The TMS320C6000 Multi-channel Buffered Serial Port (McBSP) is designed to interface to a device that supports synchronous Serial Peripheral Interface (SPI). This document describes the hardware interface between the McBSP and a SPI ROM. The McBSP operates as the master in a user-specified clock stop (CLKSTP) mode in order to communicate with the SPI ROM. The McBSP initialization and control register programming is also discussed.
Contents

TMS320C6000 McBSP interface to SPI ROM ................................................................. 3
Solution ....................................................................................................................... 3
Pin Configuration ...................................................................................................... 3
McBSP Initialization .................................................................................................. 4
Timing Analysis ....................................................................................................... 6

Figures

Figure 1. McBSP Master interface to SPI slave device ................................................. 3
Figure 2. Receive Control Register (RCR for SPI Master) ............................................ 4
Figure 3. Transmit Control Register (XCR for SPI Master) ......................................... 5
Figure 4. Sample Rate Generator Register (SRGR for SPI Master) ................................. 5
Figure 5. Pin Control Register (PCR for SPI Master) ..................................................... 5
Figure 6. Serial Port Control Register (SPCR for SPI Master) ...................................... 5
Figure 7. Clock Stop Mode Options ........................................................................... 6
Figure 8. C6201 SPI Master Timing, CLKSTP = 11b, CLKXP=0 ................................... 6

Tables

Table 1. McBSP Register values for 200MHz CPU clock .......................................... 5
Table 2. Timing Numbers for McBSP as SPI Master .................................................. 7
Table 3. Timing Analysis for SPI Master and Slave ..................................................... 7
TMS320C6000 McBSP interface to SPI ROM

Design Problem

How do I interface the Serial Peripheral Interface (SPI™) ROM to the TMS320C6201?

Solution

The multi-channel buffered serial port (McBSP) in the TMS320C6000 interfaces to a SPI ROM with no glue logic. A SPI system is typically a 4-wire interface comprising serial data in, serial data out, serial clock, and device select. The McBSP provides this 4-wire interface via DR, DX, CLXX, and FSX pins, respectively.

The McBSP supports the SPI interface for a synchronous, full-duplex, variable element length (element length is fixed for a given transfer), master or slave mode back-to-back transmission and reception. This feature is achieved by using the clock-stop (CLKSTP) mode of the McBSP. This document discusses the McBSP interface to an Atmel™ SPI serial CMOS EEPROM, which can only be a slave. The McBSP as a SPI master, generates the required control signals and clocking to the slave.

Pin Configuration

For the McBSP to master the interface, you must configure CLXX and FSX pins of the serial port as outputs only. CLXX can be generated either via the ‘C6000 CPU clock or via an external clock source input on the CLKS pin. In SPI mode, a SPI system clock or any other clock source can drive CLKS if present. The clock divide down can be programmed as per application needs.

Figure 1. McBSP Master interface to SPI slave device

The signal connectivity shown in Figure 1 is for connecting a Atmel™ SPI serial CMOS EEPROM AT25 series which has a maximum clock rate of 2.1 MHz for Vcc range from 2.7V to 5.5V. This slave device is organized as 1k/2k/4k/8k of 8-bit data and only supports SPI modes 0 and 3.
As shown in Figure 1, CLKSTP scheme supports back-to-back transmission and reception utilizing signals that correspond to the transmitter. The McBSP also simultaneously receives data by connecting the CLKX and FSX outputs as CLKR and FSR signals internally. As a good practice, CLKR and FSR should be programmed as inputs.

McBSP Initialization

The various McBSP control registers shown in Figures 2 through 6 have to be initialized for SPI operation. The serial port initialization procedure for SPI mode is as follows:

1. If McBSP is not in reset state, set /XRST = /RRST = 0 in SPCR.
2. Program the McBSP configuration registers XCR, RCR, SRGR, PCR, and SPCR for all parameters as required except for CLKSTP bits in SPCR.
3. Set /GRST=1 in SPCR to get the sample rate generator out of reset.
4. Wait two CLKG clocks for the McBSP to re-initialize.
5. Write the desired value into the CLKSTP bit-fields in the SPCR. Figure 7 shows the various CLKSTP modes that are supported by the McBSP.
6. Either (a) or (b) should be followed.

