Description

The TMS320C55x Instruction Set Simulator, available within the Code Composer Studio for TMS320C5000 IDE, simulates the instruction set of many members of the C55x family of devices. Table 1 lists the processors supported, with the corresponding configuration to be selected under the Import Configuration menu of Code Composer Studio Setup. The default configuration at installation is the C55x Functional Simulator. Subsequently, as Code Composer Studio IDE starts up it will load the last configuration selected in Code Composer Studio Setup.

Table 1. Processors Supported by the C55x Simulator

<table>
<thead>
<tr>
<th>PROCESSOR</th>
<th>CODE COMPOSER STUDIO IDE IMPORT CONFIGURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS320C55x</td>
<td>C55xx Functional Simulator</td>
</tr>
<tr>
<td>TMS320C55x</td>
<td>C55x Cycle Accurate Simulator</td>
</tr>
<tr>
<td>TMS320C55x</td>
<td>C55x Cache Simulator</td>
</tr>
<tr>
<td>TMS320C5502</td>
<td>C5502 Functional Simulator</td>
</tr>
<tr>
<td>TMS320C5502</td>
<td>C5502 Device Simulator</td>
</tr>
<tr>
<td>TMS320C5510</td>
<td>C5510 Device Simulator</td>
</tr>
<tr>
<td>TMS320C5509†</td>
<td>C5510 Device Simulator†</td>
</tr>
</tbody>
</table>

†C5510 Device Simulator configuration is the nearest possible configuration for the C5509 device. Users can use the C5510 Device Simulator with the memory configuration of the C5509 device, which can be configured using the SIM5510.cfg file. Peripheral simulation is not supported for USB, I2C, MMC and Memory Stick.

Considerations for Choosing a Simulator

The different simulator configurations provide a tradeoff between the functionality modeled, cycle accuracy of the simulation and simulation performance. Based on the needs of the application, in terms of the target functionality modeled and the levels of cycle accuracy required, different configurations can be used.

The C55x Functional Simulator can be used when the user is interested in functional verification of the core algorithm. Here, the user is not concerned about the cycle numbers and full device features.

The Cycle Accurate CPU Simulator configuration can be used if the user’s primary interest is in optimizing the core algorithms. Here the user is concerned with cycle accuracy of the core and does not need full device simulation. Any accesses outside the core will be handled by a flat memory and, therefore, the cycles measured will not account for any memory access latencies.
The Functional Device Simulator configurations model the functionality of some of the key peripherals beyond the core simulator, with minimal impact on simulator performance. The peripherals modeled in these simulators are modeled functionally to support programmer view: they are not cycle-accurate. These configurations can be used when the user is interested in the features of the device supported in the functional simulator but not present in the core simulator, particularly when cycle accuracy of the simulation is not important. For example, the C5502 Functional Simulator can be used for application development where accuracy is not needed. The C5502 Functional Simulator runs faster than the cycle-accurate C5502 Device Simulator.

The Device Simulator configurations model most of the peripherals of the device. The peripherals modeled are cycle-accurate, as in the silicon. These simulators can be used to get an indication of the cycle behavior of the application.
Supported Hardware Resources

The following section provides a concise overview of the supported hardware resources for each of the simulator configurations. For more detailed information, see the Detailed Capabilities and Limitations section, beginning on page 10.

CPU

CPU (Revision 2.1) forms the core of all C55x based simulators and can be simulated at the following two levels of accuracy:

- Functional Accuracy - the simulator simulates all the instructions functionally, neglecting pipeline effects.
- Cycle Accuracy - the simulator is pipeline accurate. Cycle effects are modeled.

All the C55x devices (C5510 and C5502) use the Cycle Accurate CPU simulator. In both cases the full instruction set architecture execution is supported. For more information, see the TMS320C55x DSP CPU Reference Guide (literature number SPRU371) and the TMS320C55x DSP CPU Mnemonic Instruction Reference Guide (literature number SPRU375) and the TMS320C55x DSP CPU Algebraic Instruction Reference Guide (literature number SPRU374).

Memory

C55x CPU Functional Simulator

The C55x CPU Functional Simulator uses the flat memory system (memory with no latency and no DARAM/SARAM).

C55x Cycle Accurate CPU Simulator

The C55x Cycle Accurate CPU Simulator supports program/data memory with latency for DARAM models. If the memory configuration is not provided, a flat memory system is used as default.

C5510 Device Simulator

The C5510 Device Simulator has internal memory interface support interfacing with SARAM and DARAM models. It also provides external memory support interfacing with Asynchronous and SBSRAM models.

C5502 Functional Simulator

The C5502 Functional Simulator supports internal memory with read/write operations.

C5502 Device Simulator

The C5502 Device Simulator provides internal memory interface support with SARAM and DARAM models. External memory support interfacing with Asynchronous, SDRAM, and SBSRAM models is also supported.
Peripherals

Table 2 shows the peripherals supported under each simulator configuration. For more information on these peripherals, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

Table 2. Peripherals Supported by the C55x Simulator†

<table>
<thead>
<tr>
<th>SIMULATOR CONFIG</th>
<th>TIMERS</th>
<th>EMIF</th>
<th>CACHE</th>
<th>PERIPHERAL BUS CONTROLLER</th>
<th>SERIAL PORTS</th>
<th>DMA CONTROLLER</th>
<th>HPI</th>
<th>API</th>
<th>DPLL</th>
<th>GPIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>C55x Functional Simulator</td>
<td>✔ (Timer0 &amp; Timer1)</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C55x Cycle Accurate Simulator</td>
<td>✔ (Timer0 &amp; Timer1)</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>C5510 Device Simulator</td>
<td>✔ (Timer0 &amp; Timer1)</td>
<td>✔ (Supports ASYNC &amp; SBSRAM No support for SDRAM)</td>
<td>✔ (I-Cache)</td>
<td>✔ (Rhea)</td>
<td>✔ (McBSP0, McBSP1 &amp; McBSP2)</td>
<td>✔ (6 channel DMA)</td>
<td>✔ (Ehpi (16) Simulation Performed using Files)</td>
<td>✔</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C5502 Functional Simulator</td>
<td>✔ (Four(4) 64-bit timers†)</td>
<td>–</td>
<td>✔ (I-Cache)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✔ (6 channel DMA) (functional only)</td>
<td>NA</td>
<td>NA</td>
<td>–</td>
</tr>
<tr>
<td>C5502 Device Simulator</td>
<td>✔ (Four(4) 64-bit timers†)</td>
<td>✔ (Supports ASYNC, SBSRAM &amp; SDRAM)</td>
<td>✔ (I-Cache)</td>
<td>✔ (VBUS)</td>
<td>✔ (McBSP0, McBSP1 &amp; McBSP2)</td>
<td>✔</td>
<td>✔ (6 channel DMA)</td>
<td>–</td>
<td>NA</td>
<td>–</td>
</tr>
</tbody>
</table>

† ✔=Supported; – = Not Supported, NA= Not Applicable
† Two general purpose timers, one watchdog timer, and one BIOS timer.
Supported Simulation Features

External Event and Data Simulation

The simulators simulate the hardware inside a particular DSP device, whereas in real hardware DSP interacts with many other external entities. The interactions between the simulator and these external entities fall into the following two categories:

- Control Signals – These signals trigger activities to the simulator (e.g. interrupts, serial port clocks, serial port synchronization events, etc.)
- Data Values – These are part of an interaction between the simulator and an external entity (e.g. read and write to peripheral registers as a part of I/O memory, serial port data, etc.)

For example, in an audio device the serial port of the DSP is connected to A/D and D/A converters or to a codec. The interaction between the DSP and the audio device happens through transfer of a synchronization signal to start a sample, as well as the sample data itself. Here the synchronization signal falls into the Control Signals category and the sample data falls in Data Values category.

