TMS320C55x Instruction Set Simulator

Technical Reference

Literature Number: SPRU599D
April 2005

TEXAS INSTRUMENTS

Contents

List of Tables

Read This First

About This Manual

This manual does the following:

- Names the TMS320C55x™ digital signal processors that are supported by configurations of the C55x™ Instruction Set Simulator.
- Lists which modules and pins of each device are modeled
- Describes capabilities and limitations of the simulator
- Explains how to configure the simulator
- Provides some benchmarking data on cycle accuracy

Notational Conventions

This document uses the following conventions:

- Program examples are shown in a `special typeface`.
- In syntax descriptions, key words or symbols are shown in **bold** and variables are shown in *italics*. Portions of a syntax that are in bold should be entered as shown; portions of a syntax that are in italics describe the type of information that should be entered.
- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets. Unless the square brackets are in a **bold** typeface, do not enter the brackets themselves.
- Braces ({ and }) indicate that you must choose one of the parameters within the braces; you do not enter the braces themselves.
- The pipe symbol (|) represents a logical OR.

Related Documentation From Texas Instruments

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

TMS320C55x DSP Mnemonic Instruction Set Reference Guide (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 (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.

Trademarks

TMS320C55x, C55x, Code Composer Studio, TMS320C5000, RTDX, DSP/BIOS, C54x are trademarks of Texas Instruments.

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

TEXAS INSTRUMENTS

Chapter 1

SPRU599D−April 2005

Introduction to the TMS320C55x Simulator

The TMS320C55x™ Instruction Set Simulator, available within the Code Composer Studio™ Integrated Development Environment (IDE) for TMS320C5000™, simulates the instruction set of many members of the C55x™ family of devices.

Topic **Page**

1.1 Features

The TMS320C55x simulators support the following features:

- TMS320C55x CPU full instruction set architecture execution
  - Pipeline protection for internal registers and memory locations
  - Parallel instruction execution
  - Support for embedded breakpoints by estop_1 instructions
- Configurable memory simulation
- External memory interface and subsystem for C5510 and C5502 device simulators
- Port Connect, supports external peripheral simulation
- Pin Connect, supports external interrupt simulation
- Simulator analysis support
- Address trace support
- Rewind support for C55x CPU simulators
- RTDX ™ support
  - Host-target and target-host communication
- DSP/BIOS™ Real-Time Analysis (RTA) support
- Pipeline stall analyzer support
- Support for generic hardware accelerator interface

1.2 Processors Supported by the C55x Simulator

Table 1-1 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 Rev 2.x CPU Functional Simulator. Subsequently, as Code Composer Studio IDE starts up it will load the most recent configuration selected in Code Composer Studio Setup.

Table 1-1. Processors Supported by the C55x Simulator

PROCESSOR	CODE COMPOSER STUDIO IDE IMPORT CONFIGURATION
TMS320C55x	C55x Rev 2.x CPU Functional Simulator
TMS320C55x	C55x Rev 3.0 CPU Functional Simulator
TMS320C55x	C55x Rev 2.x CPU Cycle Accurate Simulator
TMS320C55x	C55x Rev 3.0 CPU Cycle Accurate Simulator
TMS320C5502	C5502 Device Simulator
TMS320C5510	C5510 Device Simulator
TMS320C5509†	C5510 Device Simulator(1)

(1) 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.

1.3 Considerations for Choosing a Simulator

The different simulator configurations provide a trade-off 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 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. Cycle-accurate SARAM/DARAM memory modules are supported with this configuration.

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.

1.4 Supported Hardware Resources

The following sections provide a concise overview of the supported hardware resources for each of the simulator configurations. For more detailed information, see Chapter 3.

1.4.1 CPU

CPU Revision 2.2 forms the core of C55x Rev 2.x simulators and C55x-based device simulators. CPU Revision 3.0 forms the core of C55x Rev 3.0 simulators. C55x simulators are available 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, refer to the *TMS320C55x DSP CPU Reference Guide* (SPRU371), the *TMS320C55x DSP CPU Mnemonic Instruction Reference Guide* (SPRU375), and the *TMS320C55x DSP CPU Algebraic Instruction Reference Guide* (SPRU374).

1.4.2 Memory

The memory supported varies by the simulator configuration.

1.4.2.1 C55x Rev 2.x CPU Functional Simulator

The C55x Rev 2.x CPU Functional Simulator uses the flat memory system (memory with no latency and no DARAM/SARAM).

1.4.2.2 C55x Rev 3.0 CPU Functional Simulator

The C55x Rev 3.0 CPU Functional Simulator uses the flat memory system (memory with no latency and no DARAM/SARAM).

1.4.2.3 C55x Rev 2.x Cycle Accurate CPU Simulator

The C55x Rev 2.x Cycle Accurate CPU Simulator supports program/data memory with latency for SARAM/DARAM models. If the memory configuration is not provided, a flat memory system is used as default.

1.4.2.4 C55x Rev 3.0 Cycle Accurate CPU Simulator

The C55x Rev 3.0 Cycle Accurate CPU Simulator supports program/data memory with latency for SARAM/DARAM models. If the memory configuration is not provided, a flat memory system is used as default.