(a) This step should be performed if the CPU is used to service the McBSP. Set /XRST = /RRST = 1 to enable the serial port. Note that the value written to the SPCR at this time should have only the reset bits changed to 1 and the remaining bit-fields should have the same value as in Step 2 and 4 above.

(b) If DMA is used to perform data transfers, it should first be initialized with the appropriate read/write syncs, src/dst addresses and their update modes, transfer complete interrupt, and any other feature suitable for the application. Lastly, set the START bit. The DMA is now in the START state and waits for the synchronization events to occur. Then, pull the McBSP out of reset. For details on DMA initialization for servicing the McBSP, refer to TMS320C6000 McBSP Initialization, and TMS320C6000 DMA Applications note.

Figure 2. Receive Control Register (RCR for SPI Master)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>24</th>
<th>23</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>01</td>
</tr>
</tbody>
</table>

RPHASE  RFRLEN2  RWDLEN2  RCOMPAND  RFIG  RDATDLY

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>8</th>
<th>7</th>
<th>5</th>
<th>4</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>0</td>
</tr>
</tbody>
</table>

reserved  RFRLEN1  RWDLEN1  reserved
Figure 3. Transmit Control Register (XCR for SPI Master)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>24</th>
<th>23</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>XPHASE</td>
<td>XFRLEN2</td>
<td>XWDLEN2</td>
<td>XCOMPAND</td>
<td>XFIG</td>
<td>XDATDLY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4. Sample Rate Generator Register (SRGR for SPI Master)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| GSYNC | CLKSP | CLKSM | FSGM | FPER |
| 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 0  | 0  | 0  | 0  |
| FWID | CLKGDV |

Figure 5. Pin Control Register (PCR for SPI Master)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    | 0  | 1  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  |
| rsv | XIOEN | RIOEN | FSXM | FSRM | CLKXM | CLKRM | rsv | CLKS_STAT | DX_STAT | DR_STAT | FSXP | FSRP | CLKXP | CLKRP |

Figure 6. Serial Port Control Register (SPCR for SPI Master)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| R, +0 | FRST- | GRST- | XINTM | XSYNCRERR | XEMPTY- | XRDY | XRST- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| 0  | 0  | 0  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| DLB | RJUST | CLKSTP | reserved | reserved | reserved | RINTM | RSYNERR | RFULL | RRDY | RRST- |

Table 1. McBSP Register values for 200MHz CPU clock

<table>
<thead>
<tr>
<th>Register</th>
<th>Value in hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCR</td>
<td>0x00010000</td>
<td>single phase, one 8-bit element per frame, one bit-clock delay</td>
</tr>
<tr>
<td>XCR</td>
<td>0x00010000</td>
<td>single phase, one 8-bit element per frame, one bit-clock delay</td>
</tr>
</tbody>
</table>
| SRGR     | 0x0200005F   | • serial clock CLKX generated by CPU clock (CLKSM=1)  
|          |              | • frame sync FSX generated due to DXR-to-XSR transfer (FSGM=0)  
|          |              | • clock divide down is 95 for 200 MHz clock to generate 2.08  
|          |              | MHz shift clock (CLKGDV=0x5F) |
| PCR      | 0x00000A0C   | • FSX is an active low (FSXP=1) output (FSXM=1)  
|          |              | • FSR is an active low (FSRP=1) input (FSRM=0)  
|          |              | • CLKX is an output (CLKXM=1) and starts with a rising  
|          |              | edge (CLKXP=0) |
| SPCR[12:10] | 0x3  | CLKSTP=11b. Since CLKXP=0, this refers to data transmitted  
|          |              | on rising edge and received on falling edge of CLKX by the  
|          |              | master. This parameter can be changed as per application  
|          |              | needs. |
The example code in Appendix A initializes McBSP0 in the correct order for SPI-mode communication between the McBSP and a SPI serial EEPROM.

