0)
Figure 24-15. ROM/SRAM WRITE Timing (II)
(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2, PMC=0, ST=1)
NOTE: The status of nWait is checked at (Tacc-1) cycle.

Figure 24-16. External nWAIT READ Timing
(Tacs=0, Tcos=0, Tacc=6, Toch=0, Tcah=0, PMC=0, ST=0)

Figure 24-17. External nWAIT WRITE Timing
(Tacs=0, Tcos=0, Tacc=4, Toch=0, Tcah=0, PMC=0, ST=0)
Figure 24-18. Masked-ROM Single READ Timing (Tacs=2, Tcos=2, Tacc=8, PMC=01/10/11)

Figure 24-19. Masked-ROM Consecutive READ Timing (Tacs=0, Tcos=0, Tacc=3, Tpac=2, PMC=01/10/11)
Figure 24-20. SDRAM Single Burst READ Timing (Trp=2, Trcd=2, Tcl=2, DW=16bit)
Figure 24-21. External Bus Request in SDRAM Timing (Trp=2, Trcd=2, Tcl=2)
Figure 24-22. SDRAM MRS Timing
Figure 24-23. SDRAM Single READ Timing(I) (Trp=2, Trcd=2, Tcl=2)
Figure 24-24. SDRAM Single READ Timing(II) (Trp=2, Trcd=2, Tcl=3)
NOTE: Before executing auto/self refresh command, all banks must be in idle state.

Figure 24-25. SDRAM Auto Refresh Timing (Trp=2, Trc=4)
Figure 24-26. SDRAM Page Hit-Miss READ Timing (Trp=2, Trcd=2, Tcl=2)
NOTE: Before executing auto/self refresh command, all banks must be in idle state.

Figure 24-27. SDRAM Self Refresh Timing (Trp=2, Trc=4)
Figure 24-28. SDRAM Single Write Timing (Trp=2, Trcd=2)
Figure 24-29. SDRAM Page Hit-Miss Write Timing (Trp=2, Trcd=2, Tcl=2)
Figure 24-30. External DMA Timing (Handshake, Single transfer)

Figure 24-31. TFT LCD Controller Timing
Figure 24-32. IIS Interface Timing

Figure 24-33. IIC Interface Timing
Figure 24-34. SD/MMC Interface Timing

Figure 24-35. SPI Interface Timing (CPHA=1, CPOL=1)
Figure 24-36. NAND Flash Address/Command Timing

Figure 24-37. NAND Flash Timing
Table 24-7. Clock Timing Constants

\((V_{DDi}, V_{DDalive}, V_{DDiarm} = 1.8 \text{ V } \pm 0.15 \text{ V}, T_A = 0 \text{ to } 70 \text{ °C}, V_{DDMOP} = 3.3 \text{ V } \pm 0.3 \text{ V})\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crystal clock input frequency</td>
<td>f_{XTAL}</td>
<td>10</td>
<td>–</td>
<td>20</td>
<td>MHz</td>
</tr>
<tr>
<td>Crystal clock input cycle time</td>
<td>t_{XTALCYC}</td>
<td>50</td>
<td>–</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>External clock input frequency</td>
<td>f_{EXT}</td>
<td>–</td>
<td>–</td>
<td>66</td>
<td>MHz</td>
</tr>
<tr>
<td>External clock input cycle time</td>
<td>t_{EXTCYC}</td>
<td>15.0</td>
<td>–</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>External clock input low level pulse width</td>
<td>t_{EXTLOW}</td>
<td>7</td>
<td>–</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>External clock to HCLK (without PLL)</td>
<td>t_{EX2HC}</td>
<td>3</td>
<td>–</td>
<td>9</td>
<td>ns</td>
</tr>
<tr>
<td>HCLK (internal) to CLKOUT</td>
<td>t_{HC2CK}</td>
<td>3</td>
<td>–</td>
<td>11</td>
<td>ns</td>
</tr>
<tr>
<td>HCLK (internal) to SCLK</td>
<td>t_{HC2SCLK}</td>
<td>0</td>
<td>–</td>
<td>3</td>
<td>ns</td>
</tr>
<tr>
<td>External clock input high level pulse width</td>
<td>t_{EXTHIGH}</td>
<td>4</td>
<td>–</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>Reset assert time after clock stabilization</td>
<td>t_{RESW}</td>
<td>4</td>
<td>–</td>
<td>–</td>
<td>XTIpll or EXTCLK</td>
</tr>
<tr>
<td>PLL Lock Time</td>
<td>t_{PLL}</td>
<td>200</td>
<td>–</td>
<td>–</td>
<td>uS</td>
</tr>
<tr>
<td>Power_OFF mode return oscillation setting time</td>
<td>t_{OSC2}</td>
<td>–</td>
<td>–</td>
<td>65536</td>
<td>XTIpll or EXTCLK</td>
</tr>
<tr>
<td>The interval before CPU runs after nRESET is released.</td>
<td>t_{RST2RUN}</td>
<td>–</td>
<td>7</td>
<td>–</td>
<td>XTIpll or EXTCLK</td>
</tr>
</tbody>
</table>

\((V_{DDi}, V_{DDalive}, V_{DDiarm} = 1.8 \text{ V } \pm 0.15 \text{ V}, T_A = 0 \text{ to } 70 \text{ °C}, V_{DDMOP} = 3.3 \text{ V } \pm 0.3 \text{ V})\)
Table 24-8. ROM/SRAM Bus Timing Constants

\((V_{DI}, V_{DAliv}, V_{DIarm} = 1.8 \, V \pm 0.15 \, V, T_A = 0 \, to \, 70 \, ^\circ C, V_{DDMOP} = 3.3V \pm 0.3V)\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM/SRAM Address Delay</td>
<td>(t_{RAD})</td>
<td>3</td>
<td></td>
<td>11</td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM Chip select Delay</td>
<td>(t_{RCD})</td>
<td>2</td>
<td></td>
<td>9</td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM Output enable Delay</td>
<td>(t_{ROD})</td>
<td>2</td>
<td></td>
<td>8</td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM read Data Setup time.</td>
<td>(t_{RDS})</td>
<td>4</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM read Data Hold time.</td>
<td>(t_{RDH})</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM Byte Enable Delay</td>
<td>(t_{RBED})</td>
<td>2</td>
<td></td>
<td>8</td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM Write Byte Enable Delay</td>
<td>(t_{RWBED})</td>
<td>2</td>
<td></td>
<td>10</td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM output Data Delay</td>
<td>(t_{RDD})</td>
<td>3</td>
<td></td>
<td>12</td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM external Wait Setup time</td>
<td>(t_{WS})</td>
<td>5</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM external Wait Hold time</td>
<td>(t_{WH})</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>ROM/SRAM Write enable Delay</td>
<td>(t_{RWD})</td>
<td>2</td>
<td></td>
<td>9</td>
<td>ns</td>
</tr>
</tbody>
</table>

Table 24-9. Memory Interface Timing Constants (3.3V)

\((V_{DI}, V_{DAliv}, V_{DIarm} = 1.8 \, V \pm 0.15 \, V, T_A = 0 \, to \, 70 \, ^\circ C, V_{DDMOP} = 3.3V \pm 0.3V)\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDRAM Address Delay</td>
<td>(t_{SAD})</td>
<td>2</td>
<td></td>
<td>7</td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM Chip Select Delay</td>
<td>(t_{SCSD})</td>
<td>2</td>
<td></td>
<td>6</td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM Row active Delay</td>
<td>(t_{SRD})</td>
<td>1</td>
<td></td>
<td>5</td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM Column active Delay</td>
<td>(t_{SCD})</td>
<td>1</td>
<td></td>
<td>5</td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM Byte Enable Delay</td>
<td>(t_{SBED})</td>
<td>2</td>
<td></td>
<td>6</td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM Write enable Delay</td>
<td>(t_{SWD})</td>
<td>2</td>
<td></td>
<td>6</td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM read Data Setup time</td>
<td>(t_{SDS})</td>
<td>4</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM read Data Hold time</td>
<td>(t_{SDH})</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM output Data Delay</td>
<td>(t_{SDD})</td>
<td>2</td>
<td></td>
<td>7</td>
<td>ns</td>
</tr>
<tr>
<td>SDRAM Clock Enable Delay</td>
<td>(T_{cked})</td>
<td>2</td>
<td></td>
<td>5</td>
<td>ns</td>
</tr>
</tbody>
</table>
Table 24-10. External Bus Request Timing Constants

\( (V_{DD} = 1.8 \, V \pm 0.15 \, V, \, T_A = 0 \, \text{to} \, 70 \, ^\circ C, \, V_{EXT} = 3.3V \pm 0.3V) \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ.</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXternal Bus Request Setup</td>
<td>( t_{XnBRQS} )</td>
<td>2</td>
<td>–</td>
<td>5</td>
<td>ns</td>
</tr>
<tr>
<td>eXternal Bus Request Hold</td>
<td>( t_{XnBRQH} )</td>
<td>–</td>
<td>–</td>
<td>1</td>
<td>ns</td>
</tr>
<tr>
<td>eXternal Bus Ack Delay</td>
<td>( t_{XnBACKD} )</td>
<td>9</td>
<td>–</td>
<td>11</td>
<td>ns</td>
</tr>
<tr>
<td>HZ Delay</td>
<td>( t_{HZD} )</td>
<td>4</td>
<td>–</td>
<td>12</td>
<td>ns</td>
</tr>
</tbody>
</table>

Table 24-11. DMA Controller Module Signal Timing Constants

\( (V_{DD} = 1.8 \, V \pm 0.15 \, V, \, T_A = 0 \, \text{to} \, 70 \, ^\circ C, \, V_{EXT} = 3.3V \pm 0.3V) \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ.</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXternal Request Setup</td>
<td>( t_{XRS} )</td>
<td>2</td>
<td>–</td>
<td>6</td>
<td>ns</td>
</tr>
<tr>
<td>aCcess to Ack Delay when Low transition</td>
<td>( t_{CADL} )</td>
<td>9</td>
<td>–</td>
<td>11</td>
<td>ns</td>
</tr>
<tr>
<td>aCcess to Ack Delay when High transition</td>
<td>( t_{CADH} )</td>
<td>9</td>
<td>–</td>
<td>11</td>
<td>ns</td>
</tr>
<tr>
<td>eXternal Request Delay</td>
<td>( t_{XAD} )</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>SCLK</td>
</tr>
</tbody>
</table>
### Table 24-12. TFT LCD Controller Module Signal Timing Constants

\(V_{DD} = 1.8 \, \text{V} \pm 0.15 \, \text{V}, \, T_A = 0 \, \text{to} \, 70 \, ^\circ \text{C}, \, V_{EXT} = 3.3 \, \text{V} \pm 0.3 \, \text{V}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical sync pulse width</td>
<td>Tvspw</td>
<td>VSPW + 1</td>
<td>–</td>
<td>–</td>
<td>Phclk (note1)</td>
</tr>
<tr>
<td>Vertical back porch delay</td>
<td>Tvbpd</td>
<td>VBPD+1</td>
<td>–</td>
<td>–</td>
<td>Phclk</td>
</tr>
<tr>
<td>Vertical front porch delay</td>
<td>Tvfpd</td>
<td>VFPD+1</td>
<td>–</td>
<td>–</td>
<td>Phclk</td>
</tr>
<tr>
<td>VCLK pulse width</td>
<td>Tvclok</td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>Phclk (note2)</td>
</tr>
<tr>
<td>VCLK pulse width high</td>
<td>Tvclokh</td>
<td>0.5</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>VCLK pulse width low</td>
<td>Tvclokl</td>
<td>0.5</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>Hsync setup to VCLK falling edge</td>
<td>Tl2csetup</td>
<td>0.5</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>VDEN set up to VCLK falling edge</td>
<td>Tde2csetup</td>
<td>0.5</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>VDEN hold from VCLK falling edge</td>
<td>Tde2chold</td>
<td>0.5</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>VD setup to VCLK falling edge</td>
<td>Tvd2csetup</td>
<td>0.5</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>VD hold from VCLK falling edge</td>
<td>Tvd2chold</td>
<td>0.5</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>LEND width</td>
<td>Tlewidth</td>
<td>–</td>
<td>1</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>LEND hold from VCLK rising edge</td>
<td>Tle2chold</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>VSYNC setup to HSYNC falling edge</td>
<td>Tl2hsetup</td>
<td>HSPW + 1</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
<tr>
<td>VSYNC hold from HSYNC falling edge</td>
<td>Tl2hhold</td>
<td>HBPD + HFPD + HOZVAL + 3</td>
<td>–</td>
<td>–</td>
<td>Pvclk</td>
</tr>
</tbody>
</table>

**NOTES:**
1. HSYNC period
2. VCLK period

### Table 24-13. IIS Controller Module Signal Timing Constants

\(V_{DD} = 1.8 \, \text{V} \pm 0.15 \, \text{V}, \, T_A = 0 \, \text{to} \, 70 \, ^\circ \text{C}, \, V_{EXT} = 3.3 \, \text{V} \pm 0.3 \, \text{V}\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>IISLRCK delay time</td>
<td>(t_{LRCK})</td>
<td>0.7</td>
<td>–</td>
<td>1.4</td>
<td>ns</td>
</tr>
<tr>
<td>IISDO delay time</td>
<td>(t_{SDO})</td>
<td>0.8</td>
<td>–</td>
<td>1.7</td>
<td>ns</td>
</tr>
<tr>
<td>IISDI Input Setup time</td>
<td>(t_{SDIS})</td>
<td>6.2</td>
<td>–</td>
<td>16.3</td>
<td>ns</td>
</tr>
<tr>
<td>IISDI Input Hold time</td>
<td>(t_{SDIH})</td>
<td>0.1</td>
<td>–</td>
<td>0.1</td>
<td>ns</td>
</tr>
<tr>
<td>CODEC clock frequency</td>
<td>(f_{CODEC})</td>
<td>1/16</td>
<td>–</td>
<td>1</td>
<td>(f_{IIS_BLOCK})</td>
</tr>
</tbody>
</table>
Table 24-14. IIC BUS Controller Module Signal Timing

\( V_{DD} = 1.8 \, \text{V} \pm 0.15 \, \text{V}, \, T_A = 0 \, \text{to} \, 70 \, ^\circ \text{C}, \, V_{EXT} = 3.3 \, \text{V} \pm 0.3 \, \text{V} \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ.</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCL clock frequency</td>
<td>( t_{SCL} )</td>
<td>–</td>
<td>–</td>
<td>std. 100 fast 400</td>
<td>kHz</td>
</tr>
<tr>
<td>SCL high level pulse width</td>
<td>( t_{SCLHIGH} )</td>
<td>std. 4.0 fast 0.6</td>
<td>–</td>
<td>–</td>
<td>µs</td>
</tr>
<tr>
<td>SCL low level pulse width</td>
<td>( t_{SCLLOW} )</td>
<td>std. 4.7 fast 1.3</td>
<td>–</td>
<td>–</td>
<td>µs</td>
</tr>
<tr>
<td>Bus free time between STOP and START</td>
<td>( t_{BUF} )</td>
<td>std. 4.7 fast 1.3</td>
<td>–</td>
<td>–</td>
<td>µs</td>
</tr>
<tr>
<td>START hold time</td>
<td>( t_{STARTS} )</td>
<td>std. 4.0 fast 0.6</td>
<td>–</td>
<td>–</td>
<td>µs</td>
</tr>
<tr>
<td>SDA hold time</td>
<td>( t_{SDAH} )</td>
<td>std. 0 fast 0</td>
<td>–</td>
<td>std. - fast 0.9</td>
<td>µs</td>
</tr>
<tr>
<td>SDA setup time</td>
<td>( t_{SDAS} )</td>
<td>std. 250 fast 100</td>
<td>–</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>STOP setup time</td>
<td>( t_{STOPH} )</td>
<td>std. 4.0 fast 0.6</td>
<td>–</td>
<td>–</td>
<td>µs</td>
</tr>
</tbody>
</table>

**NOTES:** Std. means Standard Mode and fast means Fast Mode.
1. The IIC data hold time(\( t_{SDAH} \)) is minimum 0ns.
   (IIC data hold time is minimum 0ns for standard/fast bus mode in IIC specification v2.1.)
   Please check the data hold time of your IIC device if it's 0 ns or not.
2. The IIC controller supports only IIC bus device(standard/fast bus mode), not C bus device.