The simulator provides two features – namely Pin Connect and Port Connect – for the simulation of these two types of interactions, respectively.

Pin Connect

The Pin Connect tool allows the user to simulate events from the external world.

Generally, control signals from external entities to the simulator are of most interest to the user. Pin Connect provides a generic way to simulate the control signals from the external entities to the simulator. In these cases, only the control signals and the time at which the signal must be triggered are important. The simulator provides the user with a list of pins corresponding to different control signals. The user must specify all the clock values at which events are to be triggered on this pin using a special format in a file. The user must then connect this file to the pin using command window, GEL commands or Pin Connect plug-in.

The C55x simulator provides the Pin Connect feature for all processor configurations. The pins supported for different processor configurations are shown in Table 3.

<table>
<thead>
<tr>
<th>SIMULATOR CONFIGURATION</th>
<th>PINS SUPPORTED</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTERRUPTS</strong></td>
<td><strong>SERIAL PORT RELATED PINS</strong></td>
</tr>
<tr>
<td>(PULSE TYPE)</td>
<td>(PULSE W/ CLOCK RATIO SPECIFIED)†</td>
</tr>
<tr>
<td><strong>HPI PIN</strong></td>
<td>(HPI TYPE)</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>C55x Functional Simulator</td>
<td>NMI</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>C55x Cycle Accurate Simulator</td>
<td>NMI</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>C5510 Device Simulator</td>
<td>NMI</td>
</tr>
<tr>
<td></td>
<td>RST</td>
</tr>
<tr>
<td>C5502 Functional Simulator</td>
<td>NMI, RST</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† In the case of serial ports, the CLKX pin functionality is combined with the functionality of the corresponding FSX pin, and only one logical pin – FSX – is provided. The same is true of the CLKR pin. In the pin-connect file of these FSX and FSR pins, information regarding CLKX (or CLKR), such as the clock ratio with respect to CLOCKOUT, is provided in addition to the information regarding external frame synchronization events.
‡ Use of SINT4 and SINT22 is not recommended, as they are used by the Timer0 and Timer1 peripherals, respectively.
§ n = 0, 1, and 2
Port Connect

The Port Connect feature allows the user to simulate data transfer between the DSP and an external entity which is present in the real hardware.

The transfer of data between the DSP and an external entity (which sits at some particular address in the memory space of the DSP) can happen in the following two ways:

- Data from an external entity to the DSP
- Data from the DSP to an external entity

To simulate data transfer from an external entity to the DSP, first, all data values which will be transferred from the external entity to the DSP are put into a file (the details of the file format are discussed later). Then an association is made in the simulator between this file and the address at which the external entity sits. This association is called read-mode port-connect. Whenever the simulator must read from the external entity through the address associated, it reads from the file one word at a time. To simulate the data transfer from the DSP to an external entity, a file is port-connected in write mode against the address of the external entity. Whenever the simulator has to write data to that address, it writes in the file one word at a time.

In the audio example described in the Pin Connect section, although only one bit at a time of a sample transfers through the serial line, samples can be encapsulated in words as serial port reads from the Serial Port Receive Register (DRR) or writes in to the Serial Port Transmit Register (DXR). All the samples that are to be sent to the simulator are written in a file and this file is port-connected in read mode against the memory address corresponding to the DRR. Whenever the serial port inside the simulator has to read one word from the DRR as part of the receive operation, it reads the word from the file. Similarly, to get the data serial port transmits through the serial line, a file is port-connected in write mode against the memory address corresponding to the DXR. Whenever the serial port inside the simulator writes one word to the DXR as a part of the transmit operation, it also writes the word into the file.

The Port Connect tool allows the user to access a file through a memory address, and then, by connecting to the memory (port) address, read data in from a file and/or write data out of a file. The simulator provides Port Connect for all processor configurations. Port Connect is supported on both data memory and I/O memory, however, it is not recommended to use the Port Connect feature for any I/O memory space in which a peripheral is modeled. Also remember that in the case of serial ports, data can be transmitted by connecting some files at the memory mapped locations for the serial port transmit register in write mode. Similarly, data can be received by connecting some files at the memory mapped locations for serial port receive register in read mode.

For the C5502 Device Simulator, Port Connect is available for the following additional modules:

- UART: Port Connect support for THR (write) and RBR (read)
- I2C: Port Connect support for ICDXR (write) and ICDRR (read)
- McBSP: Port Connect support for DRR and DXR
- EMIF: Port Connect to Data Memory

Port Connect File Format

The Port Connect file contains one or more lines. Each line contains less than 80 characters to represent one data value. The data is specified in hex format without any ‘0x’ prefix or ‘h’ suffix.

The following is a sample Port Connect file:

```
6666
9999
aaaa
cccc
7f7f
```
Notes:
● The first value is taken as (6666)_{16} not (6666)_{10}.
● If a single file is used in read mode for a range of addresses, one line (hence, one datum) is read from the file and the file pointer is advanced to the next line for any address within the range. For example, if the example file is used for a range 0x2000-0x2004 and the read access is made to the addresses 0x2000, 0x2001, 0x2000, 0x2000, 0x2003 (in that order) during a simulation session, the values will go to the addresses in the order – 0x6666 to 0x2000, 0x9999 to 0x2001, 0xaa9a to 0x2000, 0xc0cc to 0x2000 and 0x7f7f to 0x2003. Similarly, in write mode, when there is a write to any address in the range, one line containing that data is written in the file.
● If a Port Connect file is used in read mode with the no-rewind attribute, for any access made to the address after end-of-file is reached, the value 0xFFFF is read and the file pointer is kept unchanged. Otherwise the file pointer is rewound to the beginning and then one datum is read. For example, if the example file is used for address 0x2000 in read mode, at the time of the sixth read access to that address, the value 0xFFFF is read if no-rewind attribute is set. Otherwise 0x6666 is read.

For information on how to use Port Connect features from the Command window, GEL files, or the Port Connect plug-in, please see the Code Composer Studio IDE online help.

Pipeline Stall Analyzer

C55x is a pipeline-protected device. Although the hardware guarantees correct execution (by introducing stall cycles), the program may result in more than necessary execution cycles, making it less efficient in terms of performance. Also, the maximum optimization level possible is currently limited by compiler capabilities. After the maximum optimization done by compiler DSP application, programmers are challenged by further optimization.

The C55x Simulators (except for the C55x Functional Simulator) also introduce stall cycles to avoid data hazards, similarly to the hardware. Since the simulator is a software representation of the device, it is possible to use its flexibility to help the users develop more efficient code. The C55x Pipeline Stall Analyzer plug-in helps the C55x programmer to write more efficient DSP code by avoiding unnecessary stall cycles. The Pipeline Stall Analyzer not only provides detailed analysis of various Pipeline Stalls, it also provides a visual instruction buffer queue (IBQ) display. The IBQ display is useful in cases where PC discontinuity with code that contains long instructions is causing many pre-fetch stalls. This visual aid helps the DSP programmer to reduce these pre-fetch stalls.

Please refer to the Code Composer Studio IDE online help on C55x Pipeline Stall Analyzer to obtain more information on using the C55x Pipeline Stall Analyzer plug-in.

Simulator Analysis

The C55x simulator analysis module gives a detailed look into events occurring in the hardware, expanding debugging capabilities beyond software breakpoints.

Simulated events can include different types of stall events, such as, Decode Stall Event, Address Stall Event, Cache Miss, Cache Hit, etc. The Simulator Analysis plug-in reports the occurrence of particular system events so the user can accurately monitor and measure the performance of the program. The events can be set up to either increment a counter or to halt the execution when they are triggered.