1.4.2.5 C5510 Device Simulator

The C5510 Device Simulator (Revision PG 2.2) has internal memory interface support interfacing with SARAM and DARAM models. It also provides external memory support interfacing with Asynchronous and SBSRAM models.

1.4.2.6 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.

1.4.3 Peripherals

Table 1-2 shows the peripherals supported under each simulator configuration. For more information on these peripherals, see the *TMS320C55x DSP Peripherals Reference Guide* (SPRU317).

Table 1-2. Peripherals Supported by the C55x Simulator

					SERIAL PORTS								
SIMULATOR CONFIGURATION	**TIMERS**	**EMIF**	**CACHE**	**PERIPHERAL BUS CONTROLLER**	**MCBSP**	**I2C**	**UART**	**DMA CONTROLLER**	**HPI**	**API**	**DPLL**	**GPIO**	**HWA**
C55x Rev 2.x CPU Functional Simulator	ü [1] (Timer0 & Timer1)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	ü
C55x Rev 3.0 CPU Functional Simulator	(Timer0 & Timer1)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	ü
C55x Rev 2.x CPU Cycle Accurate Simulator	ü (Timer0 & Timer1)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	ü
C55x Rev 3.0 CPU Cycle Accurate Simulator	ü (Timer0 & Timer1)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	ü
C5510 Device Simulator	ü (Timer0 & Timer1)	ü (Supports ASYNC, SBSRAM & SDRAM)	ü (I-Cache)	ü (Rhea)	ü (McBSP0, McBSP1 & McBSP2)	NA	NA	ü (6 channel DMA)	–	ü	–	–	ü
C5502 Device Simulator	ü (Four(4) 64-bit timers [2])	ü (Supports ASYNC, SBSRAM & SDRAM)	ü (I-Cache)	ü (VBUS)	ü (McBSP0, McBSP1 & McBSP2)	ü	ü	ü (6 channel DMA)	–	NA	–	–	ü

[1] ü =Supported; - = Not Supported, NA= Not Applicable
[2] Two general purpose timers, one watchdog timer, and one BIOS timer.

Chapter 2

SPRU599D−April 2005

Supported Simulation Features

This chapter provides a concise overview of the supported simulation features for each of the simulator configurations. For more detailed information, see Chapter 3.

Topic **Page**

2.1 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** trigger activities to the simulator (such as interrupts, serial port clocks, and serial port synchronization events).
- **Data Values** are part of an interaction between the simulator and an external entity (such as read and write to peripheral registers as a part of I/O memory and serial port data).

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 - *Pin Connect* and *Port Connect* - for the simulation of these two types of interactions, respectively.

2.1.1 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 the command window, GEL commands or the 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 2-1.

Table 2-1. Pins Supported by the Pin Connect Feature of theC55x Simulator

SIMULATOR CONFIGUR-ATION	PINS SUPPORTED									
	INTERRUPTS (PULSE TYPE)			SERIAL PORT RELATED PINS (PULSE WITH CLOCK RATIO SPECIFIED)[1]						HPI PIN (HPI TYPE)
C55x CPU Func-tional Simu-lator[2]	NMI	SINT2, SINT3 –SINT24[3]	–	NA						NA
C55x CPU Cycle Accurate Simu-lator[2]	NMI	SINT2, SINT3 –SINT24[3]	–	NA						NA
C5510 Device Simulator	NMI	–	INT0 –INT5	FSRn[4]	CLKRn[4]	FSXn[4]	CLKXn[4]	CLKSn[4]	–	HPI
C5502 Device Simulator	NMI	–	INT0 –INT3	FSRn[4]	CLKRn[4]	FSXn[4]	CLKXn[4]	CLKSn[4]	SCL (part of I2C)	NA

(1) 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.
(2) Applicable to Rev 2.x as well as Rev 3.0 CPU cores.
(3) Use of SINT4 and SINT22 is not recommended, as they are used by the Timer0 and Timer1 peripherals, respectively.
(4) n = 0, 1, and 2

2.1.2 Port Connect

The Port Connect feature allows the user to simulate data transfer between the DSP and an external entity that 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 a 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 file format is discussed in Section 2.1.2.2). Then an association is made in the simulator between this file and the address for the external entity. This association is called read-mode port-connect. Whenever the simulator must read from the external entity through the associated address, 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.

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 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 of 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 of 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.

2.1.2.1 Memory Ranges for Port Connect

Available memory ranges on the TMS320C55x devices, to which a file can be connected for reading/writing, are given in Table 2-2.