**Figure 7. Clock Stop Mode Options**

<table>
<thead>
<tr>
<th>Clock Configuration</th>
<th>Waveform</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKX (CLKSTP=10b, CLKXP=0)</td>
<td><img src="image1" alt="Waveform" /></td>
</tr>
<tr>
<td>CLKX (CLKSTP=11b, CLKXP=0)</td>
<td><img src="image2" alt="Waveform" /></td>
</tr>
<tr>
<td>CLKX (CLKSTP=10b, CLKXP=1)</td>
<td><img src="image3" alt="Waveform" /></td>
</tr>
<tr>
<td>CLKX (CLKSTP=11b, CLKXP=1)</td>
<td><img src="image4" alt="Waveform" /></td>
</tr>
</tbody>
</table>

**Timing Analysis**

SPI mode (0,0) of the SPI ROM corresponds to the McBSP SPI mode with CLKSTP=11b and CLKXP=0. The master (McBSP) shifts data out on the falling edge of CLKX and the slave (SPI ROM) samples the receive data on the rising edge of CLKX. The slave transmits/shifts data out on the falling edge of CLKX and the master samples the receive data on the rising edge of CLKX.

**Figure 8. C6201 SPI Master Timing, CLKSTP=11b, CLKXP=0**

The timing diagram for CLKSTP=11b, and CLKXP=0 is shown in Figure 8. The corresponding values for the timing requirements and switching characteristics for a 2.1MHz operation is shown in Table 2. The values are derived from the formula/numbers available in the TMS320C6201 datasheet.
### Table 2. Timing Numbers for McBSP as SPI Master

<table>
<thead>
<tr>
<th>NO.</th>
<th>Switching Characteristics</th>
<th>Min</th>
<th>Max</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>th(CKXH-FXL) Hold time, FSX low after CLKX high</td>
<td>L-2 = 238</td>
<td>L+2 = 242</td>
<td>ns</td>
</tr>
<tr>
<td>2</td>
<td>td(FXL-CKXL) Delay time, FSX low to CLKX low</td>
<td>T-2 = 478</td>
<td>T+2 = 480</td>
<td>ns</td>
</tr>
<tr>
<td>3</td>
<td>td(CKXH-DXV) Delay time, CLKX high to DX valid</td>
<td>-2</td>
<td>3</td>
<td>ns</td>
</tr>
<tr>
<td>6</td>
<td>tdis(CKXH-DXHZ) Disable time, DX high impedance following last data bit from CLKX high</td>
<td>-2</td>
<td>3</td>
<td>ns</td>
</tr>
<tr>
<td>7</td>
<td>td(FXL-DXV) Delay time, FSX low to DX valid</td>
<td>H-2 = 238</td>
<td>H+3 = 243</td>
<td>ns</td>
</tr>
</tbody>
</table>

### Timing Requirements

| 4   | tsu(DRV-CKXL) Setup time, DR valid before CLKX low | 12        | ns       |
| 5   | th(CKXL-DRV) Hold time, DR valid after CLKX low    | 4         | ns       |

**NOTE:** The following is true for the above calculations:

1. Since CLKGDV = 95, CLKX derived from CPU clock will have a 50% duty cycle and therefore H=L.
2. Period of CLKX, T = (1+CLKGDV) * P where P = 5ns for a 200MHz CPU clock. Hence, T = 480 ns.

As shown in Table 3, the timing numbers of the McBSP match with that of the SPI ROM with sufficient timing margins. Note that the SPI ROM timings correspond to the 2.7-5.5V range of devices. Therefore a voltage translation buffer (for example, SN54LVT16373) will have to be used between the McBSP and the SPI ROM. Use of buffers will still meet the necessary timing requirements/margins.