The ability to halt execution on an event can be used to debug the execution of the application. For example, by setting up an Address Stall Event, one can debug when exactly a PPU Address Stall occurred and can use the Pipeline Stall Analyzer plug-in to identify the instructions involved in the stall and the resource causing the conflict.

The ability to count events over a period of execution will give the user a perspective of how the program is behaving during its execution period. For example, counting the number of cache misses will help identify the hot spots for memory layout optimization.

Please refer to the Code Composer Studio IDE online help to obtain the list of analysis events available for each of the configurations and how to enable them through the Simulator Analysis Plugin.
RTDX

Real-Time Data Exchange (RTDX) is supported when running inside the simulator. To run RTDX inside the simulator, the user must link applications with the RTDX Simulator Target library. It is easy to switch applications from running inside the simulator to running on real hardware. Please refer to the Code Composer Studio IDE online documentation on RTDX for more details.

All applications using DSP/BIOS can be run on the C55x Functional, C55x Cycle Accurate, and C5510 Device Simulators. In order to enable RTA for these applications, the user must ensure that the RTDX mode in the configuration is set to simulator. Please refer to the Code Composer Studio IDE online documentation on DSP/BIOS for more details on RTA and how to configure for a Simulator target.

There is no RTDX/BIOS support for either the C5502 Functional Simulator or the C5502 Device Simulator.

DSP/BIOS

All applications using DSP/BIOS can be run on all the C55x simulators. In order to enable RTA for these applications, the user needs to ensure that the RTDX mode in the configuration is set to simulator. Please refer to the Code Composer Studio online documentation on DSP/BIOS for more details on RTA and how to configure the simulator target.

Bootload

This feature is available only C5502 Device Simulator.

The C5502 Bootload feature is used to transfer code from an external source into internal or external program memory following power-up. This allows the code to reside in slow, non-volatile memory externally and be transferred to high-speed memory to be executed.

The Bootload happens only if it is enabled by specifying a valid Bootmode through a simulator configuration file. For more details on the simulator configuration file, see the Configuring the Simulator section, starting on page 28.

Changing the Stack Configuration

The C55x supports three modes of stack operation:
● 2x16-bit memory + register stack (fast return through RETA)
● 2x16-bit memory stack (slow return via memory)
● 32-bit memory stack (slow return via memory).

To change between different stack modes, the configuration register must be modified. This register can be modified by the user. The configuration register resides at the first 4 bits of the reset vector. At reset, C55x ignores the first 8 bits of the reset vector, pushes the lower 24 bits to the program counter, and starts executing from there. The user can change the first 4 bits of the reset vector to change the stack mode. By default, the stack mode is 32-bit stack operation (slow return).

To change from the default to a different stack mode:
1) Load the program with the default stack configuration.
2) Modify the first 4 bits of the reset vector residing at 0xFFFF00 (program space) to the value desired. For example, to change from 32-bit stack mode to 2x16-bit memory and register stack mode, change the first 4 bits of reset vector from 0110 (default) to 0000 (fast return through RETA).
3) Perform a reset through the debugger.
4) Step into the program and start executing. The new stack configuration will be in effect from that point on. If the user does a restart immediately after reset instead of doing a step, the new stack configuration will not be effective. The restart will again force the PC to start address without decoding the delay slot instruction. The user must do a step to change the stack configuration.
Alternatively, the user can use memory store instruction to modify the first 4 bits of the reset vector and execute a software reset instruction. This will have the same effect as the steps described above.

For more information about different stack modes, please refer to C55x Code Composer Studio IDE online help.

C54x Compatible Mode Operation
The reset value of the 54CM bit is 1, so by default C55x simulator behaves in C54x™ compatible mode. In this mode, the user needs to be aware of the following functionality differences.

Indirect addressing
- Indirect addressing uses the ARx register in place of the DRx register for *(ARx +/- DRx) expressions
- Circular addressing always uses BK03 for block-size calculation

Repeat loop
- In C54x-compatible mode, the simulator supports only one level of hardware repeat loops. In case of nested repeats, BRC0/RSA0/REA0 registers will be used even for inner loops, so the user must save and restore these registers before the start of nested loops.
- The user can terminate/activate blockrepeat (and localrepeat) by setting the BRAF bit (ST1_55 register, bit 15) to 0/1 by bit instruction. The BRAF bit is only visible in C54x compatible mode.

ASM compatibility
The lower 5 bits of DR2 register are mirrored in the lower 5 bits of the ST1_55 register.

far() qualifier
The far() qualifier with call ACx and goto ACx instruction enables use of only the 16-bit user stack (similar to C54x). This qualifier is activated only in Lead mode and will be available for the translated code.
Detailed Capabilities of Individual Configurations

C55x Functional Simulator

The simulator simulates all the instructions functionally. The instructions are executed one at a time, neglecting the pipeline effects of the CPU.

Capabilities

- Full instruction set architecture execution (except emulation)
- Support of both software and hardware interrupts. Hardware interrupts are supported using Pin Connect.
- Functional Timer support
  - Setting up Timer registers
  - Countdown and generation of interrupts
  - Timer 0 generates SINT4 and Timer 1 generates SINT22.

For more information on the timers, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

- Generic HWA interface support
- Support of following HWA modules:
  - HWA0 - Motion estimator
  - HWA1 - DCT estimator
  - HWA2 - Pixel Interpolation
- RTDX support
  - Host-target and target-host communication
  - Both small and large memory models are supported
- Support for WrapAround Detection

Unsupported Features

- Instruction buffer unit
- Pipeline protection unit
- Instruction fetch/execution pipeline
- Memory bypass mechanism
- Slow program/data memory
- Pipeline Stall Analyzer plugin support
- I-Cache is not modeled

Note: No C55x devices are simulated using the functional CPU simulator.
C55x Cycle Accurate Simulator

The cycle accurate C55x Simulator models correctly the PPU and introduces stall cycles to avoid data hazards.

Capabilities

- Full instruction set architecture execution (except emulation instructions and IDLE instruction)
- Configurable memory system simulation
  - If the memory configuration is not provided, a flat memory system (with DARAM) is used as default
  - Program/data memory with latency for both SARAM and DARAM is supported
  - If the memory map is provided in a configuration file, the driver uses the cycle accurate memory system (SARAM/DARAM). Support of SARAM and DARAM memory models follow the C55x memory protocol and access priorities. To use the memory system, the user must set up the configuration file accordingly. See the section of this document on Configuring Simulation for more information.
  - If a hole exists in the memory map, access to an unmapped location generates a bus error and is flagged in the eighth bit of the IFR1 register (INT24).
- The estop_1 instruction can be used in the code as a software breakpoint in addition to the simulator breakpoints.
- The simulator driver includes I/O memory (a placeholder for peripherals) that supports word reads/writes. This functionality can be used for general access storage.
  I/O memory is supported for accesses from 0x0 to 0xFFFF, however, two functional timer modules are simulated at I/O memory ranges 0x1000 to 0x13FF and 0x2400 to 0x27FF. It is not recommended to do I/O memory accesses in these ranges.
- Functional Timer support
  - Setting up Timer registers
  - Countdown and generation of interrupts
  - Timer 0 generates SINT4 and Timer 1 generates SINT22.

For more information on the timers, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

- RTDX support
  - Host-target and target-host communication.
  - Both small and large memory models are supported.
- Support for WrapAround Detection
- Support for Memory Bypass Detection
- Support for Generic HWA Interface
- Support for the following HWA modules:
  - HWA0 - Motion estimator
  - HWA1 - DCT estimator
  - HWA2 - Pixel Interpolation

Limitations

- Memory map creation and deletion not supported via the Code Composer Studio IDE menu - configuration of the memory system must be done through modifying the simulator configuration file.
C5502 Functional Simulator

This simulator is referred to as a functional simulator because it models the functionality of the peripherals without modeling their full cycle behavior. This approach allows these simulators to be faster than the corresponding cycle-accurate model, while functionally allowing the user to run applications that make use of the peripherals. Since the peripheral models are not full-cycle models, cycle accuracy of the simulation is not to be expected.