Table 24-15. SD/MMC Interface Transmit/Receive Timing Constants

\( V_{DD} = 1.8 \, \text{V} \pm 0.15 \, \text{V}, \, T_A = 0 \, \text{to} \, 70 \, ^\circ \text{C}, \, V_{EXT} = 3.3 \, \text{V} \pm 0.3 \, \text{V} \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ.</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD Command output Delay time</td>
<td>( t_{SDCD} )</td>
<td>0.5</td>
<td>–</td>
<td>1.3</td>
<td>ns</td>
</tr>
<tr>
<td>SD Command input Setup time</td>
<td>( t_{SDCS} )</td>
<td>5.8</td>
<td>–</td>
<td>15.2</td>
<td>ns</td>
</tr>
<tr>
<td>SD Command input Hold time</td>
<td>( t_{SDCH} )</td>
<td>0.1</td>
<td>–</td>
<td>0.1</td>
<td>ns</td>
</tr>
<tr>
<td>SD Data output Delay time</td>
<td>( t_{SDDD} )</td>
<td>0.3</td>
<td>–</td>
<td>0.6</td>
<td>ns</td>
</tr>
<tr>
<td>SD Data input Setup time</td>
<td>( t_{SDDS} )</td>
<td>6.3</td>
<td>–</td>
<td>15.3</td>
<td>ns</td>
</tr>
<tr>
<td>SD Data input Hold time</td>
<td>( t_{SDDH} )</td>
<td>0.1</td>
<td>–</td>
<td>0.1</td>
<td>ns</td>
</tr>
</tbody>
</table>
Table 24-16. SPI Interface Transmit/Receive Timing Constants

\((V_{DD} = 1.8 \, V \pm 0.15 \, V, \, T_A = 0 \, \text{to} \, 70 \, ^\circ C, \, V_{EXT} = 3.3V \pm 0.3V)\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ.</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI MOSI Master Output Delay time</td>
<td>(t_{SPIMOD})</td>
<td>1.0</td>
<td>–</td>
<td>4.2</td>
<td>ns</td>
</tr>
<tr>
<td>SPI MOSI Slave Input Setup time</td>
<td>(t_{SPISIS})</td>
<td>0.1</td>
<td>–</td>
<td>0.1</td>
<td>ns</td>
</tr>
<tr>
<td>SPI MOSI Slave Input Hold time</td>
<td>(t_{SPISIH})</td>
<td>0.8</td>
<td>–</td>
<td>1.8</td>
<td>ns</td>
</tr>
<tr>
<td>SPI MISO Slave output Delay time</td>
<td>(t_{SPISOD})</td>
<td>8.2</td>
<td>–</td>
<td>21.4</td>
<td>ns</td>
</tr>
<tr>
<td>SPI MISO Master Input Setup time</td>
<td>(t_{SPIMIS})</td>
<td>5.6</td>
<td>–</td>
<td>14.7</td>
<td>ns</td>
</tr>
<tr>
<td>SPI MISO Master Input Hold time</td>
<td>(t_{SPIMIH})</td>
<td>0.1</td>
<td>–</td>
<td>0.1</td>
<td>ns</td>
</tr>
</tbody>
</table>

Table 24-17. USB Electrical Specifications

\((V_{DD} = 1.8 \, V \pm 0.15 \, V, \, T_A = 0 \, \text{to} \, 70 \, ^\circ C, \, V_{EXT} = 3.3V \pm 0.3V)\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suspend Device</td>
<td>ICCS</td>
<td></td>
<td></td>
<td>10</td>
<td>µA</td>
</tr>
<tr>
<td>Leakage Current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hi-Z state Input Leakage</td>
<td>ILO</td>
<td>0V &lt; VIN &lt; 3.3V</td>
<td>-10</td>
<td>10</td>
<td>µA</td>
</tr>
<tr>
<td>Input Levels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differential Input Sensitivity</td>
<td>VDI</td>
<td>(</td>
<td>(D^+) - (D^-)</td>
<td>)</td>
<td>0.2</td>
</tr>
<tr>
<td>Differential Common Mode Range</td>
<td>VCM</td>
<td>Includes VDI range</td>
<td>0.8</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>Single Ended Receiver Threshold</td>
<td>VSE</td>
<td></td>
<td>0.8</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>Output Levels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static Output Low</td>
<td>VOL</td>
<td>RL of 1.5Kohm to 3.6V</td>
<td>0.3</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Static Output High</td>
<td>VOH</td>
<td>RL of 15Kohm to GND</td>
<td>2.8</td>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>Capacitance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transceiver Capacitance</td>
<td>CIN</td>
<td>Pin to GND</td>
<td>20</td>
<td></td>
<td>pF</td>
</tr>
</tbody>
</table>
### Table 24-18. USB Full Speed Output Buffer Electrical Characteristics

\( V_{DD} = 1.8 \, V \pm 0.15 \, V, \, T_A = 0 \, \text{to} \, 70 \, ^\circ\text{C}, \, V_{EXT} = 3.3V \pm 0.3V \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition Characteristics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rise Time</td>
<td>TR</td>
<td>CL = 50pF</td>
<td>4.0</td>
<td>2.0</td>
<td>ns</td>
</tr>
<tr>
<td>Fall Time</td>
<td>TF</td>
<td>CL = 50pF</td>
<td>4.0</td>
<td>2.0</td>
<td>ns</td>
</tr>
<tr>
<td>Rise/Fall Time Matching</td>
<td>TRFM</td>
<td>(TR / TF)</td>
<td>90</td>
<td>110</td>
<td>%</td>
</tr>
<tr>
<td>Output Signal Crossover Voltage</td>
<td>VCRS</td>
<td></td>
<td>1.3</td>
<td>2.0</td>
<td>V</td>
</tr>
<tr>
<td>Drive Output Resistance</td>
<td>ZDRV</td>
<td>Steady state drive</td>
<td>28</td>
<td>43</td>
<td>ohm</td>
</tr>
</tbody>
</table>

### Table 24-19. USB Low Speed Output Buffer Electrical Characteristics

\( V_{DD} = 1.8 \, V \pm 0.15 \, V, \, T_A = 0 \, \text{to} \, 70 \, ^\circ\text{C}, \, V_{EXT} = 3.3V \pm 0.3V \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rising Time</td>
<td>TR</td>
<td>CL = 50pF</td>
<td>75</td>
<td>300</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL = 350pF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Falling Time</td>
<td>TF</td>
<td>CL = 50pF</td>
<td>75</td>
<td>300</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL = 350pF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rise/Fall Time Matching</td>
<td>TRFM</td>
<td>(TR / TF)</td>
<td>80</td>
<td>120</td>
<td>%</td>
</tr>
<tr>
<td>Output Signal Crossover Voltage</td>
<td>VCRS</td>
<td></td>
<td>1.3</td>
<td>2.0</td>
<td>V</td>
</tr>
</tbody>
</table>
Table 24-20. NAND Flash Interface Timing Constants

($V_{DDi}$, $V_{DDalive}$, $V_{DDiarm}$ = 1.8 V ± 0.15 V, $T_A$ = 0 to 70 °C, $V_{DDIO}$ = 3.3V ± 0.3V)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFCON Chip Enable delay</td>
<td>$t_{CED}$</td>
<td>–</td>
<td>6.2</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON CLE delay</td>
<td>$t_{CLED}$</td>
<td>–</td>
<td>7.1</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON ALE delay</td>
<td>$t_{ALED}$</td>
<td>–</td>
<td>7.5</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Write Enable delay</td>
<td>$t_{WED}$</td>
<td>–</td>
<td>7.2</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Read Enable delay</td>
<td>$t_{RED}$</td>
<td>–</td>
<td>7.1</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Write Data Setup time</td>
<td>$t_{WDS}$</td>
<td>–</td>
<td>6.5</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Write Data Hold time</td>
<td>$t_{WDH}$</td>
<td>1.7</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Read Data Setup requirement time</td>
<td>$t_{RDS}$</td>
<td>0.3</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>NFCON Read Data Hold requirement time</td>
<td>$t_{RDH}$</td>
<td>0.3</td>
<td>–</td>
<td>ns</td>
</tr>
</tbody>
</table>
25 MECHANICAL DATA

PACKAGE DIMENSIONS

Figure 25-1. 272-FBGA-1414 Package Dimension 1 (Top View)
Figure 25-2. 272-FBGA-1414 Package Dimension 2 (Bottom View)

The recommended land open size is 390 – 410 µm (0.39 – 0.41mm) diameter.
Appendix 1

ARM920T INTRODUCTION

The ARM920T is a member of the ARM9TDMI family of general-purpose microprocessors, which includes:

- ARM9TDMI (ARM9TDMI core)
- ARM940T (ARM9TDMI core plus cache and protection unit)
- ARM920T (ARM9TDMI core plus cache and MMU).

The ARM9TDMI processor core is a Harvard architecture device implemented using a five-stage pipeline consisting of fetch, decode, execute, memory and write stages, and can be provided as a stand-alone core which can be embedded into more complex devices. The stand-alone core has a simple bus interface that allows users to design their own caches/memory systems around it.

The ARM9TDMI family of microprocessors supports both the 32-bit ARM and 16-bit Thumb instruction sets, allowing the user to trade off between high performance and high code density.

The ARM920T is a Harvard cache architecture processor which is targeted at multiprogrammer applications where full memory management, high performance, and low power are all-important. The separate instruction and data caches in this design are 16KB each in size, with an 8-word line length. The ARM920T implements an enhanced ARM Architecture V4 MMU to provide translation and access permission checks for instruction and data addresses.

The ARM920T supports the ARM debug architecture and includes logic to assist in both hardware and software debug. The ARM920T also includes support for coprocessors, exporting the instruction and data buses along with simple handshaking signals.

The ARM920T interface to the rest of the system is via unified address and data buses. This interface is compatible with the Advanced Microcontroller Bus Architecture (AMBA) bus scheme, either as a fully compliant AMBA bus master, or as a slave for production test. The ARM920T also has a TrackingICE mode which allows an approach similar to a conventional ICE mode of operation.
PROCESSOR FUNCTIONAL BLOCK DIAGRAM

Shows the functional block diagram of the ARM920T

Figure 1-1. ARM920T Functional Block Diagram
Appendix 2

PROGRAMMER'S MODEL

ABOUT THE PROGRAMMER'S MODEL

ARM920T incorporates the ARM9TDMI integer core, which implements the ARMv4T architecture. It executes the ARM and Thumb instruction sets, and includes Embedded ICE JTAG software debug features.

The programmer's model of the ARM920T consists of the programmer's model of the ARM9TDMI with the following additions and modifications:

• The ARM920T incorporates two coprocessors:
  — CP14, which allows software access to the debug communications channel. The registers defined in CP14 are accessible with MCR and MRC instructions.
  — The system control coprocessor (CP15), which provides additional registers that are used to configure and control the caches, MMU, protection system, the clocking mode and other system options of the ARM920T, such as big or little-endian operation. The registers defined in CP15 are accessible with MCR and MRC instructions. These are described in CP15 register map summary on page 2-4.

• The ARM920T also features an external coprocessor interface which allows the attachment of a closely coupled coprocessor on the same chip, for example, a floating point unit. Registers and operations provided by any coprocessors attached to the external coprocessor interface will be accessible with appropriate coprocessor instructions.

• Memory accesses for instruction fetches and data loads and stores may be cached or buffered. Cache and write buffer configuration and operation is described in detail in following chapters.

• The MMU page tables which reside in main memory describe the virtual to physical address mapping, access permissions, and cache and write buffer configuration. These are created by the operating system software and accessed automatically by the ARM920T MMU hardware whenever an access causes a TLB miss.

• The ARM920T has a Trace Interface Port which allows the use of Trace hardware and tools for real-time tracing of instructions and data.
ABOUT THE ARM9TDMI PROGRAMMER’S MODEL

The ARM9TDMI processor core implements ARM v4T architecture, and so executes the ARM 32-bit instruction set and the compressed Thumb 16-bit instruction set. The programmer’s model is fully described in the ARM Architecture Reference Manual. The ARM9TDMI Technical Reference Manual gives implementation details including instruction execution cycle times.

The ARM v4T architecture specifies a small number of implementation options. The options selected in the ARM9TDMI implementation are listed in Table 2-1. For comparison, the options selected for the ARM7TDMI implementation are also shown.

<table>
<thead>
<tr>
<th>Processor core</th>
<th>ARM architecture</th>
<th>Data abort model</th>
<th>Value stored by direct STR, STRT, STM of PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI</td>
<td>v4T</td>
<td>Base updated</td>
<td>Address of Inst + 12</td>
</tr>
<tr>
<td>ARM9TDMI</td>
<td>v4T</td>
<td>Base restored</td>
<td>Address of Inst + 12</td>
</tr>
</tbody>
</table>

The ARM9TDMI is code-compatible with the ARM7TDMI, with two exceptions:

- The ARM9TDMI implements the base restored data abort model, which significantly simplifies the software data abort handler.
- The ARM9TDMI fully implements the instruction set extension spaces added to the ARM (32-bit) instruction set in architecture v4 and v4T.