Table 2-2. Available Memory Ranges for Port Connect for TMS320C55x Devices

SIMULATOR CONFIGURATION	AVAILABLE PORT CONNECT RANGE
C55x Rev 2.x CPU Functional Simulator	Data memory: 0x0000 - 0x7FFFFF (page 1)(1) I/O memory: 0x0000 - 0xFFFF (page 2)
C55x Rev 3.0 CPU Functional Simulator	Data memory: 0x0000 - 0x7FFFFF (page 1)(1) I/O memory: 0x0000 - 0xFFFF (page 2)
C55x Rev 2.x CPU Cycle Accurate Simulator	Data memory: 0x0000 - 0x7FFFFF (page 1)(1) I/O memory: 0x0000 - 0xFFFF (page 2)
C55x Rev 3.0 CPU Cycle Accurate Simulator	Data memory: 0x0000 - 0x7FFFFF (page 1)(1) I/O memory: 0x0000 - 0xFFFF (page 2)
C5510 Device Simulator	Data memory: 0x0000 - 0x7FFFFF (page 1)(1) I/O memory: 0x0010 - 0x7FF, 0x0816 - 0x0BFF, 0x0D00 - 0x0DFF, 0x0E02 - 0x0FFF, 0x1004 - 0x13FF, 0x1408 - 0x23FF, 0x2404 - 0x27FF, 0x2802 - 0x2BFF, 0x2C20 - 0x2FFF, 0x3020 - 0xFFFF (page 2)
C5502 Device Simulator	Data memory: 0x0000 - 0x7FFFFF (page 1)(1) I/O memory: 0x0012 - 0x07FF, 0x0840 - 0xBFF, 0xD00 - 0xDFF, 0xE02 - 0xFFF, 0x1014 - 0x13FF, 0x1405 - 0x23FF, 0x2414 - 0x27FF, 0x2820 - 0x2BFF, 0x2C20 - 0x2FFF, 0x3020 - 0x3BFF, 0x3C10 - 0x3FFF. 0x4020 - 0x9BFF, 0x9C10 - 0x9FFF, 0xA010 - 0xFFFF (page 2)

(1) Using the Port Connect feature over the memory mapped register region (0x00 - 0x5F) is not recommended.

2.1.2.2 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 in the Port Connect file is interpreted as hex data and specified with or without a preceding ‘0x’.

The following is a sample Port Connect file:

```
6666
9999
aaaa
; This is a commented line
cccc ; This is a commented sentence
7f7f
```

Comments are allowed in the Port Connect read file. Comments should begin with a semi-colon (;), as shown in the sample file.

Read modes. There are two modes to connect a Port Connect file:

- **Rewind mode** is the default mode of connection. In this mode, after completely consuming the read file contents, the simulator rewinds the file and starts reading from the beginning of the file for any further read accesses. For example, if the sample file is used for address 0x2000 in read mode, at the time of the sixth read access to that address, 0x6666 is read.
- In **no-rewind mode** for read accesses made after end-of-file, the value 0xFFFF is read and the file pointer is kept unchanged. For example, if the sample 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.

Reset. On reset, all the Read file pointers are rewound, and Write files are closed and reopened in write mode.

Port Connect can occur through the command window, GEL commands, or the Port Connect plug-in. Please see the Code Composer Studio online help for more information on how to connect and disconnect a specified port.

2.2 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 execution cycles than necessary, making it less efficient in terms of performance. Also, the maximum optimization level possible is currently limited by compiler capabilities. After the maximum optimization is done by the compiler DSP application, programmers are challenged by further optimization.

The C55x simulators (except for the C55x Rev 2.x CPU Functional Simulator and the C55x Rev 3.0 CPU 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 online help for more information on using the C55x Pipeline Stall Analyzer plug-in.

2.3 Simulation Pipeline Modes

C55x simulators (except the C55x Functional CPU Simulator) support two modes of execution called *Pipeline Flush* mode and *Pipelined* mode.

- In **Pipeline Flush mode**, when the simulation is halted, the execution pipeline is flushed. The watch window shows the correct values of local variables in this mode. Profile data may be skewed because of flushing the pipeline whenever the simulation halts. This skew is noticeable only if there are many breakpoints in the program.

  The PC (program counter) corresponds to the address of the instruction that is about to enter the *Decode* phase of the pipeline
- In **Pipelined mode**, when the simulation is halted, the execution pipeline is not flushed. Pipeline behavior is accurate in this mode. This mode provides accurate profile data. The watch window display may not show the correct values of local variables.

  The PC corresponds to the address of the instruction that is about to enter the *Execute* phase of the pipeline.

2.3.1 Switching Between the Modes

Pipeline Flush mode is the default simulation mode. The mode can be changed using the following mechanisms:

- From the Code Composer Studio Debug Menu, by enabling/disabling Flush Pipeline on Halt.
- Using the GEL command GEL_SimSetMode(Arg):
  - Arg value 0 sets the mode to Pipeline Flush mode
  - Arg value 1 sets the mode to Pipelined mode.

2.3.2 Effects of Mode Switching

From Pipeline Flush mode to Pipelined mode:

- Simulation advances the pipeline to bring the PC to the execution phase.
- After mode switching, the PC points to the instruction in the *Execute* phase of the pipeline.

From Pipelined mode to Pipeline Flush mode:

- Simulation flushes the pipeline.
- After mode switching, the PC points to the instruction in the *Decode* phase of the pipeline. However, this effect is hidden from the user unless the user does an Instruction Step after the mode change. If an Instruction Step is done after mode change, the user might see skid of a few instructions in the assembly window and PC changing in the register window.
- Breakpoints that have been set at instructions which are being flushed out of the pipeline are ignored.