The C5502 Functional Device Simulator has a cycle-accurate CPU model, whereas I-Cache, DMA, Timer, and memory are functional in nature and do not guarantee cycle accuracy.

All accesses to external memory take a delay of five cycles over internal memory accesses.

CPU and Memory

- C55x CPU full instruction set execution (except emulation instructions).
- Internal memory (flat memory in the DARAM space) with read/write operations is supported.

I-Cache

Following bank types are supported:
- Direct
- 2-way set associative

I-Cache can be enabled or disabled during Simulation.

- Set the 14th bit of ST3 to 1 to enable I-Cache. Reset it to zero to disable I-Cache. By default, none of the I-Cache banks will be enabled.
- Program the I-CACHE registers to configure the I-Cache and enable respective banks of I-Cache.
- I-Cache is functional in nature, so it does not simulate actual number of cycles, however, it does how correct hits and misses.

DMA Controller

- Programming of DMA registers.
- Transfers on all channels supported.
- Basic transfers from/to internal-external memory systems supported (Software + external event initiated)
- Status register updates and CPU interrupts for half-frame, frame, last-frame and block-transfer-complete are supported.
- Post-increment and constant addressing mode is supported.

Timer

- Four 64-bit timers are supported – 2 General Purpose (GP), 1 WatchDog (WD) and 1 BIOS.
- 32-bit chained mode supported: 32-bit counter with 32-bit linear pre-scaler.
- 32-bit unchained mode supported: two parallel 32-bit timers, one with 4-bit pre-scaler.
- The user program needs to program the timer for selecting modes.
- Timer input/output pins not supported.
- Clock and Pulse modes are supported for TSTAT bit functionality.
- TSTAT bit toggles for every time-out in Clock mode
- TSTAT bit goes high 1 cycle after time-out in Pulse mode; it remains high for the number of cycles indicated by the PWID bits.
- TSTAT does not show any functionality in WD mode.

Interrupt mapping for timer pins is as follows:

<table>
<thead>
<tr>
<th>Timer</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM0</td>
<td>SINT4</td>
</tr>
<tr>
<td>TIM1</td>
<td>SINT22</td>
</tr>
<tr>
<td>TIM2</td>
<td>SINT11</td>
</tr>
<tr>
<td>TIM3</td>
<td>not connected</td>
</tr>
</tbody>
</table>

Modules Not Supported

Following modules are not supported in the C5502 Functional Simulator:

- UART
- McBSPs
- I2C
- EMIF

Other Capabilities

The peripheral registers can be viewed in the I/O Memory space. The valid ranges for the peripheral registers are shown in the Table 4:

**Table 4. Valid Ranges for C5502 Functional Simulator Peripheral Registers**

<table>
<thead>
<tr>
<th>PERIPHERAL</th>
<th>START ADDRESS</th>
<th>END ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral Bus Controller</td>
<td>0x0000</td>
<td>0x0011</td>
</tr>
<tr>
<td>DMA Configuration registers</td>
<td>0x0C00</td>
<td>0x0CAB</td>
</tr>
<tr>
<td>DMA Global Registers</td>
<td>0x0E00</td>
<td>0x0E03</td>
</tr>
<tr>
<td>TIM0</td>
<td>0x1000</td>
<td>0x1030</td>
</tr>
<tr>
<td>TIM1</td>
<td>0x2400</td>
<td>0x2430</td>
</tr>
<tr>
<td>TIM2</td>
<td>0x4000</td>
<td>0x4030</td>
</tr>
<tr>
<td>TIM3</td>
<td>0x7800</td>
<td>0x7830</td>
</tr>
<tr>
<td>I-Cache registers</td>
<td>0x1400</td>
<td>0x1403</td>
</tr>
</tbody>
</table>

RTDX Support

- Basic HOST to TARGET and vice-versa communication.
- Support of both small and large memory model.
Code Composer Studio IDE Simulation Features

- Pin Connect supported for external interrupt simulation.
  - RESET, NMI, INT0, INT1, INT2, and INT3 pins supported
- Port Connect supported for DATA and I/O memory ranges for external and peripheral accesses simulation, respectively.
- Simulator Analysis feature (for break and count) supported for the following events:
  - CPU: PPU Stall, Pre-fetch Stall, Instruction Count, PC Discontinuity
  - MEMORY: Memory Read, Memory Write
  - CACHE: Cache Hit, Cache Miss
- Pipeline Stall Analyzer Plugin support.

Limitations

- Peripheral port accesses are not supported.
- For Bootload, set the PC (Program Counter) value to the address from which the boot code needs to be loaded.
- Due to non-availability of a C5502-specific Peripheral register plugin, the user needs to view register contents for supported peripherals through the I/O memory window.
C5502 Device Simulator

The C5502 Device Simulator has the cycle-accurate CPU and cycle-accurate Timer module.

The internal memory subsystem and external memory interface and subsystem are modeled accurately. The DMA module is also modeled accurately. The rest of the peripherals modules are not cycle-accurate.

CPU and Memory

- C55x CPU full instruction set execution (except emulation instructions).
- Cycle Accurate memory is supported (as opposed to Flat memory in case of C5502 Functional Simulator).

I-Cache

Following bank types are supported:
- Direct
- 2-way set associative

Usage

- Set the 14th bit of ST3 to 1 to enable I-Cache. Reset it to zero to disable I-Cache. By default, none of the I-Cache banks will be enabled.
- Program the I-CACHE registers to configure I-Cache and enable respective banks of I-Cache.

DMA Controller

- Programming of DMA channel registers.
- Transfers on all channels supported.
- INTMEM0, INTMEM1, PERIPH and EMIF ports supported.
- Status register updates and CPU Interrupts for different channel events supported.
- All addressing modes are supported.
- Sync events from MCBSP, UART, I2C and External Interrupts are supported.

Timer

- Four 64-bit timers are supported – two General Purpose (GP), one WatchDog (WD) and one BIOS.
- 32-bit chained mode supported: 32-bit counter with 32-bit linear pre-scaler.
- 32-bit unchained mode supported: two parallel 32-bit timers, one with 4-bit pre-scaler.
- The user program needs to program the timer for selecting modes.
- Timer input/output pins not supported.
- Clock and Pulse modes are supported for TSTAT bit functionality.
- TSTAT bit toggles for every time-out in Clock mode.
- TSTAT bit goes high one cycle after time-out in Pulse mode; it remains high for the number of cycles indicated by the PWID bits.
- TSTAT does not show any functionality in WD mode.
Interrupt mapping for timer pins is as follows:

| TIM0 : SINT4 | TIM1 : SINT22 | TIM2(WD) : SINT11 | TIM3(BIOS) : not connected |

**UART**

For more information on the UART, see the *TMS320C55x DSP Peripherals Reference Guide* (literature number SPRU317).

- Supports all UART registers.
- Registers store 8-bit data.
- Functioning started-off by writing to EPMR register and programming DLL and DLH registers appropriately.
- Internal loopback mode supported.
- Supports Received Data Available Interrupt, Receiver Line Status Interrupt, Transmitter Holding Register Empty Interrupt.
- Supports DMA Events.
- Interrupt enabling by writing to IER.
- Supports a 16-byte FIFO, default mode is non-FIFO.
- Data transfers to receiver FIFO happen after a minimum time of (value programmed in divisor latches * 16 Vbus clocks).

**Limitations**

- Input from external module not supported.
- Interrupt priority not supported – interrupts are generated in order of occurrence.
- Cycle accuracy not guaranteed.
- Stop bits of 1.5 bits not supported (only 1- and 2- supported).