### Table 3. Timing Analysis for SPI Master and Slave

<table>
<thead>
<tr>
<th>SPI ROM Timing Requirements</th>
<th>C6201 Switching Characteristics</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsu (min) Data In Setup Time</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>th (min) Data In Hold Time</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>tcss (min) /CS Setup time</td>
<td>250</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C6201 Timing Requirements</th>
<th>SPI ROM Switching Characteristics</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsu(DRV-CKXL)min Data In Setup Time</td>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>th(CKXL-DRV)min Data In Hold Time</td>
<td>4</td>
<td>240</td>
</tr>
</tbody>
</table>

For applications where the McBSP is used as a SPI slave, please ensure that the internal clock, CLKG runs at least eight times that of the master clock. Typically, programming CLKGDV=1 and using CPU clock (CLKSM=1) (when McBSP is a SPI slave) should suffice since SPI clocks are very slow.
Appendix A

TI Proprietary Information

spi3_dma.c:

Tests SPI CLKSTP mode where CLKSTP=11b and CLKXP=0. McBSP0 is the master. Any other SPI-compliant device can be a slave.

#include "common.h"

#define M0TO1 TRUE
/* NOTE TRUE MEANS EXTERNAL FOR SLAVE AND INTERNAL FOR MSTR */
#define M0TO1_MSTR M0TO1
#define XFER_SIZE 5
#define XFER_TYPE DMA_SPI
#define CLKGDV0 15
#define CLKSM0 CLK_MODE_CPU

void init_mcsp0_master(void);
void init_M0_srgr(void);
void set_clkstp(void);

void main(void)
{
  int xfer_size, xfer_type;
  int mcsp0tol;
  int mcsp0tol_rate;

  recv0_done = FALSE;
  xmit0_done = FALSE;

  mcsp0tol = M0TO1;
  mcsp0tol_rate = (M0TO1_MSTR * (CLKGDV0 + 1) * 10) / (9 + CLKSM0);
  xfer_size = XFER_SIZE;
  xfer_type = XFER_TYPE;

  init_M0_srgr();
  init_mcsp0_master();

  /* Enable sample rate generator; /GRST=1 */
  MCBSP_SAMPLE_RATE_ENABLE(0);
  MCBSP_SAMPLE_RATE_ENABLE(1);

  count_n_cpu_cycles(mcsp0tol_rate * 2); /* Wait for 2 bit clocks */
  set_clkstp(); /* set CLKSTP after pulling out of grst= */

  switch (xfer_type) {
    case DMA_SPI: 
      dma_reset();
      set_interru ts(); /* Initialize DMA to service McBSP */
      if (mcsp0tol) {

DMA2_SRC_ADDR = (unsigned int) out0;
DMA2_DEST_ADDR = MCBSP_DXR_ADDR(0);
REG_WRITE (DMA2_XFER_COUNTER_ADDR, xfer_size);
LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_ESIZE32, ESIZE, ESIZE_SZ);
LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_ADDR_INC, SRC_DIR, SRC_DIR_SZ);
LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_DMA_PRI, PRI, 1);
DMA_START(DMA2_PRIMARY_CTRL_ADDR, SEN_XEVT0, WSYNC, WSYNC_SZ);
DMA2_START(DMA_CH2);

DMA1_SRC_ADDR = (unsigned int) in0;
DMA1_DEST_ADDR = MCBSP_DRR_ADDR(0);
REG_WRITE (DMA1_XFER_COUNTER_ADDR, xfer_size);
LOAD_FIELD (DMA1_PRIMARY_CTRL_ADDR, DMA_ESIZE32, ESIZE, ESIZE_SZ);
LOAD_FIELD (DMA1_PRIMARY_CTRL_ADDR, DMA_ADDR_INC, SRC_DIR, SRC_DIR_SZ);
LOAD_FIELD (DMA1_PRIMARY_CTRL_ADDR, DMA_DMA_PRI, PRI, 1);
LOAD_FIELD (DMA1_PRIMARY_CTRL_ADDR, SEN_REVT0, RSYNC, RSYNC_SZ);
DMA1_START(DMA_CH1);