2.3.3 Recommendations on Simulation Mode

The simulation mode that is recommended varies according to which features are used:

- Pipelined Mode
  - Pipeline Stall Analyzer
  - ACT Tools
  - Simulator Analysis
- Pipeline Flush Mode
  - Watch window
  - Register window
  - Memory window

2.3.4 Related Warnings

The following warning message will be displayed from the watch window when the simulator is in Pipelined Mode:

```
Watch Window: (Warning) Pipeline flush mode is OFF. Values may be
inaccurate due to pipeline stage delays. To ensure accurate reporting, turn on
"Flush Pipeline on Halt" from Debug menu.
```

The following warning message will be displayed from Pipeline Stall Analyzer when the simulator is in Pipeline Flush Mode:

```
Simulator is currently in Pipeline Flush Mode. Pipeline Flush Mode
does not support Pipeline Stall Analyzer Tool. To use Pipeline Stall Analyzer,
turn off "Flush Pipeline on Halt" from Debug menu.
```

2.4 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 and Address Stall Event, and other events, such as Cache Miss, and Cache Hit. 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 exactly when 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 online help to obtain the list of analysis events available for each of the configurations and how to enable them through the Simulator Analysis Plug-in.

2.5 Address Trace Support

C55x simulators (except for the C55x Rev 2.x CPU Functional Simulator and the C55x Rev 3.0 CPU Functional Simulator) generate address traces of all accesses from CPU to Memory. The accesses include instruction fetch, data read, and data write. A trace record is comprised of the address, the address type (i.e., instruction fetch, data read, data write), and the cycle (time stamp) at which the access occurred. The trace is compressed using PDATS (Packed and Differential Address and Time Stamp), a popular format for address access compaction.

To generate and read traces:

- Enable trace generation from Code Composer Studio Setup. See Section 5.4.
- Generate the trace by running the application on the simulator.
- Use the trace reader to read this trace. (Refer to the trace reader documentation available with this installation.)

Note:
In the PDATS trace, the cycle (time stamp) at which a memory access occurred can change with latency and properties of memory. Therefore, the trace generated by one simulator configuration may not match the trace generated by another simulator configuration for the same application.

2.6 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 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 real-time analysis (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 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 the C5502 Device Simulator.

2.7 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.

2.8 Bootload

This feature is available only with the 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 Code Composer Studio Setup, see Chapter 5 for more information.

2.9 Changing the Stack Configuration

The C55x supports three modes of stack operation:

- 2×16-bit memory + register stack (fast return through RETA)
- 2×16-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/Run into the program and start executing. The new stack configuration will be in effect from that point on. **If the user does a restart or any PC write immediately after reset instead of doing a step/run, 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/run to change the stack configuration.**

Alternatively, the user can use a 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, refer to the C55x Code Composer Studio online help.

2.10 C54x Compatible Mode Operation

The reset value of the 54CM bit is 1, so by default the 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 the DR2 register are mirrored in the lower 5 bits of the ST1_55 register.

far() qualifier

The far() qualifier with the 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.

2.11 Stack Size Reporting

Stack size reporting provides DSP programmers with useful information on stack size usage during an application run. Simulators track stack pointers during an application run and can provide information about minimum and maximum stack pointer values at any time during an application run. This helps in estimating the stack usage of the application and understanding whether the stack is overflowing or is under utilized.

The C5500 simulators track the maximum and minimum values of the SP and SSP registers. The simulators also provide information about the PC at which minimum or maximum value of SP and SSP the PC was observed. These are exposed through pseudo registers.

Refer to the Code Composer Studio IDE online help to obtain more information on using the C55x Stack Size Reporting.

2.12 Rewind

The C55x CPU simulators under the Code Composer Studio environment support a feature called Rewind. Using Rewind, the past history of an application being executed can be viewed. This reduces the time required to debug an application. Refer to the *Rewind User's Guide* (SPRU713) for more details.

See Section 5.5 for details on enabling this feature on C55x CPU simulators.

Detailed Capabilities of Individual Configurations

This chapter describes the capabilities and known limitations of each simulator configuration. For some configurations the unsupported features are listed.

Topic **Page**

3.1 C55x Rev 2.x CPU Functional Simulator/C55x Rev 3.0 CPU Functional Simulator

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

3.1.1 Capabilities

The simulators support these features:

- Full instruction set architecture execution (except emulation)
- Both software and hardware interrupts. Hardware interrupts are supported using Pin Connect.
- Functional Timer
  - 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* (SPRU317).
- Generic HWA interface
- The following HWA modules:
  - HWA0 - Motion estimator
  - HWA1 - DCT estimator
  - HWA2 - Pixel Interpolation
- RTDX
  - Host-target and target-host communication
  - Both small and large memory models
- WrapAround Detection
- Self-modifying code
- Rewind
- Detection of program accesses to memory mapped registers region

3.1.2 Unsupported Features

The simulators do not support these features:

- Instruction buffer unit
- Pipeline protection unit
- Instruction fetch/execution pipeline
- Memory bypass mechanism
- Slow program/data memory
- Pipeline Stall Analyzer plug-in support
- I-Cache

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

3.2 C55x Rev 2.x CPU Cycle Accurate Simulator/C55x Rev 3.0 CPU Cycle Accurate Simulator

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

3.2.1 Capabilities

The simulators support these features:

- Full instruction set architecture execution (except emulation instructions and IDLE instruction)
- Configurable memory system simulation
  - If the memory configuration is not provided, a full DARAM memory system is used as the 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 follows the C55x memory protocol and access priorities. To use the memory system, the user must set up the configuration file accordingly; see Chapter 5 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 process I/O memory accesses in these ranges.
- Functional Timer
  - Setting up Timer registers
  - Countdown and generation of interrupts
  - Timer 0 generates SINT4 and Timer 1 generates SINT22.
- RTDX
  - Host-target and target-host communication
  - Both small and large memory models are supported.
- WrapAround Detection
- Memory Bypass Detection
- Self-modifying code
- Rewind
- Generic HWA Interface
- The following HWA modules:
  - HWA0 - Motion estimator
  - HWA1 - DCT estimator
  - HWA2 - Pixel interpolation
- Detection of program accesses to memory mapped registers region

3.2.2 Limitations

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

3.3 C5502 Device Simulator

The C5502 Device Simulator has the cycle-accurate CPU.