**I2C**

For more information on the I2C, see the *TMS320C55x DSP Peripherals Reference Guide* (literature number SPRU317).

- Supports read/write to all I2C registers.
- Functioning started-off by writing to Start bit of ICIMR.
- Internal loopback mode supported.
- Supports Registers-ready-for-access interrupt (ARDY), Receive interrupt/status (ICRINT and ICRRDY) interrupts and Transmit interrupt/status (ICXINT and ICXRDY) interrupts.
- Supports DMA Events, ICREVT and ICXEVT.
- Interrupt enabling by writing to ICIMR.
- Supports the following modes: Master Transmitter/Slave Receiver, Master Receiver/Slave Transmitter.
- SCL clock input can be provided by Pin Connecting the SCL line.
- Port Connect to ICDXR (for Write) and ICDRR (for Read) supported.
Clock high time and low time shall be as programmed in the ICCLKH and ICCLKL registers. The value programmed shall be used as the High time/Low time frequency.

Data shall be read / transmitted only once during high period of the clock.

Limitations

- The only control information supported is the Slave Address. Control Information like NACK, ACK etc. not supported.
- The SDA output will be without the I2C control information and will contain only the data that was written to its transmitter register (ICDXR) by the CPU. It will also contain the Slave Address if it is operating in a Master Transmitter mode.
- No Arbitration support, as it is assumed that there will be only a single master.
- Supports only a single master, so no Clock Synchronization is performed.
- Interrupt priority not supported – interrupts are generated in order of occurrence.
- Cycle accuracy not guaranteed.
- No support for General Purpose Register (ICGPIO) and Prescaler Register (ICPSC). So writing and reading to this register will have no effect on the functioning of the model.

McBSP

- Three McBSP's supported: McBSP0, McBSP1, McBSP2.
- Word sizes of 8-, 12-, 16-, 20-, 24- and 32-bits supported.
- Frame length up to 128 words supported.
- Normal/32/128 channel modes supported for receive and transmit.
- Digital Loopback mode (DLB) supported.
- CPU interrupts XINT/RINT supported.
- DMA events XEVT/REVT supported.
- XRDY/RRDY bit functionality supported for ready indication.
- Choice for external/internal clocks and frame syncs supported.
- DRR, DXR registers can be connected to files for Input/Output using the Port Connect feature.
- External input to CLKR, FSR, CLKX, FSX pins available using the Pin Connect feature.

Limitations

- SPI protocol not supported.
- CLKX, CLKR as sources to SRG clock not supported.
- No support for GPIO.
- 512-channel mode and related functionality not supported.
EMIF

- Supports x8/x16/x32 Async, SDRAM and SBSRAM memory. Memories are also implemented internally.
- Supports 4/2/1 CE configurations.
- Supports SDRAM commands - ACTV/READ/WRT/DCAB/DEAC.
- Generic SBSRAM interface is supported (Simple READs and WRITEs with SyncRL and SyncWL). Supports Read setup, Read Strobe, Read HOLD, Write Setup, Write Strobe and Write HOLD for Async memories.
- EMIF prioritizes between requests (WRITE, READ, PROG READ) coming from CPU/CACHE and DMA. The priorities are as follows, and priorities are fixed:
  1) CPU WRITE
  2) CPU READ
  3) PROG READ
  4) DMA WRITE
  5) DMA READ

Limitations

- For SDRAM Refresh (REFR/SLFREFR) and MRS commands are not supported.
- For SBSRAM Deselect operation is not supported.

Other Capabilities

The peripheral registers can be viewed in the I/O Memory space. The valid ranges for the peripheral registers are shown in Table 5:

Table 5. Valid Ranges for C5502 Device Simulator Peripheral Registers

<table>
<thead>
<tr>
<th>PERIPHERAL</th>
<th>START ADDRESS</th>
<th>END ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral Bus Controller</td>
<td>0x0000</td>
<td>0x0011</td>
</tr>
<tr>
<td>DMA</td>
<td>0xC00</td>
<td>0xC0f</td>
</tr>
<tr>
<td>TIM0</td>
<td>0x1000</td>
<td>0x1030</td>
</tr>
<tr>
<td>TIM1</td>
<td>0x2400</td>
<td>0x2430</td>
</tr>
<tr>
<td>TIM2</td>
<td>0x4000</td>
<td>0x4030</td>
</tr>
<tr>
<td>TIM3</td>
<td>0x7800</td>
<td>0x7830</td>
</tr>
<tr>
<td>I-Cache registers</td>
<td>0x1400</td>
<td>0x1403</td>
</tr>
<tr>
<td>UART registers</td>
<td>0x9C00</td>
<td>0x9C0C</td>
</tr>
<tr>
<td>I2C registers</td>
<td>0x3C00</td>
<td>0x3C0E</td>
</tr>
<tr>
<td>McBSP0</td>
<td>0x2800</td>
<td>0x281f</td>
</tr>
<tr>
<td>McBSP1</td>
<td>0x2c00</td>
<td>0x2c1f</td>
</tr>
<tr>
<td>McBSP2</td>
<td>0x3000</td>
<td>0x301f</td>
</tr>
<tr>
<td>EMIF</td>
<td>0x800</td>
<td>0x841</td>
</tr>
</tbody>
</table>
RTDX Support

- Basic HOST to TARGET and vice-versa communication.
- Support of both small and large memory model.

Code Composer Studio IDE Simulation Features

- Pin Connect supported for external interrupt simulation.
  - RESET, NMI, INT0, INT1, INT2, and INT3 pins supported
- Port Connect supported for DATA and I/O memory ranges for external and peripheral accesses simulation, respectively.
- Simulator Analysis feature (for break and count) supported for the following events:
  - CPU: PPU Stall, Pre-fetch Stall, Instruction Count, PC Discontinuity
  - MEMORY: Memory Read, Memory Write
  - CACHE: Cache Hit, Cache Miss
- Pipeline Stall Analyzer Plug-in support.

Limitations

- IDLE/Wakeup functionality is not supported.
- Analysis plug-in gives Error message “Can't Set Breakpoint: Error number -2600 …” if more than one event is selected as type “Break”. Please ignore this error message.
- Analysis plug-in does not actually reset the counters when RESET ALL is done. It is advisable to quit and restart Code Composer Studio IDE once again if one needs to do the profiling. The same is observed with Program Reload, Reset, and Restart.
- Due to non-availability of a C5502-specific Peripheral register plugin, the user needs to view register contents for supported peripherals through the I/O memory window.
C5510 Device Simulator

The C5510 Device Simulator has the cycle-accurate CPU and cycle-accurate Timer module.

The internal memory subsystem and external memory interface and subsystem are modeled accurately. The DMA module is also modeled accurately. The rest of the peripherals modules are not cycle-accurate.

**CPU**

- Full instruction set architecture execution (except emulation instructions and IDLE instruction).
- Parallel instruction execution.

**Internal Memory Subsystem**

- Internal memory interface supports interfacing with SARAM and DARAM models.
- SARAM and DARAM memory models are supported according to the C55x memory protocol and access priorities. To use the memory system, the user must set up the configuration file accordingly. See the Configuring Simulator section of this document for more information.
- Support of memory stalls due to slow program memory and access conflicts in SARAM and DARAM.
- If a hole exists in the memory map, access to an unmapped location generates a bus error, and it is flagged in the eighth bit of the IFR1 register (INT24).

**Limitations**

- PDROM is not supported. By default, an SARAM bank of the same size is mapped to the ROM space.
- Control of the memory map using the MPNMC bit is not supported.

**External Memory Interface and Subsystem**

For details on the EMIF features, see the *TMS320C55x DSP Peripherals Reference Guide* (literature number SPRU317).