These differences are explained in more detail below.

DATA ABORT MODEL

The base restored data abort model differs from the base updated data abort model implemented by ARM7TDMI.

The difference in the data abort model affects only a very small section of operating system code, the data abort handler. It does not affect user code. With the base restored data abort model, when a data abort exception occurs during the execution of a memory access instruction, the base register is always restored by the processor hardware to the value the register contained before the instruction was executed. This removes the need for the data abort handler to unwind any base register update which may have been specified by the aborted instruction.
INSTRUCTION SET EXTENSION SPACES

All ARM processors implement the undefined instruction space as one of the entry mechanisms for the undefined instruction exception. That is, ARM instructions with opcode[27:25] = 0b011 and opcode[4] = 1 are undefined on all ARM processors including the ARM9TDMI and ARM7TDMI.

ARM architecture v4 and v4T also introduced a number of instruction set extension spaces to the ARM instruction set. These are:

- arithmetic instruction extension space
- control instruction extension space
- coprocessor instruction extension space
- load/store instruction extension space.

Instructions in these spaces are undefined (they cause an undefined instruction exception). The ARM9TDMI fully implements all the instruction set extension spaces defined in ARM architecture v4T as undefined instructions, allowing emulation of future instruction set additions.

The system control coprocessor (CP15) allows configuration and control of the caches, MMU, protection system and clocking mode of the ARM920T.

The ARM920T coprocessor 15 registers are described under the following sections:

- Accessing CP15 registers on page 2-5
- Register 0: ID code register on page 2-7
- Register 0: Cache type register on page 2-8
- Register 1: Control register on page 2-10
- Register 2: Translation table base (TTB) register on page 2-12
- Register 3: Domain access control register on page 2-13
- Register 4: Reserved on page 2-14
- Register 5: Fault status registers on page 2-14
- Register 6: Fault address register on page 2-15
- Register 7: Cache operations on page 2-15
- Register 8: TLB operations on page 2-18
- Register 9: Cache lock down register on page 2-19
- Register 10: TLB lock down register on page 2-21
- Registers 11 -12 & 14: Reserved on page 2-22
- Register 13: Process ID on page 2-22
- Addresses in ARM920T on page 2-6
- Register 15: Test configuration register on page 2-24.
CP15 REGISTER MAP SUMMARY

CP15 defines 16 registers. The register map for CP15 is shown in Table 2-2

Table 2-2. CP15 Register Map

<table>
<thead>
<tr>
<th>Register</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ID code (1)</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>0</td>
<td>Cache type (1)</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>1</td>
<td>Control</td>
<td>Control</td>
</tr>
<tr>
<td>2</td>
<td>Translation table base</td>
<td>Translation table base</td>
</tr>
<tr>
<td>3</td>
<td>Domain access control</td>
<td>Domain access control</td>
</tr>
<tr>
<td>4</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>5</td>
<td>Fault status (2)</td>
<td>Fault status (2)</td>
</tr>
<tr>
<td>6</td>
<td>Fault address</td>
<td>Fault address</td>
</tr>
<tr>
<td>7</td>
<td>Unpredictable</td>
<td>Cache operations</td>
</tr>
<tr>
<td>8</td>
<td>Unpredictable</td>
<td>TLB operations</td>
</tr>
<tr>
<td>9</td>
<td>Cache lockdown (2)</td>
<td>Cache lockdown (2)</td>
</tr>
<tr>
<td>10</td>
<td>TLB lock down (2)</td>
<td>TLB lock down (2)</td>
</tr>
<tr>
<td>11</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>12</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>13</td>
<td>Process ID</td>
<td>Process ID</td>
</tr>
<tr>
<td>14</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>15</td>
<td>Test configuration</td>
<td>Test configuration</td>
</tr>
</tbody>
</table>

NOTES:
1. Register location 0 provides access to more than one register. The register accessed depends upon the value of the opcode_2 field. See the register description for details.
2. Separate registers for instruction and data. See the register description for details.
ACCESSING CP15 REGISTERS

Throughout this section the following terms and abbreviations are used.

Table 2-3. CP15 Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpredictable</td>
<td>UNP</td>
<td>For reads, the data returned when reading from this location is unpredictable; it could have any value. For writes, writing to this location will cause unpredictable behavior, or an unpredictable change in device configuration.</td>
</tr>
<tr>
<td>Should be zero</td>
<td>SBZ</td>
<td>When writing to this location, all bits of this field should be 0.</td>
</tr>
</tbody>
</table>

In all cases, reading from, or writing any data values to any CP15 registers, including those fields specified as unpredictable or should be zero, will not cause any permanent damage.

All CP15 register bits that are defined and contain state, are set to zero by BnRES except V-Bit in register 1, which takes the value of macrocell input VINITHI when BnRES is asserted.

CP15 registers can only be accessed with MRC and MCR instructions in a privileged mode. The instruction bit pattern of the MCR and MRC instructions is shown in Figure 2-1. The assembler for these instructions is

\[
\text{MCR/MRC}\{\text{cond}\} \ P15,\text{opcode}_1,\text{Rd},\text{CRn},\text{CRm},\text{opcode}_2
\]

![Figure 2-1. CP15 MRC and MCR Bit Pattern](image)

Instructions CDP, LDC and STC, along with unprivileged MRC and MCR instructions to CP15 will cause the undefined instruction trap to be taken. The CRn field of MRC and MCR instructions specifies the coprocessor register to access. The CRm field and opcode_2 field are used to specify a particular action when addressing registers.

Attempting to read from a non-readable register, or writing to a non-writable register will cause unpredictable results.

The opcode_1, opcode_2 and CRm fields should be zero, except when the values specified are used to select the desired operations, in all instructions which access CP15. Using other values will result in unpredictable behavior.
Addresses in ARM920T

Three distinct types of address exist in an ARM920T system:

- virtual address (VA)
- modified virtual address (MVA)
- physical address (PA).

Below is an example of the address manipulation when the ARM9TDMI requests an instruction.

1) The VA of the instruction (IVA) is issued by the ARM9TDMI.
2) This is translated by the ProcID to the instruction MVA (IMVA). It is the IMVA that the instruction cache and MMU see.
3) If the protection check carried out by the IMMU on the IMVA does not abort and the IMVA tag is in the instruction cache the instruction data is returned to the ARM9TDMI.
4) If the instruction cache misses (the IMVA tag is not in the instruction cache) then the IMMU performs a translation to produce the instruction PA (IPA). This address is given to the AMBA bus interface to perform an external access

<table>
<thead>
<tr>
<th>Domain</th>
<th>Domain</th>
<th>Caches &amp; TLBs</th>
<th>AMBA bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Virtual</td>
<td>Modified Virtual</td>
<td>Physical</td>
</tr>
</tbody>
</table>
REGISTER 0: ID CODE REGISTER

This is a read-only register which returns a 32-bit device ID code.

The ID code register is accessed by reading CP15 register 0 with the opcode_2 field set to any value other than 1 (the CRm field should be zero when reading). For example:

MRC p15,0,Rd,c0,c0,0; returns ID register

The contents of the ID code are shown in Table 2-5.

<table>
<thead>
<tr>
<th>Register bits</th>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>Implementor</td>
<td>0x41</td>
</tr>
<tr>
<td>23:20</td>
<td>Specification revision</td>
<td>0x1</td>
</tr>
<tr>
<td>19:16</td>
<td>Architecture version (4T)</td>
<td>0x2</td>
</tr>
<tr>
<td>15:4</td>
<td>Part number</td>
<td>0x920</td>
</tr>
<tr>
<td>3:0</td>
<td>Layout revision</td>
<td>0x0</td>
</tr>
</tbody>
</table>
REGISTER 0: CACHE TYPE REGISTER

This is a read-only register which contains information about the size and architecture of the caches, allowing operating systems to establish how to perform such operations as cache cleaning and lockdown. Future ARM cached processors will contain this register, allowing RTOS vendors to produce future-proof versions of their operating systems.

The cache type register is accessed by reading CP15 register 0 with the opcode_2 field set to 1. For example:

MRC p15,0,Rd,c0,c0,1; returns cache details

The format of the register is shown in Table 2-6.

<table>
<thead>
<tr>
<th>Register Bits</th>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:29</td>
<td>Reserved</td>
<td>000</td>
</tr>
<tr>
<td>28:25</td>
<td>Cache type</td>
<td>0110</td>
</tr>
<tr>
<td>24</td>
<td>Harvard/Unified</td>
<td>1 (defines Harvard cache)</td>
</tr>
<tr>
<td>23:21</td>
<td>Reserved</td>
<td>000</td>
</tr>
<tr>
<td>20:18</td>
<td>DCache size</td>
<td>101 (defines 16KB)</td>
</tr>
<tr>
<td>17:15</td>
<td>DCache associativity</td>
<td>110 (defines 64 way)</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td>13:12</td>
<td>DCache words per line</td>
<td>10 (defines 8 words per line)</td>
</tr>
<tr>
<td>11:9</td>
<td>Reserved</td>
<td>000</td>
</tr>
<tr>
<td>8:6</td>
<td>ICache size</td>
<td>101 (defines 16KB)</td>
</tr>
<tr>
<td>5:3</td>
<td>ICache Associativity</td>
<td>110 (defines 64 way)</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td>1:0</td>
<td>ICache words per line</td>
<td>10 (defines 8 words per line)</td>
</tr>
</tbody>
</table>

Bits [28:25] indicate which major cache class the implementation falls into. 0x6 means that the cache provides:

- Cache-clean-step operation
- Cache-flush-step operation
- Lock down facilities
Bits [20:18] give the data cache size. Bits [8:6] give the instruction cache size. Table 2-7 on page 2-9 shows the meaning of values used for cache size encoding.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>512B</td>
</tr>
<tr>
<td>001</td>
<td>1KB</td>
</tr>
<tr>
<td>010</td>
<td>2KB</td>
</tr>
<tr>
<td>011</td>
<td>4KB</td>
</tr>
<tr>
<td>100</td>
<td>8KB</td>
</tr>
<tr>
<td>101</td>
<td>16KB</td>
</tr>
<tr>
<td>110</td>
<td>32KB</td>
</tr>
<tr>
<td>111</td>
<td>64KB</td>
</tr>
</tbody>
</table>

Bits [17:15] give the data cache associativity. Bits [5:3] give the instruction cache associativity. Table 2-8 on page 2-9 shows the meaning of values used for cache associativity encoding.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>512B</td>
</tr>
<tr>
<td>001</td>
<td>1KB</td>
</tr>
<tr>
<td>010</td>
<td>2KB</td>
</tr>
<tr>
<td>011</td>
<td>4KB</td>
</tr>
<tr>
<td>100</td>
<td>8KB</td>
</tr>
<tr>
<td>101</td>
<td>16KB</td>
</tr>
<tr>
<td>110</td>
<td>32KB</td>
</tr>
<tr>
<td>111</td>
<td>64KB</td>
</tr>
</tbody>
</table>

Bits [13:12] give the data cache line length. Bits [1:0] give the instruction cache line length.
Table 2-9 shows the meaning of values used for line length encoding

<table>
<thead>
<tr>
<th>Bits [13:12]/Bits [1:0]</th>
<th>Words Per Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>2</td>
</tr>
<tr>
<td>01</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>16</td>
</tr>
</tbody>
</table>

**REGISTER 1: CONTROL REGISTER**

This register contains the control bits of the ARM920T. All reserved bits should either be written with zero or one, as indicated, or written using read-modify-write. The reserved bits have an unpredictable value when read. To read and write this register:

MRC p15, 0, Rd, c1, c0, 0; read control register
MCR p15, 0, Rd, c1, c0, 0; write control register

All defined control bits are set to zero on reset except the V-Bit which is set to zero at reset if the VINITHI pin is LOW, or one if the VINITHI pin is HIGH. The functions of the control bits are shown in Table 2-10.
### Table 2-10. Control Register 1-bit Functions

<table>
<thead>
<tr>
<th>Register Bits</th>
<th>Name</th>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>iA bit</td>
<td>Asynchronous clock select</td>
<td>See Table 2-11 on page 2-11.</td>
</tr>
<tr>
<td>30</td>
<td>nF bit</td>
<td>notFastBus select</td>
<td>See Table 2-11 on page 2-11.</td>
</tr>
<tr>
<td>29:15</td>
<td>–</td>
<td>Reserved</td>
<td>Read = Unpredictable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write = Should be zero</td>
</tr>
<tr>
<td>14</td>
<td>RR bit</td>
<td>Round robin replacement</td>
<td>0 = Random replacement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Round robin replacement</td>
</tr>
<tr>
<td>13</td>
<td>V bit</td>
<td>Base location of exception registers</td>
<td>0 = Low addresses = 0x0000 0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = High addresses = 0xFFFF 0000</td>
</tr>
<tr>
<td>12</td>
<td>I bit</td>
<td>Instruction cache enable</td>
<td>0 = Instruction cache disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Instruction cache enabled</td>
</tr>
<tr>
<td>11:10</td>
<td>–</td>
<td>Reserved</td>
<td>Read = 00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write = 00</td>
</tr>
<tr>
<td>9</td>
<td>R bit</td>
<td>ROM protection</td>
<td>This bit modifies the MMU protection system. See Table 3-6 on page 3-20</td>
</tr>
<tr>
<td>8</td>
<td>S bit</td>
<td>System protection</td>
<td>This bit modifies the MMU protection system. See Table 3-6 on page 3-20</td>
</tr>
<tr>
<td>7</td>
<td>B bit</td>
<td>Big-endian/little-endian</td>
<td>0 = Little-endian operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Big-endian operation</td>
</tr>
<tr>
<td>6:3</td>
<td>–</td>
<td>Reserved</td>
<td>Read = 1111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write = 1111</td>
</tr>
<tr>
<td>2</td>
<td>C bit</td>
<td>Data cache enable</td>
<td>0 = Data cache disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Data cache enabled</td>
</tr>
<tr>
<td>1</td>
<td>A bit</td>
<td>Alignment fault enable</td>
<td>Data address alignment fault checking</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = Fault checking disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Fault checking enabled</td>
</tr>
<tr>
<td>0</td>
<td>M bit</td>
<td>MMU enable</td>
<td>0 = MMU disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = MMU enabled</td>
</tr>
</tbody>
</table>

Register 1 bits 31:30 select the clocking mode of the ARM920T, as shown in Table 2-11.

### Table 2-11. Clocking Modes

<table>
<thead>
<tr>
<th>Clocking Mode</th>
<th>iA</th>
<th>nF</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastBus mode</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Synchronous</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Enabling the MMU

Care must be taken with the address mapping of the code sequence used to enable the MMU, see Enabling the MMU on page 3-25.