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

3.3.1 CPU

The simulators support these features:

- C55x CPU full instruction set execution (except emulation instructions)
- Cycle Accurate memory.

3.3.2 Internal Memory Subsystem

The internal memory subsystem consists of these features:

- The 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 Section 5.6 for more information.
- 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.

3.3.3 Peripherals

The peripheral modules supported are I-Cache, DMA Controller, Timer, UART, I2C, McBSP, and EMIF.

3.3.3.1 I-Cache

The following bank types are supported:

- Direct
- Two-way set associative

3.3.3.2 DMA Controller

The DMA controller functionality consists of the following:

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

3.3.3.3 Timer

The timer functionality consists of the following:

- Four 64-bit timers: two General Purpose (GP), one WatchDog (WD) and one BIOS.
- 32-bit chained mode: 32-bit counter with 32-bit linear pre-scaler.
- 32-bit unchained mode: 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 are **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

3.3.3.4 UART

The UART functionality consists of the following:

- All UART registers
- Registers store 8-bit data.
- Functioning is started-off by writing to the EPMR register and programming the DLL and DLH registers appropriately.
- Internal loopback mode
- Received Data Available Interrupt, Receiver Line Status Interrupt, and Transmitter Holding Register Empty Interrupt
- DMA Events
- Interrupt enabling by writing to IER.
- 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

These are the limitations of the UART support:

- Input from an external module is not supported.
- Interrupt priority is not supported. Interrupts are generated in order of occurrence.
- Cycle accuracy is not guaranteed.
- Stop bits of 1.5 bits are not supported. Only stop bits of 1 or 2 bits are supported.

3.3.3.5 I2C

The I2C functionality consists of the following:

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

Limitations

These are the limitations of the I2C support:

- The only control information supported is the slave address; not information like NACK and ACK.
- The SDA output does not include the I2C control information and contains only the data that was written to its transmitter register (ICDXR) by the CPU. SDA output also contains the slave address if it is operating in a Master Transmitter mode.
- No arbitration is supported 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 is not supported. Interrupts are generated in order of occurrence.
- Cycle accuracy is not guaranteed.
- There is no support for the General Purpose Register (ICGPIO) and the Prescaler Register (ICPSC). So writing and reading to these registers will have no effect on the functioning of the model.

3.3.3.6 McBSP

The multi-channel buffered serial port (McBSP) functionality consists of the following:

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

Limitations

These features are not supported:

- SPI protocol
- CLKX and CLKR as sources to SRG clock
- GPIO
- 512-channel mode and related functionality
- Multi-channel mode of operation

3.3.3.7 EMIF

The EMIF functionality consists of the following:

- Memory types: x8/x16/x32 Async, SDRAM and SBSRAM memory. Memory types are also implemented internally.
- 4/2/1 CE configurations
- SDRAM commands: ACTV/READ/WRT/DCAB/DEAC.
- Generic SBSRAM interface (Simple READs and WRITEs with SyncRL and SyncWL) includes: 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 fixed as follows:
  1. CPU WRITE
  2. CPU READ
  3. PROG READ
  4. DMA WRITE
  5. DMA READ

Limitations

These features are not supported:

- The SDRAM Refresh (REFR/SLFREFR) and MRS commands
- The SBSRAM Deselect operation

3.3.4 RTDX Support

The RTDX functionality consists of the following features:

- Host-target and target-host communication
- Both small and large memory models

3.3.5 Self-Modifying Code

The self-modifying code functionality consists of the following:

- Applications where the CPU modifies program memory contents are supported.
- Applications where peripherals, such are DMA, modify program memory contents are not supported.

3.4 C5510 Device Simulator

The C5510 Device Simulator has the cycle-accurate CPU.

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

3.4.1 CPU

Provides full instruction set architecture execution (except emulation instructions and IDLE instruction).

3.4.2 Internal Memory Subsystem

The internal memory subsystem functionality consists of the following:

- The internal memory subsystem interfaces with SARAM and DARAM models
- SARAM and DARAM memory models according to the C55x memory protocol and access priorities. To use the memory system, the user must set up the configuration file accordingly. See Section 5.6 for more information.
- 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. The bus error is flagged in bit 8 of the IFR1 register (INT24).

Limitations

These features are not supported:

- PDROM. By default, an SARAM bank of the same size is mapped to the ROM space.
- Control of the memory map using the MPNMC bit

3.4.3 Peripherals

The peripheral modules supported are I-Cache, DMA Controller, Peripheral Bus Controller, Timer, McBSP, and EMIF.