- Three types of memory are available that can be configured as external memory: asynchronous 32-bit, asynchronous 16-bit, and 32-bit SBSRAM.
- The type of memory attached is determined by the programming of the chip enable (CE) space register in the EMIF. The memory type should be kept as EXTERNAL in the configuration file.
- If a hole exists in the memory map, CPU access generates a bus error, and it is flagged in the eighth bit of the IFR1 register (INT24).

**Limitations**

- Asynchronous 8-bit memory is not supported.
- SDRAM interface in EMIF is not supported.
- Minimum 2-cycle strobe period is needed for the asynchronous memory interface.
I-Cache
For details on the I-Cache features, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

- To enable the I-Cache, set bit 14 of ST3 to 1. Reset it to zero to disable.
- By default, none of the I-Cache banks are enabled.
- Program the I-CACHE registers to configure the I-Cache and enable respective banks of the I-Cache.
- I-Cache flushing is supported.
- The two-way I-Cache bank is supported.

DMA Controller
For details on the DMA Controller features, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

- Programming of DMA channel registers is supported.
- Transfers on all channels supported.
- All different ports are supported.
- Status register updates and CPU interrupts for different channel events supported.
- All addressing modes are supported.
- Sync events from McBSP and External Interrupts are supported.

Peripheral Bus Controller
For details on the Peripheral Bus Controller features, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

The peripheral registers can be viewed in the I/O memory space. The start addresses for the peripherals are seen in the following table.

<table>
<thead>
<tr>
<th>PERIPHERAL</th>
<th>START ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral Bus Controller</td>
<td>0x0000</td>
</tr>
<tr>
<td>External Memory Interface</td>
<td>0x0800</td>
</tr>
<tr>
<td>DMA Configuration Register</td>
<td>0x0C00</td>
</tr>
<tr>
<td>Timer Registers</td>
<td>0x1000</td>
</tr>
<tr>
<td>McBSP</td>
<td>0x2800</td>
</tr>
</tbody>
</table>

Note that Port Connect via the I/O port or I/O memory access is not supported at the above address spaces.

Timer
For details on the Timer features, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

Limitations
- Timer input/output pins are not supported.
Multi-Channel Buffered Serial Ports

For details on the McBSP features, see the TMS320C55x DSP Peripherals Reference Guide (literature number SPRU317).

- Port Connect is supported for the simulation of McBSP receive and transmit functionality.
- For the receive functionality, the user must attach a file at address 0x4801, 0x2C01, 0x3001 (DRR1 for three McBSPs).
- For the transmit functionality, the user must attach a file at address 0x4803, 0x2C03, 0x3003 (DXR1 for three McBSPs).

Limitations

- For the receive/transmit functionality, it is assumed that the clocks are synchronized.
- Only internal clock (CPU clock) synchronization is supported.

Simulating Enhanced Host Port Interface

The C5510 simulator provides support for the simulation of enhanced host port interfaces (EHPI). This simulation is performed using files that specify the values of control signals and the corresponding address and data values.

When simulating EHPI, two files are associated with EHPI. The input file specifies the commands from the host, and the output file stores output data to the host. The output file is named host.out. The name of this file cannot be changed.

Setting up the Input Command File

To simulate EHPI, the user must first create an input command file that lists the EHPI commands with their corresponding data and/or address. The format for this file is as follows:

```
{ Command1;
  Command2;
}
```

Commands use the following format:

```
Command clock_cycle [address] [data];
```

- Each command must be on a new line and the new line must not contain anything else.
- To specify comments, use a hash (#) as the first character in the line.
- The clock_cycle parameter specifies the DSP clock cycle in which the host applies the request to EHPI.
- The address parameter represents the 16-bit address field for multiplexed mode and 20-bit address field for non-multiplexed mode.
- The data parameter represents a 16-bit data field.

The command specifies the type of operation requested by the host, and can be any of the commands shown in Table 6.
Table 6. EHPI Operation Commands

<table>
<thead>
<tr>
<th>COMMANDS FOR NON-MULTIPLEXED MODE</th>
<th>SYNTAX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WRITEMEM</strong> (data write) writes a specified data word at the specified address.</td>
<td><strong>WRITEMEM</strong> clock_cycle address data</td>
</tr>
<tr>
<td><strong>READMEM</strong> (data read) reads the data word at the specified address.</td>
<td><strong>READMEM</strong> clock_cycle address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMMANDS FOR MULTIPLEXED MODE</th>
<th>SYNTAX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WRITEHPIA</strong> (hpia write) writes the specified address to the HPIA register of EHPI.</td>
<td><strong>WRITEHPIA</strong> clock_cycle address</td>
</tr>
<tr>
<td><strong>WRITEHPID</strong> (hpid write) writes the specified data at the address specified in the HPIA register.</td>
<td><strong>WRITEHPID</strong> clock_cycle data</td>
</tr>
<tr>
<td><strong>WRITEHPIDAUTOINC</strong> (hpid write autoinc) writes the specified data at the address specified in the HPIA register. HPIA is then post incremented.</td>
<td><strong>WRITEHPIDAUTOINC</strong> clock_cycle data</td>
</tr>
<tr>
<td><strong>READHPID</strong> (hpid read) reads data at the address specified in the HPIA register.</td>
<td><strong>READHPID</strong> clock_cycle</td>
</tr>
<tr>
<td><strong>READHPIDAUTOINC</strong> (read hpid autoinc) reads data at the address specified in the HPIA register. HPIA is post incremented.</td>
<td><strong>READHPIDAUTOINC</strong> clock_cycle</td>
</tr>
<tr>
<td><strong>READHPIC</strong> (hpic read) reads value from the HPIC register</td>
<td><strong>READHPIC</strong> clock_cycle</td>
</tr>
<tr>
<td><strong>WRITEHPIC</strong> (hpic write) writes the specified data value to the HPIC register.</td>
<td><strong>WRITEHPIC</strong> clock_cycle data</td>
</tr>
</tbody>
</table>

Figure 1 and Figure 2 show sample input file examples.

**Figure 1. Example Input File for Non-Multiplex Mode**

```
{ 
  WriteMem 10 0x11020 0x1234;
  ReadMem 25 0x11020;
  WriteMem 40 0x11020 0x4321;
  ReadMem 55 0x11021;
  WriteMem 70 0x11022 0xabcd;
  ReadMem 85 0x11022;
  WriteMem 100 0x11023 0xbcda;
  ReadMem 125 0x11023;
  WriteMem 140 0x11024 0x612;
  WriteMem 160 0x11025 0x64cd;
  WriteMem 180 0x11026 0xac72;
  ReadMem 215 0x11024;
  ReadMem 230 0x11025;
  ReadMem 245 0x11026;
}
```

**Figure 2. Example Input File for Multiplex Mode**

```
{ 
  WriteHpiia 10 0x11020;
  WriteHpidAutoinc 25 0xabcd;
  WriteHpidAutoinc 40 0x1342;
  WriteHpidAutoinc 60 0x6ca3;
  WriteHpid 80 0x15b2;
  WriteHpid 100 0x11020;
  ReadHpidAutoinc 120;
  ReadHpidAutoinc 160;
  ReadHpidAutoinc 200;
}
```
Connecting the Input Command File to the Interrupt Pin

To connect the input file to the interrupt pin, either the Pin Connect tool or the Command Window tool within the Code Composer Studio IDE can be used.

To use the Pin Connect tool:

5) From the Tools menu, select Pin Connect.
6) From the list of available pins, select HPI.
7) Connect the file.