} // end of while loop

SET_BIT (MCBSP_SPCR_ADDR(0), RRST);
SET_BIT (MCBSP_SPCR_ADDR(0), XRST);
count_n_cpu_cycles(mcs0to1_rate * 2);
while (!xmit0_done || !recv0_done);
brk;

CSR |= 0x00007000;      /* PowerDown PD3 to shut off MCSP */

void
init_mcsp0_master(void)
{
  /* PCR setup*/
  LOAD_FIELD (MCBSP_PCR_ADDR(0), M0TO1_MSTR, CLKXM, 1);
  LOAD_FIELD (MCBSP_PCR_ADDR(0), !M0TO1_MSTR, CLKRM, 1);
  LOAD_FIELD (MCBSP_PCR_ADDR(0), CLKX_POL_FALLING, CLKXP, 1);
  LOAD_FIELD (MCBSP_PCR_ADDR(0), FSYNC_POL_LOW, FSXP, 1);
  LOAD_FIELD (MCBSP_PCR_ADDR(0), FSYNC_MODE_INT, FSXM, 1);
  LOAD_FIELD (MCBSP_PCR_ADDR(0), FSYNC_MODE_EXT, FSRM, 1);

  /* SRGR setup */
  LOAD_FIELD (MCBSP_SRGR_ADDR(0), FSX_DXR_TO_XSR, FSGM, 1);

  /* XCR setup */
  LOAD_FIELD (MCBSP_XCR_ADDR(0), SINGLE_PHASE, XPHASE, 1);
  LOAD_FIELD (MCBSP_XCR_ADDR(0), WORD_LENGTH_20, XWDLEN1, XWDLEN1_SZ);
  LOAD_FIELD (MCBSP_XCR_ADDR(0), 0, XFRLEN1, XFRLEN1_SZ);
  LOAD_FIELD (MCBSP_XCR_ADDR(0), DATA_DELAY1, XDATDLY, XDATDLY_SZ);
  LOAD_FIELD (MCBSP_XCR_ADDR(0), NO_COMPAND_MSB_1ST, XCOMPAND, XCOMPAND_SZ);

  /* RCR setup */
  LOAD_FIELD (MCBSP_RCR_ADDR(0), SINGLE_PHASE, RPHASE, 1);
  LOAD_FIELD (MCBSP_RCR_ADDR(0), WORD_LENGTH_20, RWDLEN1, RWDLEN1_SZ);
  LOAD_FIELD (MCBSP_RCR_ADDR(0), 0, RFRLEN1, RFRLEN1_SZ);
  LOAD_FIELD (MCBSP_RCR_ADDR(0), DATA_DELAY1, RDATDLY, RDATDLY_SZ);
  LOAD_FIELD (MCBSP_RCR_ADDR(0), NO_COMPAND_MSB_1ST, RCOMPAND, RCOMPAND_SZ);

  /* SPCR */
void init_M0_srgr(void)
{
    LOAD_FIELD (MCBSP_SRGR_ADDR(0), CLKGDV0, CLKGDV, CLKGDV_SZ);
    LOAD_FIELD (MCBSP_SRGR_ADDR(0), CLKSM0, CLKSM, 1);
    LOAD_FIELD (MCBSP_SRGR_ADDR(0), CLKS_POL_RISING, CLKSP, 1);
    LOAD_FIELD (MCBSP_SRGR_ADDR(0), GSYNC_OFF, GSYNC, 1);
}

void set_clkstp(void)
{
    /* SPCR */
    LOAD_FIELD (MCBSP_SPCR_ADDR(0), CLKSTP_DELAY, CLKSTP, CLKSTP_SZ);
}

void set_interrupts(void)
{
    intr_init();
    INTR_MAP_RESET();
    /* Hook interrupt service routine to an interrupt */
    intr_hook (c_int11, CPU_INT11);
    intr_hook (c_int09, CPU_INT9);