3.4.3.1 I-Cache

The I-Cache functionality consists of the following:

- To enable the I-Cache, set bit 14 of ST3 to 1. Reset it to 0 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
- The two-way I-Cache bank

3.4.3.2 DMA Controller

The DMA controller functionality consists of the following:

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

3.4.3.3 Peripheral Bus Controller

The peripheral registers can be viewed in the I/O memory space. The start addresses for the peripherals are seen in the following table. Port Connect via the I/O port or I/O memory access is not supported at these address spaces.

PERIPHERAL	START ADDRESS
Peripheral Bus Controller	0x0000
External Memory Interface	0x0800
DMA Configuration Register	0x0C00
Timer Registers	0x1000 0x2400
McBSP	0x2800 0x2C00 0x3000

3.4.3.4 Timer

The timer input/output pins are not supported.

3.4.3.5 McBSP

The multi-channel buffered serial port (McBSP) functionality consists of the following:

- 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, or 0x3001 (DRR1 for three McBSPs).
- For the transmit functionality, the user must attach a file at address 0x4803, 0x2C03, or 0x3003 (DXR1 for three McBSPs).

Limitations

These are the limitations of the McBSP support:

- For the receive/transmit functionality, it is assumed that the clocks are synchronized.
- Only internal clock (CPU clock) synchronization is supported.
- The multi-channel mode of operation is not supported.

3.4.3.6 EMIF and Subsystem

The EMIF functionality consists of the following:

- 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. The bus error is flagged in bit 8 of the IFR1 register (INT24).

Limitations

These are the limitations of the EMIF support:

- Asynchronous 8-bit memory is not supported.
- A minimum 2-cycle strobe period is needed for the asynchronous memory interface.
- Self-modifying code is not supported.

3.4.4 RTDX Support

The RTDX functionality consists of the following:

- Host-target and target-host communication
- Both small and large memory models

3.4.5 HWA Support

The HWA functionality consists of the following:

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

3.4.6 Self-Modifying Code

The self-modifying code functionality consists of the following:

- Applications where the CPU modifies program memory contents are supported.
- Applications where peripherals, such as DMA, modify program memory contents are not supported.

3.4.7 Unsupported Modules

These modules are not supported:

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

3.4.8 Other Limitations

The simulator has these additional limitations:

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

Chapter 4

SPRU599D − April 2005

Differences Between Simulator and Silicon

This chapter describes the differences between the simulator features and the silicon.

Topic **Page**

4.1 Features to Detect Hardware Limitations

The memory bypass and wrap-around detection features enable the simulator to detect hardware limitations.

4.1.1 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 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:

- 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.
- || represents a parallel instruction pair.

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

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 situations.

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

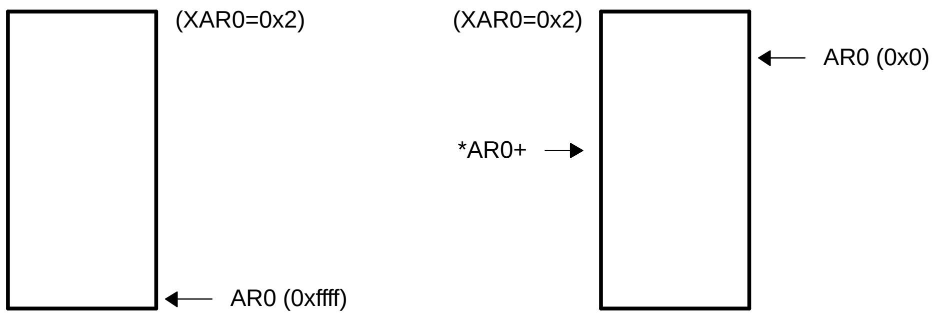
See Section 5.8 for help on enabling this feature.

4.1.2 *Wrap-Around Detection*

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

- **Direct Addressing** where a location is referenced using a memory offset from a Data Page pointer (XDP+DP).
- **Absolute Addressing** where the address is supplied as a constant parameter with the instruction.
- **Indirect Addressing** where 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 a 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, the *page-boundary wrap-around detection* feature 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 with *addr* replaced by the address:

WARNING: PAGE BOUNDARY CROSSING DETECTED AT PROGRAM COUNTER 0x*addr*

See Section 5.1 for help on configuring this feature.

4.2 Reload/Restart Behavior in the Simulator

Application execution on C55x simulators may not be proper if reload/restart is performed without having done a reset.

Configuring the Simulator

Simulators can be configured for different features though the Code Composer Studio Setup program. However, to modify the advanced options, you need to modify the base configuration file (see Section 5.10).

Topic **Page**

5.1 Setting the Wrap-Around Detection Feature

This feature can be configured through Code Composer Studio Setup by setting the appropriate option for the Detect page wrap around entry in the processor properties window.

5.2 Setting the Bootload on the C5502 Device Simulator

Bootload can be configured through Code Composer Studio Setup by selecting one of the following options for the Boot Mode entry under the processor properties window:

- NONE (default)
- EMIF
- QUICK (Quickboot)

The simulator expects the boot table to be present at 0x200000 word address in case BOOT_MODE is set to EMIF.

Use Quickboot for HPI/IIC/UART/McBSP bootloads. Use the Boot Code option in Code Composer Studio Setup to specify the *boot table name*. You cannot specify the Boot Code option if the simulator is in EMIF Mode.