See Instruction cache enable/disable on page 4-3 and Data cache and write buffer enable/disable on page 4-6 for restrictions and effects of having caches enabled with the MMU disabled.

REGISTER 2: TRANSLATION TABLE BASE (TTB) REGISTER

This is the translation table base register, for the currently active first level translation table. The contents of register 2 are shown in Table 2-12.

<table>
<thead>
<tr>
<th>Register Bits</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:14</td>
<td>Pointer to first level translation table base. Read/write</td>
</tr>
<tr>
<td>13:0</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>Read = Unpredictable</td>
</tr>
<tr>
<td></td>
<td>Write = Should be zero</td>
</tr>
</tbody>
</table>

Reading from register 2 returns the pointer to the currently active first level translation table in bits[31:14]. Writing to register 2 updates the pointer to the first level translation table from the value in bits[31:14] of the written value.

Bits[13:0] should be zero when written, and are unpredictable when read.

The following instructions can be used to access the TTB:

MRC p15, 0, Rd, c2, c0, 0; read TTB register
MCR p15, 0, Rd, c2, c0, 0; write TTB register
REGISTER 3: DOMAIN ACCESS CONTROL REGISTER

Register 3 is the read/write domain access control register consisting of sixteen 2-bit fields. Each of these 2-bit fields defines the access permissions for the domains shown in Table 2-13.

Table 2-13. Register 3: Domain Access Control

<table>
<thead>
<tr>
<th>Register Bits</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:30</td>
<td>D15</td>
</tr>
<tr>
<td>29:28</td>
<td>D14</td>
</tr>
<tr>
<td>27:26</td>
<td>D13</td>
</tr>
<tr>
<td>25:24</td>
<td>D12</td>
</tr>
<tr>
<td>23:22</td>
<td>D11</td>
</tr>
<tr>
<td>21:20</td>
<td>D10</td>
</tr>
<tr>
<td>19:18</td>
<td>D9</td>
</tr>
<tr>
<td>17:16</td>
<td>D8</td>
</tr>
<tr>
<td>15:14</td>
<td>D7</td>
</tr>
<tr>
<td>13:12</td>
<td>D6</td>
</tr>
<tr>
<td>11:10</td>
<td>D5</td>
</tr>
<tr>
<td>9:8</td>
<td>D4</td>
</tr>
<tr>
<td>7:6</td>
<td>D3</td>
</tr>
<tr>
<td>5:4</td>
<td>D2</td>
</tr>
<tr>
<td>3:2</td>
<td>D1</td>
</tr>
<tr>
<td>1:0</td>
<td>D0</td>
</tr>
</tbody>
</table>

The encoding of the two bit domain access permission field is given in Table 3-5 on page 3-19. The following instructions can be used to access the domain access control register:

MRC p15, 0, Rd, c3, c0, 0; read domain 15:0 access permissions
MCR p15, 0, Rd, c3, c0, 0; write domain 15:0 access permissions
REGISTER 4: RESERVED

Accessing (reading or writing) this register will cause unpredictable behavior.

REGISTER 5: FAULT STATUS REGISTERS

Register 5 is the fault status register (FSR). The FSR contains the source of the last data fault, indicating the domain and type of access being attempted when the data abort occurred.

Table 2-14. Fault Status Register

<table>
<thead>
<tr>
<th>Register Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:9</td>
<td>UNP when read/SBZ for write.</td>
</tr>
<tr>
<td>8</td>
<td>0 when read/SBZ for write.</td>
</tr>
<tr>
<td>7:4</td>
<td>Domain being accessed when fault occurred (D15-D0).</td>
</tr>
<tr>
<td>3:0</td>
<td>Fault type.</td>
</tr>
</tbody>
</table>

The fault type encoding is shown in Fault address and fault status registers on page 3-18.

The data FSR is defined in ARM architecture v4T. Additionally, a pipelined prefetch FSR is available, for debug purposes only. The pipeline matches that of the ARM9TDMI.

The following instructions can be used to access the data and prefetch FSR:

MRC p15, 0, Rd, c5, c0, 0 ;read data FSR value
MCR p15, 0, Rd, c5, c0, 0 ;write data FSR value
MRC p15, 0, Rd, c5, c0, 1 ;read prefetch FSR value
MCR p15, 0, Rd, c5, c0, 1 ;write prefetch FSR value

The ability to write to the FSR is useful for a debugger to restore the value of the FSR. The register should be written using the read-modify-write method. Bits[31:8] should be zero.
REGISTER 6: FAULT ADDRESS REGISTER

Register 6 is the fault address register (FAR) which contains the modified virtual address of the access being attempted when the last fault occurred. The FAR is only updated for data faults, not for prefetch faults.

(The address for a prefetch fault can be found in R14.)

The following instructions can be used to access the FAR:

MRC p15, 0, Rd, c6, c0, 0 ; read FAR data
MCR p15, 0, Rd, c6, c0, 0 ; write FAR data

The ability to write to the FAR is intended for a debugger to restore a previous state.

REGISTER 7: CACHE OPERATIONS

Register 7 is a write-only register used to manage the instruction and data caches, ICache and DCache.

The cache operations provided by register 7 are described in Table 2-15.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidate cache</td>
<td>Invalidates all cache data, including any dirty data (note). Use with caution.</td>
</tr>
<tr>
<td>Invalidate single entry using modified virtual address</td>
<td>Invalidates a single cache line, discarding any dirty data (note). Use with caution.</td>
</tr>
<tr>
<td>Clean D single entry using either index or modified virtual address</td>
<td>Writes the specified cache line to main memory if the line is marked valid and dirty and marks the line as not dirty (note). The valid bit is unchanged.</td>
</tr>
<tr>
<td>Clean and Invalidate D entry using either index or modified virtual address</td>
<td>Writes the specified cache line to main memory if the line is marked valid and dirty (note). The line is marked not valid.</td>
</tr>
<tr>
<td>Prefetch cache line</td>
<td>Performs an ICache lookup of the specified modified virtual address. If the cache misses, and the region is cacheable, a linefill will be performed.</td>
</tr>
</tbody>
</table>

NOTE: Dirty data is data that has been modified in the cache but not yet written to main memory.
The function of each cache operation is selected by the opcode_2 and CRm fields in the MCR instruction used to write CP15 register 7. Writing other opcode_2 or CRm values is unpredictable.

Reading from CP15 register 7 is unpredictable.

Table 2-16 on page 2-16 shows instructions that can be used to perform cache operations with register 7

### Table 2-16. Cache Operations Register 7

<table>
<thead>
<tr>
<th>Function</th>
<th>Data</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidate ICache &amp; DCache</td>
<td>SBZ</td>
<td>MCR p15,0,Rd,c7,c7,0</td>
</tr>
<tr>
<td>Invalidate ICache</td>
<td>SBZ</td>
<td>MCR p15,0,Rd,c7,c5,0</td>
</tr>
<tr>
<td>Invalidate ICache single entry (using MVA)</td>
<td>MVA format</td>
<td>MCR p15,0,Rd,c7,c5,1</td>
</tr>
<tr>
<td>Prefetch ICache line (using MVA)</td>
<td>MVA format</td>
<td>MCR p15,0,Rd,c7,c13,1</td>
</tr>
<tr>
<td>Invalidate DCache</td>
<td></td>
<td>MCR p15,0,Rd,c7,c6,0</td>
</tr>
<tr>
<td>Invalidate DCache single entry (using MVA)</td>
<td>MVA format</td>
<td>MCR p15,0,Rd,c7,c6,1</td>
</tr>
<tr>
<td>Clean DCache single entry (using MVA)</td>
<td>MVA format</td>
<td>MCR p15,0,Rd,c7,c10,1</td>
</tr>
<tr>
<td>Clean and Invalidate DCache entry (using MVA)</td>
<td>MVA format</td>
<td>MCR p15,0,Rd,c7,c14,1</td>
</tr>
<tr>
<td>Clean DCache single entry (using index)</td>
<td>Index format</td>
<td>MCR p15,0,Rd,c7,c10,2</td>
</tr>
<tr>
<td>Clean and Invalidate DCache entry (using index)</td>
<td>Index format</td>
<td>MCR p15,0,Rd,c7,c14,2</td>
</tr>
<tr>
<td>Drain write buffer (1)</td>
<td>SBZ</td>
<td>MCR p15,0,Rd,c7,c10,4</td>
</tr>
<tr>
<td>Wait for interrupt (2)</td>
<td>SBZ</td>
<td>MCR p15,0,Rd,c7,c0,4</td>
</tr>
</tbody>
</table>

### NOTES:
1. Will stop execution until the write buffer has drained.
2. Will stop execution in a LOW power state until an interrupt occurs.
The operations which can be carried out upon a single cache line identify the line using the data passed in the MCR instruction. The data is interpreted using one of the following formats:

**Figure 2-2. Register 7 MVA Format**

**Figure 2-3. Register 7 Index Format**

The use of register 7 is discussed in Chapter 4 Caches, Write Buffer and Physical Address TAG (PATAG) RAM.
REGISTER 8: TLB OPERATIONS

Register 8 is a write-only register used to manage the translation lookaside buffers (TLBs), the instruction TLB and the data TLB.

Five TLB operations are defined and the function to be performed is selected by the opcode_2 and CRm fields in the MCR instruction used to write CP15 register 8. Writing other opcode_2 or CRm values is unpredictable. Reading from CP15 register 8 is unpredictable.

Table 2-17 on page 2-18 shows instructions that can be used to perform TLB operations using register 8.

<table>
<thead>
<tr>
<th>Function</th>
<th>Data</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidate TLB(s)</td>
<td>SBZ</td>
<td>MCR p15,0,Rd,c8,c7,0</td>
</tr>
<tr>
<td>Invalidate I TLB</td>
<td>SBZ</td>
<td>MCR p15,0,Rd,c8,c5,0</td>
</tr>
<tr>
<td>Invalidate I TLB single entry (using MVA)</td>
<td>MVA format</td>
<td>MCR p15,0,Rd,c8,c5,1</td>
</tr>
<tr>
<td>Invalidate D TLB</td>
<td>SBZ</td>
<td>MCR p15,0,Rd,c8,c6,0</td>
</tr>
<tr>
<td>Invalidate D TLB single entry (using MVA)</td>
<td>MVA format</td>
<td>MCR p15,0,Rd,c8,c6,1</td>
</tr>
</tbody>
</table>

**NOTE:** These functions invalidate all the un-preserved entries in the TLB.
Invalidate TLB single entry functions invalidate any TLB entry corresponding to the modified virtual address given in Rd, regardless of its preserved state. See Register 10: TLB lock down register on page 2-21.

Figure 2-4 shows the modified virtual address format used for operations on single entry TLB lines using register 8.

![Figure 2-4. Register 8 MVA Format](image)
REGISTER 9: CACHE LOCK DOWN REGISTER

Register 9 is the cache lock down register. The cache lock down register is 0x0 on reset. The cache lock down register allows software to control which cache line in the ICache or DCache respectively is loaded for a linefill and to prevent lines in the ICache or DCache from being evicted during a linefill, locking them into the cache.

There is a register for each of the ICache and DCache, the value of opcode_2 determines which cache register to access:

opcode_2 = 0x0 causes the DCache register to be accessed
opcode_2 = 0x1 causes the ICache register to be accessed.

The Opcode_1 and CRm fields should be zero.

Reading CP15 register 9 returns the value of the cache lock down register, which is the base pointer for all cache segments.


Writing CP15 register 9 updates the cache lock down register, both the base and the current victim pointer for all cache segments. Bits [25:0] should be zero.

The victim counter specifies the cache line to be used as the victim for the next linefill. This is incremented using either a random or round-robin replacement policy, determined by the state of the RR-bit in register 1. The victim counter generates values in the range (base to 63). This locks lines with index values in the range (0 to base-1). If base = 0, there are no locked lines.

Writing to CP15 register 9 updates the base pointer and the current victim pointer. The next linefill will use and then increment the victim pointer. The victim pointer will continue incrementing on linefills and will wrap around to the base pointer. For example, setting the base pointer to 0x3 prevents the victim pointer from selecting entries 0x0 to 0x2, locking them into the cache.

Load a cache line into ICache line 0 and lock it down:
MCR to CP15 register 9, opcode_2 = 0x1, Victim=Base=0x0
MCR I prefetch. Assuming the ICache misses, a linefill will occur to line 0.
MCR to CP15 register 9, opcode_2 = 0x1, Victim=Base=0x1

Further ICache linefills will now occur into lines 1 - 63.

Load a cache line into DCache line 0 and lock it down:
MCR to CP15 register 9, opcode_2 = 0x0, Victim=Base=0x0
Data load (LDR/LDM). Assuming the DCache misses, a linefill will occur to line 0.
MCR to CP15 register 9, opcode_2 = 0x0, Victim=Base=0x1

Further DCache linefills will now occur into lines 1 - 63.

NOTE: Writing CP15 register 9, with the CRm field set to 0b00001, updates the current victim pointer only for the specified segment only. Bits [31:26] specify the victim; bits [7:5] specify the segment (for a 16KB cache) and all other bits should be zero. This encoding is intended for debug use. It is not necessary and not advised, to use this encoding.
Figure 2-5 shows the format of bits in register 9

![Figure 2-5. Register 9](image)

Table 2-18 shows the instructions needed to access the cache lock down register:

<table>
<thead>
<tr>
<th>Function</th>
<th>Data</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read DCache lock down base</td>
<td>Base</td>
<td>MRC p15,0,Rd,c9,c0,0</td>
</tr>
<tr>
<td>Write DCache victim and lock down base</td>
<td>Victim=Base</td>
<td>MCR p15,0,Rd,c9,c0,0</td>
</tr>
<tr>
<td>Read ICache lock down base</td>
<td>Base</td>
<td>MRC p15,0,Rd,c9,c0,1</td>
</tr>
<tr>
<td>Write ICache victim and lock down base</td>
<td>Victim=Base</td>
<td>MCR p15,0,Rd,c9,c0,1</td>
</tr>
</tbody>
</table>
REGISTER 10: TLB LOCK DOWN REGISTER

Register 10 is the TLB lock down register. The TLB lock down register is 0x0 on reset. There is a TLB lock down register for each of the TLBs, the value of opcode_2 determines which TLB register to access:

opcode_2 = 0x0 causes the D TLB register to be accessed
opcode_2 = 0x1 causes the I TLB register to be accessed.

Reading CP15 register 10 returns the value of the TLB lock down counter base register, the current victim number and the preserve bit (P bit). Note that bits [19:1] are unpredictable when read.