To use the Command Window tool:

1) From the Tools menu, select Command Window.
2) In the Command Window, enter the following command:

```
pinc HPI, filename
```

Limitations

- The host can access only internal SARAM.
- The command file must be connected before the execution of the program begins.
- A gap of approximately 15 cycles (DSP cycles) must exist between two consecutive host commands to ensure correct operation.

RTDX Support

- Host-target and target-host communication is supported.
- Both small and large memory models are supported.

HWA Support

- HWA0 - Motion estimator
- HWA1 - DCT estimator
- HWA2 - Pixel Interpolation

Modules Not Supported

- ROM model
- General Purpose I/Os
- Clock PLL

Other Limitations

- Port Connect is not supported for Data Memory.
- There is no Pin Connect support on Timer input pins.
- Memory map creation and deletion is not supported via the Code Composer Studio menu. However, the user can configure the memory system in the simulator configuration file by following the correct syntax.
Differences Between Simulator and Silicon

Register, Memory and Watch Window Behavior

At times the memory and watch window updates on the simulator might be delayed longer than expected, and the register window updates might occur earlier than expected for some of the registers. The following paragraphs give the reasons for these behaviors.

C55x has the following pipeline stages:

Decode, Address, Access1, Access, Read, Execute, Memory Write

At any point of simulation, the blue arrow in the disassembly window or yellow arrow in source window correspond to the instruction (or source line corresponding to the instruction) which is currently in the execute phase of the pipeline. This means the instruction is about to be executed, i.e., an instruction at the end of the Read phase of the pipeline. A step command at this point would execute the instruction at the blue arrow and go until the end of the Read phase of the next instruction in the pipeline.

Some instructions in C55x finish operation in the Address phase of the pipeline (e.g., modifications of address registers, loading of repeat counters etc.). When the PC reaches those instructions, the results are already available and the new value of the modified registers can be seen in the register display window.

The Memory Write phase is activated only for those instructions where memory write/memory-mapped register writes occur.

Consider an example where a memory write instruction (Inst1) is followed by a non-memory write instruction (Inst2), and where Inst1 has completed the Read phase and is about to execute. If a single step is applied, Inst1 completes the Execute phase and enters the Write phase, while Inst2 enters the Execute phase. The user will not able to see the memory updates of Inst1 because the writes to memory only happen after the Write phase completion. In silicon, this problem might not arise because the step is implemented using some special breakpoint. By the time the ISR for the special breakpoint is handled, the wait state might have elapsed and, therefore, the memory and watch window update properly.

Blue Arrow Effect for a Memory Write Instruction Followed by a Non-memory Instruction

In this case, the first instruction finishes operation in the Memory write phase while the second instruction finishes operation in the Execute phase. Since they are pipelined, both the instructions effectively finish operation at the same clock cycle. If the PC indicator is at the first instruction, one step command will finish operation of both the instructions and the PC indicator will jump to the third instruction in the disassembly window. This might be little confusing for the user. However, the user can set a breakpoint in the second instruction and stop just before the execution if necessary.

Simulation Features to Enable Detection of Hardware Limitations

Memory Bypass Detection

The CPU bypass happens when a memory read instruction follows closely after (with no more than two slots in between) a memory write instruction to the same address. This bypass mechanism is used to resolve read-after-write dependency on a memory element.

When the relationship of the CPU bypass and the CPU STALL-AC2 (see below for the explanation of CPU STALL_AC2) is one of following five types, the read data may be corrupted – that is, the data bypassed through the internal path should be the data written by the CPU a few cycles ago, but it may be some different data.

[TYPE0]

(1) Instruction Writing to memory

(2) Instruction Reading from same memory || instruction causing CPU STALL-AC2
[TYPE1]
(1) Instruction Writing to memory
(2) any other instruction
(3) Instruction Reading from same memory || instruction causing CPU STALL-AC2

[TYPE2]
(0) Instruction Writing to memory
(1) any other instruction
(2) any other instruction
(3) Instruction Reading from same memory || instruction causing CPU STALL-AC2

[TYPE3]
(1) Instruction Writing to memory
(2) Instruction Causing CPU STALL-AC2
(3) Instruction Reading from same memory

[TYPE4]
(0) Instruction Writing to memory
(1) any other instruction
(2) Instruction causing CPU STALL-AC2
(3) Instruction Reading from same memory

NOTE : "||" represents a parallel instruction pair. "CPU STALL-AC2" is a CPU internal signal to stall (freeze) the ACCESS2, ACCESS1, ADDRESS or DECODE pipeline stage, in order to avoid a pipeline conflict which may happen at the READ, EXE or WRITE phase.

When the simulator detects the occurrence of any one of the five conditions, it flags a warning message to the user, with the program counter value of memory read instruction, and halts the simulation. The user can choose whether to continue execution further.

The warning indicates a potential data corruption for the memory read instruction. The user can avoid this by getting rid of the memory bypass condition in such situation.

Note that the CPU STALL-AC2 can be caused by not only the instruction in slot#1, but also the instruction performed in slot#2.

This feature can be enabled using the simulator configuration file. For more details, see the Configuring the Simulator section, starting on page 28.

WrapAround Detection
In the C55x CPU, addresses are comprised of 23 bit values (data address). There are three types of addressing modes:

Direct Addressing:
In this type of memory addressing, a location is referenced using a memory offset from a Data Page pointer (XDP+DP)
Absolute Addressing:
In this type of memory addressing, the address is supplied as a constant parameter with the instruction.

Indirect addressing:
In this type of memory addressing, a location is referenced using an address pointer (composed of XARn+ARn, with or without modifiers)

In the address calculations that use Address pointers XARn and ARn or XCDP and CDP, the XARn or XCDP represents the upper seven bits of the address (referred to as "PAGE") and ARn or CDP represents the lower 16 bits of the address (referred to as "OFFSET"). In C55x CPU revision 2.1/2.2, any pointer increment/decrement type of addressing mode (for example,"ARn+, *AR(#k)) causes a wrap-around of the lower 16 bits ("OFFSET") if the address crosses page boundary. This is illustrated in following figure.

This situation may result in the user misinterpreting data if he assumes the memory addressing to be linear. Also, in C55x CPU revision 3.0 onwards, the data addressing is linear, so this wraparound on page boundary does not occur. This can cause discrepancies in data interpretation across different versions of the CPU.

To address the above problem, a feature called "PAGE BOUNDARY WRAP-AROUND DETECTION" has been added in the C55x simulator to detect and warn the user against page boundary crossing by any Data Pointer.

Upon detection of a page-boundary wrap-around by data pointers, the CPU will be halted at the ADDRESS phase of the instruction, and the following message will appear in the Messages window of Code Composer Studio IDE:

"WARNING: PAGE BOUNDARY CROSSING DETECTED AT PROGRAM COUNTER 0x??????"

This feature can be enabled using the Simulator configuration file. For more details, see the Configuring the Simulator section, starting on page 28.
Configuring the Simulator

The simulator uses config files to provide the user the ability to configure itself. There is a default config file in the ‘drivers’ sub-directory in the Code Composer Studio installation directory corresponding to each configuration listed in Table 1.

Creating a Memory Map

This section describes the syntax of the C55x configuration file and how this can be used to configure memory sub-systems.

Following are the types of memories supported for simulation.