Use NONE if bootload is not required.

5.3 Setting the Overlay Mode

This feature can be configured through Code Composer Studio Setup by setting the appropriate option for the Simulate Overlay entry in the processor properties window.

5.4 Setting the Address Trace Generation Mode

This feature can be configured through Code Composer Studio Setup by setting the appropriate option for the Generate Address Trace entry in the processor properties window.

The address trace (compressed using the PDATS format) is generated in the same folder as the program file, with the extension .pdat. For example, if the program file is c:\test\mp3decoded.out, the address trace will be generated in the c:\test\mp3decoded.pdat file.

Alternatively, if you want the trace file generated at a custom location, you can modify the base configuration file (see Section 5.10) with the following syntax in the MODULE C55X section:

MODULE C55X;
    PDATS_TRACE *filename* **;**
END C55X;

For example, the following configuration file entry will generate c:\mp3decoder.pdat as the address trace file:

```
MODULE C55X;
      PDATS_TRACE c:\mp3decoder.pdat;
END C55X;
```

The generated trace can be interpreted using the trace reader provided with this Code Composer Studio release. The trace reader is available in the <*CCStudio Installation Directory*>\bin\utilities\trace folder, and the documentation is available in the <*CCStudio Installation Directory*>\bin\utilities\trace\docs folder.

5.5 Setting Up the Rewind Feature

Using Rewind you can view the past history of an application being executed on the simulator. This reduces the time required to debug an application. Presently, only CPU simulators support this option. This feature is disabled (OFF) by default, but it can be enabled (ON) from Code Composer Studio Setup for the CPU simulators.

This feature can be configured through Code Composer Studio Setup by setting the appropriate option for the Rewind entry in the processor properties window.

Once Rewind is enabled, Code Composer Studio Setup will also let you select between two options for where the Rewind trace will be located:

- In Memory (default)
- On Disk

See the *Rewind User's Guide* (SPRU713), for more information on configuring the simulator for Rewind.

5.6 Creating a Memory Map

This section describes the syntax for specifying the memory map in the configuration files, and how this can be used to configure memory sub-systems.

These 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)

5.6.1 Configuration File

The memory map can be specified through the MEMORY_MAP section of the configuration file. Here is an example to illustrate the syntax for specifying the memory (address and length are in bytes):

```
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;
```

Overlapping memory banks are not supported.

Note:
MEMORY_MAP is not configurable through Code Composer Studio Setup, you must configure the base configuration file (see Section 5.10).

5.6.2 Fields in Memory Map Specifications

The fields used to specify the memory map are:

- *Bank Type* is either [1] SARAM or [2] DARAM
- *Bank Name* is the tag-name to distinguish different memory banks of the same type. For example, SARAM4 and ASYNC32. (This can be any other name.)
- *Start Address* gives the starting address of the memory bank in C55x memory address range from 0x000000 - 0xffffff.
- *Bank Size* is size of that memory bank. Bank Size can be 0x0 - 0xffffff, 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 defined such that they are overlapping, the bank defined first in the configuration file is checked first. For example, in the following case:

  Bank1 0x0 - 0x200

  Bank2 0x3 - 0xffffff

  The actual constructed memory map looks like:

  Bank1 0x0 - 0x200

  Bank2 0x200 - 0xffffff

5.7 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:

1. Create a data file that specifies interrupt intervals.
2. Start the Pin Connect tool. From the Code Composer Studio Tools menu, choose Pin Connect.
3. Connect the data file to an external interrupt pin.
4. Load and Run the program.

5.7.1 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:

[*clock-cycle***,** *logic value***]**[rpt {*n*|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. 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.

5.7.2 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.

5.7.3 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 preceding 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.

5.7.4 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 parentheses 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), and 65th (45+20) CPU clock cycles.

5.7.5 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.

5.8 Setting the Memory Bypass Detection Modes

To enable the memory bypass detection capability in the simulator, the base configuration file (see Section 5.10) must include the following in the MODULE C55x section:

MODULE C55X;
BYPASS_DETECTION {**ON** | **OFF**}
END C55X;

BYPASS_DETECTION ON enables memory bypass detection.

BYPASS_DETECTION OFF disables memory bypass detection.

Note:
This feature must be configured by modifying the base configuration file (see Section 5.10.)

5.9 Applicable Configurable Features for Individual Configurations

Table 5-1 provides the supported configurable features for each of the C55x simulator configurations.

- The items in the NOT APPLICABLE ENTRIES column should not be put in the base configuration file (see Section 5.10) for the corresponding simulator driver.
- MODULE PROFILE was used to enable multi-event profiler, code coverage, and cache analysis data collection when simulators are used with the Analysis Toolkit (ATK). From CCStudio 2.4 onwards, this section should not be used as the data collection will be enabled and controlled through Profile Setup in the CCStudio IDE.
- See the *Rewind User's Guide* (SPRU713) for advanced options with Rewind.
- These features are configured in the base configuration files for the devices, if applicable (see Section 5.10.) These values should be changed only if you want to change the default values for the features, or if you want to use a feature that is not configurable from Code Composer Studio Setup.