Writing CP15 register 10 updates the TLB lock down counter base register, the current victim pointer and the state of the preserve bit. Bits [19:1] should be zero when written.

Table 2-19 shows the instructions needed to access the TLB lock down register.

Table 2-19. Accessing the TLB Lock Down Register 10

<table>
<thead>
<tr>
<th>Function</th>
<th>Data</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read D TLB lock down</td>
<td>TLB lock-down</td>
<td>MRC p15,0,Rd,c10,c0,0</td>
</tr>
<tr>
<td>Write D TLB lock down</td>
<td>TLB lock-down</td>
<td>MCR p15,0,Rd,c10,c0,0</td>
</tr>
<tr>
<td>Read I TLB lock down</td>
<td>TLB lock-down</td>
<td>MRC p15,0,Rd,c10,c0,1</td>
</tr>
<tr>
<td>Write I TLB lock down</td>
<td>TLB lock-down</td>
<td>MCR p15,0,Rd,c10,c0,1</td>
</tr>
</tbody>
</table>

Figure 2-6 shows the format of bits in register 10.

The entries in the TLBs are replaced using a round robin replacement policy. This is implemented using a victim counter which counts from entry 0 up to 63 and then wraps back round to the base value and continues counting, wrapping around to the base value from 63 each time.

There are two mechanisms available for ensuring entries are not removed from the TLB:

Locking an entry down prevents it from being selected for overwriting during a table walk, this is achieved by programming the base value to which the victim counter reloads. For example, if the bottom 3 entries (0–2) are to be locked down, the base counter should be programmed to 3.

An entry can also be preserved during an Invalidate All instruction. This is done by ensuring the P bit is set when the entry is loaded into the TLB.
Load a single entry into I TLB location 0, make it immune to Invalidate All and lock it down:
MCR to CP15 register 10, opcode_2 = 0x1, Base Value = 0, Current Victim = 0, P = 1
MCR I prefetch. Assuming an I TLB miss occurs, then entry 0 will be loaded.
MCR to CP15 register 10, opcode_2 = 0x1, Base Value = 1, Current Victim = 1, P = 0

Load a single entry into D TLB location 0, make it immune to Invalidate All and lock it down:
MCR to CP15 register 10, opcode_2 = 0x0, Base Value = 0, Current Victim = 0, P = 1
Data load (LDR/LDM) or store (STR/STM). Assuming a D TLB miss occurs, then entry 0 will be loaded.
MCR to CP15 register 10, opcode_2 = 0x0, Base Value = 1, Current Victim = 1, P = 0

REGISTERS 11-12 & 14: RESERVED
Accessing (reading or writing) any of these registers will cause unpredictable behavior.

REGISTER 13: PROCESS ID
Register 13 is the process identifier register. The process identifier register is 0x0 on reset.
Reading from CP15 register 13 returns the value of the process identifier. Writing CP15 register 13 updates the
process identifier to the value in bits [31:25]. Bits [24:0] should be zero.

Register 13 bit assignments are shown in Figure 2-7.

```
31 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Proc ID 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SBZ
```

Figure 2-7. Register 13

Register 13 can be accessed using the following instructions:
MRC p15, 0, Rd, c13, c0, 0 ;read process identifier
MCR p15, 0, Rd, c13, c0, 0 ;write process identifier
Using the process Identifier (ProcID)

Addresses issued by the ARM9TDMI core in the range 0 to 32MB are translated by CP15 register 13, the ProcID register. Address A becomes A + (ProcID x 32MB). It is this translated address that is seen by both the Caches and MMU. Addresses above 32MB undergo no translation. This is shown in Figure 2-8 on page 2-23.

The ProcID is a seven bit field, enabling 64 x 32MB processes to be mapped.

NOTES

If ProcID is zero, as it is on reset, then there is a flat mapping between the ARM9TDMI and the Caches and MMU.

---

![Figure 2-8. Address Mapping Using CP15 Register 13](image-url)
Changing the ProcID - performing a fast context switch

A fast context switch is done by writing to CP15 register 13. The contents of the caches and TLBs do not have to be flushed after a fast context switch because they still hold valid address tags. It should be noted that the two instructions after the MCR to write the ProcID will have been fetched with the old ProcID:

{ProcID = 0}

MOV r0, #1:SHL:25 ; Fetched with ProcID = 0
MCR p15,0,r0,c13,c0,0 ; Fetched with ProcID = 0
A1 ; Fetched with ProcID = 0
A2 ; Fetched with ProcID = 0
A3 ; Fetched with ProcID = 1

REGISTER 15: TEST CONFIGURATION REGISTER

Register 15 is used for test purposes. Accessing (reading or writing) this register will cause the ARM920T to have unpredictable behavior.
Appendix 3

MMU

ABOUT THE MMU

ARM920T implements an enhanced ARM Architecture V4 MMU to provide translation and access permission checks for the instruction and data address ports of the ARM9TDMI. The MMU is controlled from a single set of two-level page tables stored in main memory, and are enabled by M-Bit in CP15 register 1, providing a single address translation and protection scheme. The instruction and data TLBs in the MMU can be independently locked and flushed.

The MMU features are:

- standard ARM V4 MMU mapping sizes, domains, and access protection scheme
- mapping sizes are 1MB sections, 64KB large pages, 4KB small pages and new 1KB tiny pages
- access permissions for sections
- access permissions for large pages and small pages can be specified separately for each quarter of the page (these quarters are called sub-pages)
- 16 domains implemented in hardware
- 64 entry instruction TLB and 64 entry data TLB
- hardware page table walks
- round-robin replacement algorithm (also called cyclic)
- invalidate whole TLB via CP15 Register 8
- invalidate TLB entry, selected by modified virtual address, via CP15 register 8
- independent lockdown of instruction TLB and data TLB via CP15 register 10.

ACCESS PERMISSIONS AND DOMAINS

For large and small pages, access permissions are defined for each sub-page (1KB for small pages, 16KB for large pages). Sections and tiny pages have a single set of access permissions.

All regions of memory have an associated domain. A domain is the primary access control mechanism for a region of memory and defines the conditions in which an access can proceed. The domain determines whether:

- the access permissions are used to qualify the access
- the access is unconditionally allowed to proceed
- the access is unconditionally aborted.

In the latter two cases, the access permission attributes are ignored.

There are 16 domains, which are configured using the domain access control register.
TRANSLATED ENTRIES

Each TLB caches 64 translated entries. During CPU memory accesses, the TLB provides the protection information to the access control logic.

If the TLB contains a translated entry for the modified virtual address, the access control logic determines whether access is permitted:

- If access is permitted and an off-chip access is required, the MMU outputs the appropriate physical address corresponding to the modified virtual address.
- If access is permitted and an off-chip access is not required, the cache services the access.
- If access is not permitted, the MMU signals the CPU core to abort.

If a TLB misses (it does not contain an entry for the virtual address) the translation table walk hardware is invoked to retrieve the translation information from a translation table in physical memory. Once retrieved, the translation information is written into the TLB, possibly overwriting an existing value.

The entry to be written is chosen by cycling sequentially through the TLB locations. To enable use of TLB locking features, the location to be written can be specified using CP15 register 10, TLB lockdown.

When the MMU is turned off (as happens on reset), no address mapping occurs and all regions are marked as non-cacheable and non-bufferable. See About the caches and write buffer on page 4-1.
**MMU PROGRAM ACCESSIBLE REGISTERS**

Table 3-1 shows system control coprocessor (CP15) registers which are used, in conjunction with page table descriptors stored in memory, to determine the operation of the MMU.

<table>
<thead>
<tr>
<th>Register</th>
<th>Number</th>
<th>Bits</th>
<th>Register description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control register</td>
<td>1</td>
<td>M, A, S, R</td>
<td>Contains bits to enable the MMU (M bit), enable data address alignment checks (A bit) and to control the access protection scheme (S bit and R bit).</td>
</tr>
<tr>
<td>Translation table base register</td>
<td>2</td>
<td>31:14</td>
<td>Holds the physical address of the base of the translation table maintained in main memory. This base address must be on a 16KB boundary and is common to both TLBs.</td>
</tr>
<tr>
<td>Domain access control register</td>
<td>3</td>
<td>31:0</td>
<td>Comprises sixteen 2-bit fields. Each field defines the access control attributes for one of 16 domains (D15–D0).</td>
</tr>
<tr>
<td>Fault status register</td>
<td>5 (I &amp; D)</td>
<td>7:0</td>
<td>Indicates the cause of a data and prefetch abort and the domain number of the aborted access, when an abort occurs. Bits 7:4 specify which of the 16 domains (D15–D0) was being accessed when a fault occurred. Bits 3:0 indicate the type of access being attempted. The value of all other bits is unpredictable. The encoding of these bits is shown in Table 3-4 on page 3-18.</td>
</tr>
<tr>
<td>Fault address register</td>
<td>6 (D)</td>
<td>31:0</td>
<td>Holds the virtual address associated with the access that caused the data abort. See Table 3-4 on page 3-18 for details of the address stored for each type of fault. ARM9TDMI Register 14 can be used to determine the virtual address associated with a prefetch abort.</td>
</tr>
<tr>
<td>TLB operations register</td>
<td>8</td>
<td>31:0</td>
<td>Writing to this register causes the MMU to perform TLB maintenance operations; either invalidating all the (unpreserved) entries in the TLB, or invalidating a specific entry.</td>
</tr>
<tr>
<td>TLB lock down register</td>
<td>10 (I &amp; D)</td>
<td>31:20 &amp; 0</td>
<td>Allows specific page table entries to be locked into the TLB and the TLB victim index to be read/written: opcode 2 = 0x0 accesses the D TLB lock down register opcode 2 = 0x1 accesses the I TLB lock down register. Locking entries in the TLB guarantees that accesses to the locked page or section can proceed without incurring the time penalty of a TLB miss. This allows the execution latency for time-critical pieces of code such as interrupt handlers to be minimized.</td>
</tr>
</tbody>
</table>
All the CP15 MMU registers, except register 8, contain state and can be read using MRC instructions and written using MCR instructions. Registers 5 and 6 are also written by the MMU during a data abort. Writing to Register 8 causes the MMU to perform a TLB operation, to manipulate TLB entries. This register cannot be read. The instruction TLB (ITLB) and data TLB (DTLB) both have a copy of register 10, the opcode_2 field in the CP15 instruction is used to determine which one is accessed.

The system control coprocessor (CP15) is described in Programmer’s Model on page 2-1. Details of register format and the coprocessor instructions to access them are given there.

ADDRESS TRANSLATION

The MMU translates virtual addresses generated by the CPU core (and by CP15 register 13) into physical addresses to access external memory. It also derives and checks the access permission, using a translation lookaside buffer (TLB).

The MMU table walking hardware is used to add entries to the TLB. The translation information, which comprises both the address translation data and the access permission data, resides in a translation table located in physical memory. The MMU provides the logic needed to traverse this translation table and load entries into the TLB.

There are up to two stages to the hardware table walking (and hence permission checking) process. The number of stages depends on whether the address in question has been marked as a section-mapped access or a page-mapped access. There is one size of section and three sizes of page-mapped access (large pages, small pages and tiny pages). The translation process always starts out in the same way, with a level one fetch. A section-mapped access requires only a level one fetch, but a page-mapped access requires a subsequent level two fetch.
Figure 3-1. Translating Page Tables
HARDWARE TRANSLATION PROCESS

TRANSLATION TABLE BASE

The hardware translation process is initiated when the TLB does not contain a translation for the requested modified virtual address. The translation table base (TTB) register points to the base address of a table in physical memory which contains section and/or Page descriptors. The 14 low-order bits of the TTB register are set to zero on a read and the table must reside on a 16KB boundary.

![Translation Table Base Register](image)

The translation table has up to 4096 x 32-bit entries, each describing 1MB of virtual memory. This allows up to 4GB of virtual memory to be addressed. Figure 3-1 on page 3-5 illustrates the table walk process.
LEVEL ONE FETCH

Bits 31:14 of the translation table base register are concatenated with bits 31:20 of the modified virtual address to produce a 30-bit address as illustrated in Figure 3-3 on page 3-7.

This address selects a 4-byte translation table entry which is a level one descriptor for either a section or a page table.

Figure 3-3. Accessing the Translation Table Level One Descriptors
LEVEL ONE DESCRIPTOR

The level one descriptor returned is either a section descriptor a coarse page table descriptor, or a fine page table descriptor. A section descriptor provides the base address of a 1MB block of memory. The page table descriptors provide the base address of a page table that contains level two descriptors.

There are two sizes of page table:

- coarse page tables have 256 entries, splitting the 1MB the table describes into 4KB blocks
- fine page tables have 1024 entries, splitting the 1MB the table describes into 1KB blocks

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Invalid</td>
<td>Generates a section translation fault.</td>
</tr>
<tr>
<td>01</td>
<td>Coarse page table</td>
<td>Indicates that this is a coarse page table descriptor.</td>
</tr>
<tr>
<td>10</td>
<td>Section</td>
<td>Indicates that this is a section descriptor.</td>
</tr>
<tr>
<td>11</td>
<td>Fine page table</td>
<td>Indicates that this is a fine page table descriptor.</td>
</tr>
</tbody>
</table>
SECTION DESCRIPTOR

Bits 3:2 (C & B) indicate whether the area of memory mapped by this section is treated as write-back cacheable, write-through cacheable, non-cached buffered or non-cached non-buffered.

Bit 4 should be written to 1 for backward compatibility.

Bits 8:5 specify one of the 16 possible domains (held in the domain access control registers) that contain the primary access controls.

Bit 9 is always written as 0.

Bits 11:10 (AP) specify the access permissions for this section.

Bits 19:12 are always written as 0.

Bits 31:20 form the corresponding bits of the physical address for a section.

COARSE PAGE TABLE_DESCRIPTOR

Bits 3:2 are always written as 0.

Bit 4 is always written as 1.

Bits 8:5 specify one of the 16 possible domains (held in the Domain access control registers) that contain the primary access controls.

Bit 9 is always written as 0.

Bits 31:10 form the base for referencing the level two descriptor. (The coarse page table index for the entry is derived from the modified virtual address.)

If a coarse page table descriptor is returned from the level one fetch, a level two fetch is initiated.

FINE PAGE TABLE_DESCRIPTOR

Bits 3:2 are always written as 0.

Bit 4 is always written as 1.

Bits 8:5 specify one of the 16 possible domains (held in the domain access control registers) that contain the primary access controls.

Bits 11:9 are always written as 0.

Bits 31:12 form the base for referencing the level two descriptor. (The fine page table index for the entry is derived from the modified virtual address.)