- SARAM - Single Access RAM, Only one read/write can be done per cycle.
- DARAM - Dual Access RAM, Two reads/writes can be done per cycle.
- EXTERNAL - To be handled through EMIF (External Memory). (Only available in C5510 simulation)

The memory map can be specified through the configuration file and this is included as part of the MEMORY_MAP section of the configuration file. Below is the syntax for specifying the memory map:

```
MODULE      MEM_MAP;
MEMORY MEM0;
  START 0x0; // Start Address
  NAME DARAM0; // Bank Name
  LENGTH 0x2000; // Length of the bank
  TYPE DARAM; // Type of the bank
  LATENCY 0; // Number of Wait states.
END MEM0;
END MEM_MAP;
```

**Fields in Memory Map specifications**

- "Bank Type" is either [1] SARAM or [2] DARAM
- "Bank Name" is the tag-name to distinguish different memory banks of same type. For example, SARAM4, ASYNC32, etc. (This can be any other name.)
- "Start Address" gives the starting address of the memory bank in C55x memory address range. (from 0x000000 - 0xffffffff)
- "Bank Size" is size of that memory bank (This can be 0x0 - 0xffffffff, depending on the start address.)
- "Latency" can be 0 or 1. Latency 1 means that an instruction with a read request in cycle 'n' will get its data in cycle 'n+3'. Latency 0 (zero latency memory) will get its data 'n+2'.
- When two memory banks are given such that they are overlapping, the bank given first in the configuration file is first checked. For example, in following case:
  
  Bank1 0x0 - 0x200
  Bank2 0x3 - 0xffffffff

  The actual constructed memory map looks like:
  
  Bank1 0x0 - 0x200
  Bank2 0x200 - 0xffffffff
File Format for Pin Connect

The simulator allows the user to simulate and monitor external interrupt signals.

The Pin Connect tool enables the user to specify the interval at which selected external interrupts will occur.

To simulate external interrupts:

8) Create a data file that specifies interrupt intervals.
9) Start the Pin Connect tool. From the Code Composer Studio Tools menu, choose Pin Connect.
10) Connect the data file to an external interrupt pin.
11) Load the program.
12) Run the program.

Setting Up the Input File

To simulate external interrupts, the user must first create a data file that specifies interrupt intervals. Interrupt
intervals are expressed as a function of CPU clock cycles. Simulation begins at the first clock cycle. An interrupt
will occur at each specified clock cycle.

The data file must contain a CPU clock cycle parameter in the following format:

\[[\text{clock-cycle}, \text{logic value}] \ [\text{rpt} \ (n|\text{EOS})]\]

- **clock-cycle**: The CPU clock-cycle parameter specifies the intervals at which interrupts will occur. Clock cycles can be specified as absolute or relative.
- **logic value**: The logic value parameter is valid only for the pins of waveform-type (e.g., FSX0 pin in C6201 simulator). This value (0 or 1) must be used to force the pin value to low or high at the corresponding cycle. A logic value of 0 causes the pin value to go low, and a logic value of 1 causes it to go high.

For example,

\[\[12, 1\] \ [56, 0] \ [78, 1]\]

If connected to the FSX0 pin in C6201, this will cause the pin to go high at the twelfth cycle, low at the 56th cycle, and then high at the 78th cycle.

- **rpt**: Repeat the same pattern a fixed number of times.
- **n**: A positive integer value specifying the number of times to repeat.
- **EOS**: Repeat the same pattern until the end of simulation.

**Absolute Clock Cycle**

To use an absolute clock cycle, the cycle value must represent the actual CPU clock cycle where an interrupt
should be simulated. For example,

\[12 \ 34 \ 56\]

Interrupts are simulated at the twelfth, 34th, and 56th CPU clock cycles. No operation is performed on the clock
cycle value; the interrupt occurs exactly as the clock cycle value is written.

---

**Texas Instruments**

POST OFFICE BOX 1443 • HOUSTON, TEXAS 77251-1443
**Relative Clock Cycle**

The user can also select a clock cycle that is relative to the time at which the last event occurred. A plus sign (+) before a clock cycle adds that value to the total clock cycles proceeding it. For example,

```
12  +34  55
```

In this example, a total of three interrupts are simulated at the 12th, 46th (12 + 34), and 55th CPU clock cycles. The user can mix both relative and absolute values in the data file.

**Repetition of Patterns for a Specified Number of Times**

The user can format the data file to repeat a particular pattern for a fixed number of times. For example,

```
5  (+10  +20)  rpt  2
```

The values inside the parenthesis represent the portion that is repeated. Therefore, an interrupt is simulated at the fifth CPU cycle, then the 15th (5+10), 35th (15+20), 45th (35+10), 65th (45+20) CPU clock cycles.

**Repetition to the End of Simulation (EOS)**

To repeat the same pattern throughout the simulation, add the string EOS to the line. For example,

```
10  (+5  +20)  rpt  EOS
```

Interrupts are generated at the tenth CPU cycle, the 15th (10+5), the 35th (15+20), the 40th (35+5), the 60th (40+20), and so on, continuing in that pattern until the end of simulation.

**Enabling Memory Bypass Feature Detection**

To enable the Memory Bypass Detection capability in the simulator, the config file must include the following in the MODULE C55x section:

```
MODULE  C55X;
  BYPASS_DETECTION ON; /* OFF will disable the Memory Bypass Detection*/
END C55X;
```

**Enabling WrapAround Feature Detection**

To enable the feature to detect page boundary wraparound, the config file must include the following in the MODULE C55x section:

```
MODULE C55X;
  WRAPAROUND_DETECTIONON; /* OFF will disable the Wraparound Detection */
END C55X;
```

**Enabling Bootload on C5502 Device Simulator**

In the C5502 Simulator configuration file (SIM5502.cfg), add following section:

```
MODULE BOOTLOAD ;
  BOOT_MODE  <EMIF/QUICKBOOT/NO_BOOT>;
BOOT_CODE    <Boot Table Name>;
END BOOTLOAD;
```

- In case of bootload using EMIF, the simulator expects the boot table to be present at 0x200000 word address.
- Do not use BOOT_CODE if it is in EMIF mode.
- Use QUICKBOOT for HPI/IIC/UART/ McBSP etc., bootloads. Specify the boot table name to be specified as BOOT_CODE.
- If bootload is not required, use NO_BOOT or do not put the above section in configuration file.
Performance Numbers

Table 7 shows the performance numbers of the simulator for different device configurations. These numbers were gathered on a 900MHz Intel® Pentium® III PC with 128MB of RAM under normal load conditions. The application used for measurement is the Reed-Solomon encoding and decoding application from a standard benchmarking suite.

Table 7. Performance Numbers of the C55x Simulator

<table>
<thead>
<tr>
<th>SIMULATOR CONFIGURATION</th>
<th>SIMULATOR SPEED</th>
</tr>
</thead>
<tbody>
<tr>
<td>C55x Functional Simulator</td>
<td>1.35 MIPS</td>
</tr>
<tr>
<td>C55x Cycle Accurate Simulator</td>
<td>310 KCPS, 210 KIPS</td>
</tr>
<tr>
<td>C5510 Device Simulator</td>
<td>25 KCPS, 10 KIPS</td>
</tr>
<tr>
<td>C5502 Functional Simulator</td>
<td>67 KCPS, 27 KIPS</td>
</tr>
<tr>
<td>C5502 Device Simulator</td>
<td>40 KCPS, 12 KIPS</td>
</tr>
</tbody>
</table>

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Related Documentation

**TMS320C55x DSP CPU Reference Guide** (literature number SPRU371) describes the architecture, registers, and operation of the CPU for the TMS320C55x fixed-point DSPs. This book also describes how to make individual portions of the DSP inactive to save power.

**TMS320C55x DSP Peripherals Reference Guide** (literature number SPRU317) describes the peripherals, interfaces, and related hardware that are available on TMS320C55x DSPs.

**TMS320C55x DSP Mnemonic Instruction Set Reference Guide** (literature number SPRU374) describes the TMS320C55x DSP mnemonic instructions individually. Also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the algebraic instruction set.

**TMS320C55x DSP Algebraic Instruction Set Reference Guide** (literature number SPRU375) describes the TMS320C55x DSP algebraic instructions individually. Also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the mnemonic instruction set.
IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of that third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright © 2002, Texas Instruments Incorporated