    INTR_ENABLE(CPU_INT_NMI); /* Enable NMIE */
    INTR_GLOBAL_ENABLE(); /* Set GIE in CSR*/
    INTR_ENABLE(11);
    INTR_ENABLE(9);
    return;
}

interrupt void c_int11(void) /* DMA ch2 */
{
    xmit0_done = TRUE;
    return;
}

interrupt void c_int09(void) /* DMA ch1 */
{
    recv0_done = TRUE;
    return;
}

/* ================================= */
/* DMA DATA TRANSFER COMPLETION ISRS */
/* ================================= */
* count.asm

.global _count_n_cpu_cycles

.text

* CLEARS ALL REGISTERS FOR SIMULATION PURPOSES

_count_n_cpu_cycles:

; counts n cycles where n has a minimum of 15
; A4 gets the value passed from main code (C)
SUB .L1 A4, 15, A1
|| B .S2 LOOP

B .S2 LOOP
|| CMPLT .L1 A1, 0, A2

B .S2 LOOP
|| [A2] MVK .S1 0, A1

[A1] B .S2 LOOP

[A1] B .S2 LOOP

LOOP:
[A1] B .S2 LOOP
|| [A1] SUB .L1 A1, 1, A1
B .S2 B3
NOP 5

/****************** End count.asm ***************************************************/

/******************* COMMON.H V1.00 **********************************************/
/* Copyright (c) 1997 Texas Instruments Incorporated */
/******************* COMMON.H V1.00 **********************************************/

#include <dma.h>
#include <emif.h>
#include <intr.h>
#include <cache.h>
#include <timer.h>
#include <hpi.h>
#include <mcbsp.h>
#include <regs.h>
#include <stdio.h>
#include <trgcio.h>
#include <stdlib.h>

/* variables used in tcase */
int mcsplto0;
int mcsplto0;
volatile int xmit1_done;
volatile int recv0_done;
volatile int xmit0_done;
volatile int recv1_done;

/* The foll. CLKSTP defines should actually be added to mcbsp.h in the devlib src */
#define CLKSTP_OFF 0x00 /* CLKSTP mode disabled */
#define CLKSTP_NO_DELAY 0x02 /* Clock starts without delay */
#define CLKSTP_DELAY 0x03 /* Clock starts with delay */

#define FALSE 0
#define TRUE 1

/* BUFFERS DEFINED IN data6201.asm */
#define BUFFER_SIZE 256
extern int in0[BUFFER_SIZE];
extern int out0[BUFFER_SIZE];

extern cregister volatile unsigned int AMR;
extern cregister volatile unsigned int CSR;
extern cregister volatile unsigned int IFR;
extern cregister volatile unsigned int ISR;
extern cregister volatile unsigned int ICR;
extern cregister volatile unsigned int IER;

extern interrupt void c_nmi01(void);
extern interrupt void c_int04(void);
extern interrupt void c_int05(void);
extern interrupt void c_int06(void);
extern interrupt void c_int07(void);
extern interrupt void c_int08(void);
extern interrupt void c_int09(void);
extern interrupt void c_int10(void);
extern interrupt void c_int11(void);
extern interrupt void c_int12(void);
extern interrupt void c_int13(void);
extern interrupt void c_int14(void);
extern interrupt void c_int15(void);

extern void set_interrupts(void);
#define DMA_XFER 0
#define POLL_XFER 1
#define INT_XFER 2
#define GPIO 3
#define DLB1 4
#define DLB2 5
#define SPLIT_XFER 6
#define HW_BYTE 7
#define DMA_SPI 8
/**************** End common.h ************************************/