If a fine page table descriptor is returned from the level one fetch, a level two fetch is initiated.
TRANSLATING SECTION REFERENCES

Figure 3-5 illustrates the complete section translation sequence. Note that access permissions contained in the level one descriptor must be checked before the physical address is generated.

![Diagram of section translation process]

Figure 3-5. Section Translation
LEVEL TWO DESCRIPTOR

If the level one fetch returns either a coarse page table descriptor or a fine page table descriptor, this provides the base address of the page table to be used. The page table is then accessed and a level two descriptor is returned. This defines either a tiny, a small or a large page descriptor:

- a tiny page descriptor provides the base address of a 1KB block of memory
- a small page descriptor provides the base address of a 4KB block of memory
- a large page descriptor provides the base address of a 64KB block of memory

Coarse page tables have 256 entries, each entry describing 4KB. These entries can provide base addresses for either small or large pages. Large page descriptors must be repeated in 16 consecutive entries.

Fine page tables have 1024 entries, each entry describing 1KB. These entries can provide base addresses for either tiny, small or large pages. Small page descriptors must be repeated in 4 consecutive entries and large page descriptors must be repeated in 64 consecutive entries.

The figure below shows the format of level one descriptors.

```
| 31  | 16  | 15  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2  | 1  | 0  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|    |    |    |
|     |     |     |     |     |     |     |     |     |     |     |     |     | Fault |    |    |    |
| 0   | 0   |     |     |     |     |     |     |     |     |     |     |     | Large page |    |    |    |
|     |     |     |     |     |     |     |     |     |     |     |     |     | Small page |    |    |    |
|     |     |     |     |     |     |     |     |     |     |     |     |     | Tiny page |    |    |    |
|     |     |     |     |     |     |     |     |     |     |     |     |     |         |    |    |    |
|     |     |     |     |     |     |     |     |     |     |     |     |     |         |    |    |    |
|     |     |     |     |     |     |     |     |     |     |     |     |     |         |    |    |    |
```

Figure 3-6. Page Table Entry (Level One Descriptor)

Bits[1:0] indicate the page size and validity and are interpreted as follows.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Invalid</td>
<td>Generates a page translation fault.</td>
</tr>
<tr>
<td>01</td>
<td>Large page</td>
<td>Indicates that this is a 64KB page.</td>
</tr>
<tr>
<td>10</td>
<td>Small page</td>
<td>Indicates that this is a 4KB page.</td>
</tr>
<tr>
<td>11</td>
<td>Tiny page</td>
<td>Indicates that this is a 1KB page.</td>
</tr>
</tbody>
</table>
Bit 3:2 (C & B) indicate whether the area of memory mapped by this page is treated as write-back cacheable, write-through cacheable, non cached buffered or non-cached non-buffered.

Domain access control on page 3-19 and Fault checking sequence on page 3-21 show how to interpret the access permission (ap) bits.

NOTES

Tiny pages do not support sub page permissions and therefore only have one set of access permission bits.

Bits 31:10 (tiny pages), 31:12 (small pages) or bits 31:16 (large pages) are used to form the corresponding bits of the physical address.

TRANSLATING LARGE PAGE REFERENCES

Figure 3-7 on page 3-13 illustrates the complete translation sequence for a 64KB large page.

As the upper four bits of the page index and low-order four bits of the coarse page table index overlap, each coarse page table entry for a large page must be duplicated 16 times (in consecutive memory locations) in the coarse page table.

If a large page descriptor is included in a fine page table the upper six bits of the page index and low-order six bits of the fine page table index overlap, each fine page table entry for a large page must therefore be duplicated 64 times.
Figure 3-7. Large Page Translation from a Coarse Page Table
TRANSLATING SMALL PAGE REFERENCES

Figure 3-8 illustrates the complete translation sequence for a 4KB small page. If a small page descriptor is included in a fine page table, the upper two bits of the page index and low-order two bits of the fine page table index overlap. Each fine page table entry for a small page must therefore be duplicated four times.

![Small Page Translation Diagram]

Figure 3-8. Small Page Translation from a Coarse Page Table
TRANSLATING TINY PAGE REFERENCES

Figure 3-9 on page 3-16 illustrates the complete translation sequence for a 1KB tiny page. Page translation involves one additional step beyond that of a section translation: the level one descriptor is the fine page table descriptor and this is used to point to the level one descriptor.

NOTES

The domain specified in the level one description and access permissions specified in the level one description together determine whether the access has permissions to proceed. See section Domain access control on page 3-19) for details.
Figure 3-9. Tiny Page Translation from a Fine Page Table
SUB-PAGES

Access permissions can be defined for sub pages of small and large pages. If, during a page walk, a small or large page has a non-identical sub page permission, only the sub page being accessed is written into the TLB. For example, a 16KB (large page) sub page entry will be written into the TLB if the sub page permission differs and a 64KB entry will be put in the TLB if the sub page permissions are identical.

When sub page permissions are used and the page entry then needs invalidating, all four sub pages must be invalidated separately.

MMU FAULTS AND CPU ABORTS

The MMU generates an abort on the following types of faults:

- alignment faults (data accesses only)
- translation faults
- domain faults
- permission faults.

In addition, an external abort may be raised by the external system as a result of certain types of external data access.

Alignment fault checking is enabled by the A-bit in CP15 register 1. Alignment fault checking is not affected by whether or not the MMU is enabled. Translation, domain and permission faults are only generated when the MMU is enabled.

The access control mechanisms of the MMU detect the conditions that produce these faults. If a fault is detected as the result of a memory access, the MMU will abort the access and signal the fault condition to the CPU core. The MMU retains status and address information about faults generated by the data accesses in the fault status register and fault address register (see section Fault address and fault status registers on page 3-18). The MMU does not retain status about faults generated by instruction fetches.

An access violation for a given memory access inhibits any corresponding external access, with an abort returned to the CPU core.
FAULT ADDRESS AND FAULT STATUS REGISTERS

On a data abort, the MMU places an encoded 4 bit value, FS[3:0], along with the 4-bit encoded domain number, in the Data fault status register (FSR). Similarly, on a prefetch abort, in the Prefetch fault status register, intended for debug purposes only. In addition, the modified virtual address associated with the data abort is latched into the fault address register (FAR). If an access violation simultaneously generates more than one source of abort, they are encoded in the priority given in Table 3-4. The fault address register is not updated by faults caused by instruction prefetches.

FAULT STATUS

The remainder of this chapter describes the various access permissions and controls supported by the data MMU and details how these are interpreted to generate faults.

Table 3-4. Priority Encoding of Fault Status

<table>
<thead>
<tr>
<th>Source</th>
<th>Status</th>
<th>Domain</th>
<th>FAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest priority Alignment</td>
<td>0b00x1</td>
<td>invalid</td>
<td>MVA of access causing abort</td>
</tr>
<tr>
<td>Translation</td>
<td>Section Page</td>
<td>0b0101 0b0111</td>
<td>invalid valid</td>
</tr>
<tr>
<td>Domain</td>
<td>Section Page</td>
<td>0b1001 0b1011</td>
<td>valid valid</td>
</tr>
<tr>
<td>Permission</td>
<td>Section Page</td>
<td>0b1101 0b1111</td>
<td>valid valid</td>
</tr>
<tr>
<td>Lowest priority External abort on NCNB access or NCB read.</td>
<td>Section Page</td>
<td>0b1000 0b1010</td>
<td>valid valid</td>
</tr>
</tbody>
</table>

NOTES:

1. Data FSR only.
   Alignment faults may write either 0b0001 or 0b0011 into FS[3:0].
   Invalid values in domain[3:0] occur because the fault is raised before a valid domain field has been read from a page table descriptor.
   Any abort masked by the priority encoding may be regenerated by fixing the primary abort and restarting the instruction.
   NCNB means Non-Cacheable and Non-Bufferable.
   NCB means Non-Cacheable but Bufferable.
2. Instruction FSR only.
   The same priority applies as for the Data fault status register, except that alignment faults cannot occur, and external aborts apply only to NC (Non-cacheable) reads.
DOMAIN ACCESS CONTROL

MMU accesses are primarily controlled via domains. There are 16 domains and each has a 2-bit field to define access to it. Two types of user are supported, clients and managers. See Table 3-5. The domains are defined in the domain access control register. Figure 3-10 illustrates how the 32 bits of the register are allocated to define the 16 2-bit domains.

![Figure 3-10. Domain Access Control Register Format](image)

Table 3-5 defines how the bits within each domain are interpreted to specify the access permissions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>No Access</td>
<td>Any access will generate a domain fault.</td>
</tr>
<tr>
<td>01</td>
<td>Client</td>
<td>Accesses are checked against the access permission bits in the section or page descriptor.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td>Reserved. Currently behaves like the no access mode.</td>
</tr>
<tr>
<td>11</td>
<td>Manager</td>
<td>Accesses are not checked against the access permission bits so a permission fault cannot be generated.</td>
</tr>
</tbody>
</table>
Table 3-6 shows how to interpret the access permission (AP) bits and how their interpretation is dependent upon the S and R bits (control register bits 8 and 9).

**Table 3-6. Interpreting Access Permission (AP) Bits**

<table>
<thead>
<tr>
<th>AP</th>
<th>S</th>
<th>R</th>
<th>Supervisor Permissions</th>
<th>User Permissions</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>No access</td>
<td>No access</td>
<td>Any access generates a permission fault</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>Read only</td>
<td>No access</td>
<td>Supervisor read only permitted</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>Read only</td>
<td>Read only</td>
<td>Any write generates a permission fault</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>1</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>x</td>
<td>x</td>
<td>Read/write</td>
<td>No access</td>
<td>Access allowed only in supervisor mode</td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td>x</td>
<td>Read/write</td>
<td>Read only</td>
<td>Writes in user mode cause permission fault</td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td>x</td>
<td>Read/write</td>
<td>Read/write</td>
<td>All access types permitted in both modes.</td>
</tr>
<tr>
<td>xx</td>
<td>1</td>
<td>1</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FAULT CHECKING SEQUENCE

The sequence by which the MMU checks for access faults is different for sections and pages. The sequence for both types of access is shown below. The conditions that generate each of the faults are described on the following pages.

Figure 3-11. Sequence for Checking Faults
ALIGNMENT FAULT

If alignment fault is enabled (A-Bit in CP15 register 1 set), the MMU will generate an alignment fault on any data word access the address of which is not word aligned, or on any halfword access the address of which is not halfword aligned, irrespective of whether the MMU is enabled or not. An alignment fault will not be generated on any instruction fetch, nor on any byte access.

NOTES

If the access generates an alignment fault, the access sequence will abort without reference to further permission checks.

TRANSLATION FAULT

There are two types of translation fault, section and page:

Section  A section translation fault is generated if the level one descriptor is marked as invalid. This happens if bits[1:0] of the descriptor are both 0.

Page  A page translation fault is generated if the level one descriptor is marked as invalid. This happens if bits[1:0] of the descriptor are both 0.

DOMAIN FAULT

There are two types of domain fault, section and page. In both cases the level one descriptor holds the 4-bit domain field which selects one of the 16 2-bit domains in the domain access control register. The two bits of the specified domain are then checked for access permissions as detailed in Table 3-6 on page 3-20. In the case of a section, the domain is checked once the level one descriptor is returned and in the case of a page, the domain is checked once the level one descriptor is returned.

If the specified access is either no access (00) or reserved (10) then either a section domain fault or page domain fault occurs.
PERMISSION FAULT

If the 2-bit domain field returns 01 (client) then access permissions are checked as follows:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If the level one descriptor defines a section-mapped access, the AP bits of the descriptor define whether or not the access is allowed according to Table 3-6 on page 3-20. Their interpretation is dependent upon the setting of the S and R bits (control register bits 8 and 9). If the access is not allowed, a section permission fault is generated.</td>
</tr>
<tr>
<td>Large page, small page</td>
<td>If the level one descriptor defines a page-mapped access and the level two descriptor is for a large or small page, four access permission fields (ap3-ap0) are specified, each corresponding to one quarter of the page. Hence, for small pages ap3 is selected by the top 1KB of the page and ap0 is selected by the bottom 1KB of the page. For large pages, ap3 is selected by the top 16KB of the page and ap0 is selected by the bottom 16KB of the page. The selected AP bits are then interpreted in exactly the same way as for a section (see Table 3-6 on page 3-20), the only difference being the fault generated is a page permission fault.</td>
</tr>
<tr>
<td>Tiny page</td>
<td>If the level one descriptor defines a page-mapped access and the level two descriptor is for a tiny page, the AP bits of the level one descriptor define whether or not the access is allowed in the same way as for a section. The fault generated is a page permission fault.</td>
</tr>
</tbody>
</table>
EXTERNAL ABORTS

In addition to the MMU-generated aborts the ARM920T can be externally aborted by the AMBA bus, which may be used to flag an error on an external memory access. However, not all accesses can be aborted in this way and the Bus Interface Unit (BIU) ignores external aborts that can not be handled.

The following accesses may be aborted:

- non-cached reads
- unbuffered writes
- read-lock-write sequence, to non-cacheable memory.

In the case of a read-lock-write (SWP) sequence in which the read aborts, the write will always be attempted.
INTERACTION OF THE MMU AND CACHES

The MMU is enabled and disabled using bit 0 of the CP15 control register.

ENABLING THE MMU

To enable the MMU:

1) Program the translation table base and domain access control registers.
2) Program level 1 and level 2 page tables as required.
3) Enable the MMU by setting bit 0 in the control register.

Care must be taken if the translated address differs from the untranslated address as several instructions following the enabling of the MMU may have been prefetched with the MMU off (using physical = virtual address - flat translation) and enabling the MMU may be considered as a branch with delayed execution. A similar situation occurs when the MMU is disabled. Consider the following code sequence:

MRC p15, 0, R1, c1, C0, 0: Read control rejection
ORR R1, #0x1
MCR p15,0,R1,C1, C0,0 ; Enable MMU
Fetch Flat
Fetch Flat
Fetch Translated

The instruction and data caches can be enabled simultaneously with the MMU using a single MCR instruction.

DISABLING THE MMU

To disable the MMU, clear bit 0 in the control register. The data cache should be disabled prior to, or, at the same time as the MMU is disabled by clearing Bit 2 of the control register. See the paragraph in Enabling the MMU regarding prefetch effects.

NOTES

If the MMU is enabled, then disabled and subsequently re-enabled the contents of the TLBs will have been preserved. If these are now invalid, the TLBs should be invalidated before the MMU is re-enabled. See Register 8: TLB operations on page 2-18.
APPENDIX 4

CACHES, WRITE BUFFER

ABOUT THE CACHES AND WRITE BUFFER

The ARM920T includes an instruction cache, a data cache, a write buffer and a Physical Address TAG RAM to reduce the effect of main memory bandwidth and latency on performance.

- The ARM920T implements separate 16KB instruction and 16KB data caches.
- The caches have the following features:
  - Virtually-addressed 64-way associative cache.
  - 8 words per line (32 bytes per line) with one valid bit and two dirty bits per line, allowing half-line write-backs.
  - Write-through and write-back cache operation (write-back caches are also known as copy back caches), selected per memory region by the C and B bits in the MMU translation tables (for data cache only).
  - Pseudo-random or round-robin replacement, selectable via RR bit in CP15 register 1.
  - Low-power CAM-RAM implementation.
  - Caches independently lockable with granularity of 1/64th of cache, which is 64 words (256 bytes).
  - For compatibility with Microsoft WindowsCE, and to reduce interrupt latency, the physical address corresponding to each data cache entry is stored in the physical address TAG RAM for use during cache line write-backs, in addition to the virtual address TAG stored in the cache CAMs. This means that the MMU is not involved in cache write-back operations, removing the possibility of TLB misses related to the write-back address.
  - Cache maintenance operations to provide efficient cleaning of the entire data cache, and to provide efficient cleaning and invalidation of small regions of virtual memory. The latter allows ICache coherency to be efficiently maintained when small code changes occur, for example self-modifying code and changes to exception vectors.

The write buffer can hold 16 words of data and four addresses.
INSTRUCTION CACHE

The ARM920T includes a 16KB instruction cache. The ICache has 512 lines of 32 bytes (8 words), arranged as a 64-way set-associative cache and uses modified virtual addresses, translated by CP15 register 13 (see Address translation on page 3-4), from the ARM9TDMI core.

The ICache implements allocate-on-read-miss. Random or round-robin replacement can be selected under software control via the RR bit (CP15 register 1, bit 14). Random replacement is selected at reset.

Instructions can also be locked in the ICache such that they cannot be overwritten by a linefill. This operates with a granularity of 1/64th of the cache, which is 64 words (256 bytes).

All instruction accesses are subject to MMU permission and translation checks. Instruction fetches which are aborted by the MMU will not cause linefills or instruction fetches to appear on the ASB.

For clarity, the I bit (bit 12 in CP15 register 1) is referred to as the Icr bit throughout the following text. The C bit from the MMU translation table descriptor corresponding to the address being accessed is referred to as Ctt.
INSTRUCTION CACHE ENABLE/DISABLE

On reset, the ICache entries are all invalidated and the ICache is disabled.

The ICache is enabled by writing 1 to the Icr bit, and disabled by writing 0 to the Icr bit.

The ICache is usually used with the MMU enabled, in which case the C bit in the relevant MMU translation table descriptor indicates whether an area of memory is cacheable. If the ICache is enabled with the MMU disabled, all instruction fetches are treated as cacheable.

When the ICache is disabled, the cache contents are ignored and all instruction fetches appear on the ASB as separate non-sequential accesses.

NOTES

ARM920T implements a non-sequential access on the ASB as an A-TRAN cycle followed by an S-TRAN cycle. It does not produce N-TRAN cycles.

If the cache is subsequently re-enabled its contents will be unchanged. If the contents are no longer coherent with main memory the ICache should be invalidated prior to being enabled (see Register 7: Cache operations on page 2-15).

The MMU and ICache can be enabled simultaneously by writing a 1 to bit 0 and to bit 12 in CP15 register 1 with a single MCR instruction.

INSTRUCTION CACHE OPERATION

If the ICache is disabled, each instruction fetch results in a separate non-sequential memory access on the ASB, giving very low performance to burst memory such as page mode DRAM or synchronous DRAM. Therefore, the ICache should be enabled as soon as possible after reset.

If the ICache is enabled, an ICache lookup is performed for each instruction fetch regardless of the setting of the Ctt bit in the relevant MMU translation table descriptor. If the required instruction is found in the cache, the lookup is called a cache hit. If the required instruction is not found in the cache, the lookup is called a cache miss.

If the instruction fetch is a cache hit and Ctt=1 indicating a cacheable region of memory, then the instruction is returned from the cache to the ARM9TDMI CPU core. If it is a cache miss and Ctt=1, then an 8-word linefill will be performed, possibly replacing another entry. The entry to be replaced, (called the victim), is chosen from the entries which are not locked using either a random or round-robin replacement policy.

If Ctt=0, indicating a non-cacheable region of memory, then a single non-sequential memory access will appear on the ASB.

NOTES

If Ctt=0, indicating a non-cacheable region of memory, then the cache lookup should result in a cache miss. The only way that it can result in a cache hit is if software has changed the value of the Ctt bit in the MMU translation table descriptor without invalidating the cache contents. This is a programming error, as the behavior in this case is architecturally unpredictable and varies between implementations.
INSTRUCTION CACHE REPLACEMENT ALGORITHM

The ICache and DCache replacement algorithm is selected by the RR bit in the CP15 control register (CP15 register 1, bit 14). Random replacement is selected at reset. Setting the RR bit to 1 selects round-robin replacement.

INSTRUCTION CACHE LOCKDOWN

Instructions can be locked into the ICache, causing the ICache to guarantee a hit, and providing optimum and predictable execution time.

Instructions are locked into the ICache by first ensuring the code to be locked is not already in the cache. This is tested by flushing either the whole ICache or specific lines. A short software routine can then be used to load the instructions into the ICache. The software routine must either be non-cacheable, or already in the ICache, but not in an ICache line which is about to be overwritten. The instructions to be loaded must be from a memory region which is cacheable.

The software routine operates by writing to CP15 register 9 to force the replacement counter to a specific ICache line and by using the prefetch ICache line operation available via CP15 register 7 to force the ICache to perform a lookup. This will miss and a linefill will be performed loading the cache line into the entry specified by the replacement counter. Once all the instructions have been loaded, they are then locked by writing to CP15 register 9 to set the replacement counter base to be one higher than the number of locked cache lines.

See Data cache lockdown on page 4-9 for a more complete explanation of cache locking.
DATA CACHE AND WRITE BUFFER

The ARM920T includes a 16KB data cache and a write buffer to reduce the effect of main memory bandwidth and latency on data access performance. The DCache has 512 lines of 32 bytes (8-words), arranged as a 64-way set-associative cache and uses virtual addresses from the ARM9TDMI CPU core. The write buffer can hold up to 16 words of data and 4 separate addresses. The operation of the data cache and write buffer are intimately connected.

The DCache supports write-through (WT) and writeback (WB) memory regions, controlled by the C and B bits in each section and page descriptor within the MMU translation tables. For clarity, these bits are referred to as Ctt and Btt in the following text. For details see Data cache and write buffer operation on page 4-6.

Each DCache line has two dirty bits, one for the first 4-words of the line, the other for the last 4-words, and a single virtual TAG address and valid bit for the entire 8-word line. The physical address from which each line was loaded is stored in the PA TAG RAM and is used when writing modified lines back to memory.

A linefill always loads a complete 8-word line.

When a store hits in the DCache, if the memory region is WB, the associated dirty bit is set marking the appropriate half-line as being modified. If the cache line is replaced due to a linefill, or if the line is the target of a DCache clean operation, the dirty bits are used to decide whether the whole, half, or none of the line is written back to memory. The line is written back to the same physical address from which it was loaded, regardless of any changes to the MMU translation tables.

The DCache implements allocate-on-read-miss. Random or round-robin replacement can be selected under software control via the RR bit (CP15 register 1, bit 14). Random replacement is selected at reset.

Data can also be locked in the DCache such that it cannot be overwritten by a linefill. This operates with a granularity of 1/64th of the cache, which is 64 words (256 bytes).

All data accesses are subject to MMU permission and translation checks. Data accesses which are aborted by the MMU will not cause linefills or data accesses to appear on the ASB.

For clarity, the C bit (bit 2 in CP15 register 1) is referred to as the Ccr bit throughout the following text.
DATA CACHE AND WRITE BUFFER ENABLE/DISABLE

On reset, all DCache entries are invalidated, the DCache is disabled, and the write buffer contents are discarded.

There is no explicit write buffer enable bit implemented in ARM920T. Situations in which the write buffer is used are described below.

The DCache is enabled by writing 1 to the Ccr bit, and disabled by writing 0 to the Ccr bit.

The DCache must be enabled only when the MMU is enabled. This is because the MMU translation tables define the cache and write buffer configuration for each memory region.

When the DCache is disabled the cache contents are ignored and all data accesses appear on the Advanced System Bus as separate non-sequential accesses. If the cache is subsequently re-enabled its contents will be unchanged. Depending on the software system design, the cache may need to be cleaned after it is disabled, and invalidated before it is re-enabled. See Cache coherence on page 4-10.

The MMU and DCache can be enabled or disabled simultaneously with a single MCR which changes bit 0 and bit 2 in the control register (CP15 register 1).

DATA CACHE AND WRITE BUFFER OPERATION

The DCache and write buffer configuration of each memory region is controlled by the C and B bits in each section and page descriptor in the MMU translation tables. For clarity, these bits are referred to as Ctt and Btt in the following text. The configuration is modified by the DCache enable bit in the CP15 control register, which is referred to as Ccr.

If the DCache is enabled, a DCache lookup is performed for each data access initiated by the ARM9TDMI CPU core, regardless of the value of the Ctt bit in the relevant MMU translation table descriptor. If the accessed virtual address matches the virtual address of an entry in the cache, the lookup is called a cache hit. If the required address does not match any entry in the cache, the lookup is called a cache miss. In this context a data access means any type of load (read) or store (write) or swap instruction, including LDR, LDRB, LDRH, LDM, LDC, STR, STRB, STRH, STC, SWP and SWPB.

To ensure that accesses appear on the ASB in program order, ARM920T will wait for all writes in the write buffer to complete on the ASB before starting any other ASB access. The ARM9TDMI CPU core can continue executing at full speed reading instructions and data from the caches and writing to the DCache and write buffer while buffered writes are being written to memory via the ASB.

Table 4-1 describes the DCache and write buffer behavior for each type of memory configuration. Ctt AND Ccr means the bitwise Boolean AND of Ctt with Ccr.
### Table 4-1. Data Cache and Write Buffer Configuration

<table>
<thead>
<tr>
<th>Ctt and Ccr</th>
<th>Btt</th>
<th>Data cache, write buffer and memory access behavior</th>
</tr>
</thead>
</table>
| 0 (1)      | 0   | Non-cached, non-buffered (NCNB)  
Reads and writes are not cached and always perform accesses on the ASB.  
Writes are not buffered. The CPU halts until the write is completed on the ASB.  
Cache hits should never occur. (2) |
| 0          | 1   | Non-cached buffered (NCB)  
Reads and writes are not cached, and always perform accesses on the ASB.  
Cache hits should never occur.  
Writes are placed in the write buffer and will appear on the ASB. The CPU continues execution as soon as the write is placed in the write buffer.  
Reads may be externally aborted.  
Writes can not be externally aborted. |
| 1          | 0   | Cached, write-through mode (WT)  
Reads which hit in the cache will read the data from the cache and do not perform an access on the ASB.  
Reads which miss in the cache cause a linefill.  
All writes are placed in the write buffer and will appear on the ASB. The CPU continues execution as soon as the write is placed in the write buffer.  
Writes which hit in the cache update the cache.  
Writes cannot be externally aborted. |
| 1          | 1   | Cached, write-back mode (WB)  
Reads which hit in the cache will read the data from the cache and do not perform an ASB access.  
Reads which miss in the cache cause a linefill.  
Writes which miss in the cache are placed in the write buffer and will appear on the ASB. The CPU continues execution as soon as the write is placed in the write buffer.  
Writes which hit in the cache update the cache and mark the appropriate half of the cache line as dirty, and do not cause an ASB access.  
Cache write-backs are buffered.  
Writes (Cache write-misses and cache write-backs) cannot be externally aborted. |

**NOTES:**

1. The control register C bit (Ccr) being zero disables all lookups in the cache, while the translation table descriptor C bit (Ctt) being zero only stops new data being loaded into the cache. With Ccr = 1 and Ctt = 0 the cache will still be searched on every access to check whether the cache contains an entry for the data.

2. It is an operating system software error if a cache hit occurs when reading from, or writing to, a region of memory marked as NCNB or NCB. The only way this can occur is if the operating system changes the value of the C and B bits in a page table descriptor, while the cache contains data from the area of virtual memory controlled by that descriptor. The cache and memory system behavior resulting from changing the page table descriptor in this way is unpredictable. If the operating system needs to change the C and B bits of a page table descriptor, it must ensure that the caches do not contain any data controlled by that descriptor. In some circumstances, the operating system may need to clean and flush the caches to ensure this.
A linefill performs an 8-word burst read from the ASB and places it as a new entry in the cache, possible replacing another line at the same location within the cache. The location which is replaced (called the victim) is chosen from the entries which are not locked using either a random or round-robin replacement policy. If the cache line being replaced is marked as dirty, indicating that it has been modified and that main memory has not been updated to reflect the change, a cache writeback occurs.

Depending on whether one or both halves of the cache line are dirty, the writeback will perform a 4 or 8-word sequential burst write access on the ASB. The writeback data is placed in the write buffer and then the linefill data is read from the ASB. The CPU can continue while the writeback data is written to memory via the ASB.

Load multiple (LDM) instructions accessing NCNB or NCB regions perform sequential bursts on the ASB. Store multiple (STM) instructions accessing NCNB regions also perform sequential bursts on the ASB.

The sequential burst will be split into two bursts if it crosses a 1KB boundary. This is because the smallest MMU protection and mapping size is 1KB, so the memory regions on each size of the 1KB boundary may have different properties.

This means that no sequential access generated by ARM920T will cross a 1KB boundary, which can be exploited to simplify memory interface design. For example, a simple page mode DRAM controller could perform a page-mode access for each sequential access, provided the DRAM page size is 1KB or larger.

See also Cache coherence on page 4-10.

DATA CACHE REPLACEMENT ALGORITHM

The DCache and ICache replacement algorithm is selected by the RR bit in the CP15 Control register (CP15 register 1, bit 14). Random replacement is selected at reset. Setting the RR bit to 1 selects round-robin replacement.

SWAP INSTRUCTIONS

Swap instruction (SWP or SWPB) behavior is dependent on whether the memory region is cacheable or non-cacheable.

Swap instructions to cacheable regions of memory are useful for implementing semaphores or other synchronization primitives in multithreaded uniprocessor software systems.

Swap instructions to non-cacheable memory regions are useful for synchronization between two bus masters in a multi-master bus system. This could be two processors, or a processor and a DMA controller.

When a swap instruction accesses a cacheable region of memory (WT or WB), the DCache and write buffer behavior will be the same as having a load followed by a store according to the normal rules described. The BLOK pin will not be asserted during the execution of the instruction. It is guaranteed that no interrupt can occur between the load and store portions of the swap.

When a swap instruction accesses a non-cacheable (NCB or NCNB) region of memory, the write buffer is drained, and a single word or byte will be read from the ASB. The write portion of the swap will then be treated as non-bufferable, regardless of the value of Btt, and the processor stalled until the write is completed on the ASB. The BLOK pin will be asserted to indicate that the read and write should be treated as an atomic operation on the bus.

Like all other data accesses, a swap to a non-cacheable region which hits in the cache indicates a programming error.
DATA CACHE ORGANIZATION

The DCache is organized as 8 segments, each containing 64 lines, and each line containing 8-words. The line’s position within its segment is a number from 0 to 63 which is called the index. A line in the cache can be uniquely identified by its segment and index. The index is independent of the line’s virtual address. The segment is selected by bits [7:5] of the virtual address of the line.

Bits [4:2] of the virtual address specify which word within a cache line is accessed. For halfword operations, bit [1] of the virtual address specifies which halfword is accessed within the word. For byte operations, bits [1:0] specify which byte within the word is accessed.

Bits [31:8] of the virtual address of the each cache line is called the TAG. The virtual address TAG is stored in the cache along with the 8-words of data, when the line is loaded by a linefill.

Cache lookups compare bits [31:8] of the modified virtual address of the access with the stored TAG to determine whether the access is a hit or miss. The cache is therefore said to be virtually addressed.

DATA CACHE LOCKDOWN

Data can be locked into the DCache causing the DCache to guarantee a hit, and providing optimum and predictable execution time.

When no data is locked in the DCache, and a linefill occurs, the replacement algorithm chooses a victim cache line to be replaced by selecting an index in the range (0 to 63). The segment is specified by bits [7:5] of the virtual address of the data access which missed.

Data is locked into the DCache by restricting the range of victim numbers produced by the replacement algorithm, so that some cache lines are never selected as victims. The base pointer for the DCache victim generator can be set by writing to CP15 register 9. The replacement algorithm chooses a victim cache line in the range (base to 63), locking in the cache the lines with index in the range (0 to base - 1).

Data is loaded and locked into the DCache by first ensuring the data to be locked is not already in the cache. This can be ensured by cleaning and flushing either the whole DCache or specific lines. A short software routine can then be used to load the data into the DCache.

The software routine to load the data operates by writing to CP15 register 9 to force the replacement counter to a specific DCache line and then executing a load instruction to perform a cache lookup. This will miss and a linefill will be performed, bringing 8 words of data into the cache line specified by the replacement counter, in the segment specified by bits [7:5] of the modified virtual address accessed by the load.

To load further lines into the cache, the software routine can loop performing one load from each line to be loaded. As each line contains 8 words, each loop should add 32 (bytes) to the load address. The software routine needs to move the victim counter to the next index after it has loaded a line into the last available segment with the current index. As there are 8-segments, this will occur after 8-cache lines have been loaded.

Once all the data has been loaded, it is locked by writing to CP15 register 9 to move the replacement counter base to be one higher than the highest index of the locked cache lines.

The software routine that loads and locks the data in the DCache can be located in a cacheable region of memory providing it does not contain any loads or stores other than the loads which are used to bring the data to be locked into the DCache. The data to be loaded must be from a memory region which is cacheable.
CACHE COHERENCE

The ICache and DCache contain copies of information normally held in main memory. If these copies of memory information get out of step with each other because one is updated and the others are not updated, they are said to have become incoherent. If the DCache contains a line which has been modified by a store or swap instruction, and the main memory has not been updated, the cache line is said to be dirty. Clean operations force the cache to write dirty lines back to main memory.

On the ARM920T, software is responsible for maintaining coherence between main memory, the ICache and the DCache.

Register 7: Cache operations on page 2-15 describes facilities for invalidating the entire ICache or individual ICache lines, and for cleaning or invalidating DCache lines, or for invalidating the entire DCache.

To clean the entire DCache efficiently, software should loop though each cache entry using the clean D single entry (using index) operation or the clean and invalidate D entry (using index) operation. This should be performed by a two-level nested loop going though each index value for each segment. See Data cache organization on page 4-9.

DCache, ICache, and memory coherence is generally achieved by:

- cleaning the DCache to ensure memory is up to date with all changes
- invalidating the ICache to ensure that the ICache is forced to re-load instructions from memory.

Software can minimize the performance penalties of cleaning and invalidating caches by:

- Cleaning only small portions of the cache when only a small area of memory needs to be made coherent, for example, when updating an exception vector entry.
- Invalidating only small portions of the ICache when only a small number of instructions are modified, for example, when updating an exception vector entry.
- Not invalidating the ICache in situations where it is known that the modified area of memory cannot be in the cache, for example, when mapping a new page into the currently running process.

The ICache needs to be made coherent with a changed area of memory after any changes to the instructions which appear at a virtual address, and before the new instructions are executed.

Dirty data in the DCache can be pushed out to main memory by cleaning the cache.
Situations which necessitate cache cleaning and invalidating include:

- writing instructions to a cacheable area of memory using STR or STM instructions, for example:
  - self-modifying code
  - JIT compilation
  - copying code from another location
  - downloading code via the EmbeddedICE JTAG debug features
  - updating an exception vector entry.

- another bus master, such as a DMA controller, modifying a cacheable area in main memory
- turning the MMU on or off
- changing the virtual-to-physical mappings in the MMU page tables
- turning the ICache or DCache on, if its contents are no longer coherent.

The DCache should be cleaned, and both caches invalidated, before the cache and write buffer configuration of an area of memory is changed by modifying Ctt or Btt in the MMU translation table descriptor. This is not necessary if it is known that the caches cannot contain any entries from the area of memory whose translation table descriptor is being modified.

Changing the process ID in CP15 register 13 does not change the contents of the cache or memory, and does not affect the mapping between cache entries and physical memory locations. It only changes the mapping between ARM9TDMI addresses and cache entries. This means that changing the process ID does not lead to any coherency issues. No cache cleaning or cache invalidation is required when the process ID is changed.

At reset the DCache and ICache entries are all invalidated and the DCache and ICache are disabled.

The software design also needs to consider that the pipelined design of the ARM9TDMI core means that it fetches three instructions ahead of the current execution point. So, for example, the three instructions following an MCR which invalidates the ICache, will have been read from the ICache before it is invalidated.
CACHE CLEANING WHEN LOCKDOWN IS IN USE

The clean D single entry (using index) and clean and invalidate D entry (using index) operations can leave the victim pointer set to the index value used by the operation. In some circumstances, if DCache locking is in use, this could leave the victim pointer in the locked region, leading to locked data being evicted from the cache. The victim pointer can be moved outside the locked region by implementing the cache loop enclosed by the reading and writing of the Base and Victim pointer:

MRC p15, 0, Rd, c9, c0, 0 ; Read D Cache Base into Rd

Index Clean or Index Clean and Invalidate loops

MCR p15, 0, Rd, c9, c0, 0 ; Write D Cache Base and Victim from Rd

Clean D single entry (using VA) and clean and invalidate D entry (using VA) operations do not move the victim pointer, so there is no need to reposition the victim pointer after using these operations.

IMPLEMENTATION NOTES

This section describes the behavior of the ARM920T implementation in areas which are architecturally unpredictable. For portability to other ARM implementations, software should not depend on this behavior.

A read from a non-cacheable (NCB or NCNB) region which unexpectedly hits in the cache will still read the required data from the ASB. The contents of the cache will be ignored, and the cache contents will not be modified. This includes the read portion of a swap (SWP or SWPB) instruction.

A write to a non-cacheable (NCB or NCNB) region which unexpectedly hits in the cache will update the cache and will still cause an access on the ASB.

PHYSICAL ADDRESS TAG RAM

The ARM920T implements a PA TAG RAM in order to perform write backs from the data cache.

A write back occurs when dirty data that is about to be overwritten by linefill data comes from a memory region that is marked as a write back region. This data is written back to main memory to maintain memory coherency.

Dirty data is data that has been modified in the cache, but not updated in main memory.

When a line is written into the data cache, the physical address TAG (DPA[31:5]) is written into the PA TAG RAM. If this line comes to be written back to main memory, the PA TAG RAM is indexed into by the data cache and the physical address (WBPA[31:0]) is returned to the AMBA Bus interface so that it can perform the write back.

The PA TAG RAM Array for a 16k data cache comprises 8 segments x 64 rows/segment x 26 bits/row. There are two test interfaces to the PA TAG RAM:

Debug interface, see Scan chain 4 - debug access to the PA TAG RAM

AMBA test interface, see PA TAG RAM test
APPENDIX 5

CLOCK MODES

OVERVIEW

The ARM920T has two functional clock inputs, BCLK and FCLK. Internally, the ARM920T is clocked by GCLK, which can be seen on the CPCLK output as shown in Figure 5-1. GCLK can be sourced from either BCLK or FCLK depending on the clocking mode, selected using nF bit and iA bit in CP15 register 1 (see Register 1: Control register on page 2-10). The three clocking modes are FastBus, synchronous and asynchronous.

The ARM920T is a static design and both clocks can be stopped indefinitely without loss of state. From Figure 5-1 it can be seen that some of the ARM920T macrocell signals will have timing specified with relation to GCLK, which can be either FCLK or BCLK depending on the clocking mode.

Figure 5-1. ARM920T Clocking
FASTBUS MODE

In this mode of operation the BCLK input is the source for GCLK. The FCLK input is ignored. This mode is typically used in systems with high speed memory.

SYNCHRONOUS MODE

This mode is typically used in systems with low speed memory. In this mode GCLK can be sourced from BCLK and FCLK. BCLK is used to control the AMBA memory interface. FCLK is used to control the internal ARM9TDMI processor core and any cache operations. FCLK must have a higher frequency and must also be an integer multiple of BCLK, with a BCLK transition only when FCLK is HIGH. An example is shown in Figure 5-2.

If the ARM920T performs an external access, for example, a cache linefill, the ARM920T will switch to BCLK to perform the access. The delay when switching from FCLK to BCLK is a minimum of one FCLK phase and a maximum of one BCLK cycle. An example of the clock switching is shown in Figure 5-3. The delay when switching from BCLK to FCLK is a maximum of one FCLK phase.

Care must be taken if BCLK is stopped by the system so that when BCLK is restarted it does not violate any of the above restrictions.
ASYNCHRONOUS MODE

This mode is typically used in systems with low speed memory. In this mode of operation GCLK can be sourced from BCLK and FCLK. BCLK is used to control the AMBA memory interface. FCLK is used to control the internal ARM9TDMI processor core and any cache operations. The one restriction is that FCLK must have a higher frequency than BCLK. An example is shown in Figure 5-4.

![Figure 5-4. Asynchronous Clocking Mode](image)

If the ARM920T performs an external access, for example, a cache miss or a cache line fill, ARM920T will switch to BCLK to perform the access. The delay when switching from FCLK and BCLK is a minimum of one BCLK cycle, and a maximum of one and a half BCLK cycles. An example of the clock switching is shown in Figure 5-4. When switching from BCLK to FCLK the minimum delay is one FCLK cycle and the maximum delay is one and a half FCLK cycles. An example of the clock switching is shown in Figure 5-5.

![Figure 5-5. Switching from FCLK to BCLK in Asynchronous Mode](image)
S3C SERIES MASK ROM ORDER FORM

Product description:
Device Number:  S3C__________ - ___________(write down the ROM code number)
Product Order Form:  [ ] Package  [ ] Pellet  [ ] Wafer  Package Type:  __________

Package Marking (Check One):
[ ] Standard  [ ] Custom A  (Max 10 chars)
[ ] Custom B  (Max 10 chars each line)

SEC  @ YWW
Device Name  @ YWW
Device Name  @ YWW

@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

Delivery Dates and Quantities:

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Required Delivery Date</th>
<th>Quantity</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM code</td>
<td></td>
<td>Not applicable</td>
<td>See ROM Selection Form</td>
</tr>
<tr>
<td>Customer sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Risk order</td>
<td></td>
<td></td>
<td>See Risk Order Sheet</td>
</tr>
</tbody>
</table>

Please answer the following questions:

❖ For what kind of product will you be using this order?
[ ] New product  [ ] Upgrade of an existing product
[ ] Replacement of an existing product  [ ] Other

If you are replacing an existing product, please indicate the former product name

( )

❖ What are the main reasons you decided to use a Samsung microcontroller in your product?
Please check all that apply.
[ ] Price  [ ] Product quality  [ ] Features and functions
[ ] Development system  [ ] Technical support  [ ] Delivery on time
[ ] Used same micom before  [ ] Quality of documentation  [ ] Samsung reputation

Mask Charge (US$ / Won):  __________________________________________

Customer Information:
Company Name:  ____________________  Telephone number  ____________________

Signatures:  __________________________________________
(Person placing the order)  (Technical Manager)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)
S3C SERIES
REQUEST FOR PRODUCTION AT CUSTOMER RISK

Customer Information:
Company Name: ________________________________________________________________
Department: ________________________________________________________________
Telephone Number: __________________________ Fax: _____________________________
Date: __________________________

Risk Order Information:
Device Number: S3C________ - ________ (write down the ROM code number)
Package: Number of Pins: ____________ Package Type: _____________________
Intended Application: ________________________________________________________________
Product Model Number: __________________________________________________________

Customer Risk Order Agreement:
We hereby request SEC to produce the above named product in the quantity stated below. We believe our risk order product to be in full compliance with all SEC production specifications and, to this extent, agree to assume responsibility for any and all production risks involved.

Order Quantity and Delivery Schedule:
Risk Order Quantity: _____________________ PCS

Delivery Schedule:

<table>
<thead>
<tr>
<th>Delivery Date (s)</th>
<th>Quantity</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signatures: ________________________________________________________________
(Person Placing the Risk Order) ____________________________________________
(SEC Sales Representative)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)
S3C2410X MASK OPTION SELECTION FORM

Device Number: S3C2410X - __________ (write down the ROM code number)

Attachment (Check one): 
- [ ] Diskette
- [ ] PROM

Customer Checksum: ________________________________________________

Company Name: ____________________________________________________

Signature (Engineer): ________________________________________________

Please answer the following questions:

Application (Product Model ID: ______________________)
- [ ] Audio
- [ ] Video
- [ ] Telecom
- [ ] LCD Databank
- [ ] Caller ID
- [ ] LCD Game
- [ ] Industrials
- [ ] Home Appliance
- [ ] Office Automation
- [ ] Remocon
- [ ] Other

Please describe in detail its application

______________________________________________________________________