Table 5-1. Applicable Configuration File Entries for C55x Simulator Configurations

SIMULATOR CONFIGURATION	DRIVER NAME	DEFAULT CONFIGURATION FILE	DEFAULT ENTRIES	APPLICABLE OPTIONAL ENTRIES	NOT APPLICABLE ENTRIES
C55x Rev 2.x CPU Functional Simulator	Tisimfuncc55x.dvr	SIM55xx.cfg		• MODULE C55x parameters: WRAPAROUND_DETECTION OVERLAY • MODULE REWIND	• MODULE C55x parameters: CHIP PDATS_TRACE BYPASS_DETECTION • MODULE CHIPNAME • MODULE CACHE • MODULE PROFILE • MODULE BOOTLOAD • MODULE MEM_MAP
C55x Rev 2.x CPU Cycle Accurate Simulator	Tisimc55x.dvr	SIM55xx.cfg		• MODULE C55x parameters: WRAPAROUND_DETECTION OVERLAY PDATS_TRACE BYPASS_DETECTION • MODULE REWIND • MODULE MEM_MAP	• MODULE C55x parameters: CHIP • MODULE CHIPNAME • MODULE CACHE • MODULE PROFILE • MODULE BOOTLOAD
C55x Rev 3.0 Functional Simulator	Tisimfuncc55x.dvr	SIM55x_rev30.cfg	• MODULE C55x parameters: CPU_REV rev 3.0	• MODULE C55x parameters: WRAPAROUND_DETECTION OVERLAY • MODULE REWIND	• MODULE C55x parameters: CHIP PDATS_TRACE BYPASS_DETECTION • MODULE CHIPNAME • MODULE CACHE • MODULE PROFILE • MODULE BOOTLOAD • MODULE MEM_MAP
C55x Rev 3.0 CPU Cycle Accurate Simulator	tisimc55x.dvr	SIM55x_rev30.cfg	• MODULE C55x parameters: CPU_REV rev 3.0	• MODULE C55x parameters: WRAPAROUND_DETECTION OVERLAY PDATS_TRACE BYPASS_DETECTION • MODULE REWIND • MODULE MEM_MAP	• MODULE C55x parameters: CHIP • MODULE CHIPNAME • MODULE CACHE • MODULE PROFILE • MODULE BOOTLOAD
C5510 Device Simulator	Tisimc5510.dvr	SIM5510.cfg	• MODULE C55x parameters: CHIP C5510 • MODULE CHIPNAME • MODULE C5510 • MODULE MEM_MAP	• MODULE C55x parameters: WRAPAROUND_DETECTION OVERLAY PDATS_TRACE BYPASS_DETECTION • MODULE CACHE	• MODULE C55x parameters: CPU_REV • MODULE PROFILE • MODULE BOOTLOAD • MODULE REWIND

Table 5-1. Applicable Configuration File Entries for C55x Simulator Configurations (continued)

SIMULATOR CONFIGURATION	DRIVER NAME	DEFAULT CONFIGURATION FILE	DEFAULT ENTRIES	APPLICABLE OPTIONAL ENTRIES	NOT APPLICABLE ENTRIES
C5502 Device Simulator	Tisimc5502.dvr	SIM5502.cfg	• MODULE C55x parameters: CHIP C5502 • MODULE C5502 • MODULE MEM_MAP	• MODULE C55x parameters: WRAPAROUND_DETECTION OVERLAY PDATS_TRACE BYPASS_DETECTION • MODULE CACHE • MODULE BOOTLOAD	• MODULE C55x parameters: CPU_REV • MODULE CHIPNAME • MODULE PROFILE • MODULE REWIND

5.10 Base Configuration File

There is a configuration file (.cfg) corresponding to each simulation driver present in the drivers sub-folder in the Code Composer Studio installation directory. These files hold the default values for the features supported by the corresponding simulation driver. These values should be changed only if you want to change the default values for the features, or if you want to use a feature that is not configurable from Code Composer Studio Setup, such as BYPASS_DETECTION.

The name of the base configuration file appears in the processor properties dialog in Code Composer Studio Setup as a read-only value.

Statistics

This chapter provides the performance numbers and cycle accuracy figures for the simulator.

Topic **Page**

6.1 Performance Numbers

Table 6-1 shows the performance numbers of the simulator for different device configurations. These numbers were gathered on a 2.4GHz Intel™Pentium™ IV PC with 512MB 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 6-1. Performance Numbers of the C55x Simulator

SIMULATOR CONFIGURATION	SIMULATOR SPEED
C55x Rev 2.x CPU Functional Simulator	2.48 MIPS
C55x Rev 3.0 CPU Functional Simulator	2.43 MIPS
C55x Rev 2.x CPU Cycle Accurate Simulator	1.24 MCPS, 554 KIPS
C55x Rev 3.0 CPU Cycle Accurate Simulator	1.04 MCPS, 531 KIPS
C5510 Device Simulator	690 KCPS, 323 KIPS
C5502 Device Simulator	546 KCPS, 184 KIPS

6.2 Cycle Accuracy

TMS320C5500 simulators have been validated for cycle accuracy using a benchmark suite of applications. The measurements on device simulators have been carried out in the following categories:

- CPU + INTMEM (SARAM/DARAM)
- CPU + INTMEM+ EMIF
- CPU + INTMEM + DMA + EMIF
- Full applications using all of the above models

The details on the benchmarking data are provided in Table 6-2.

Table 6-2. Benchmarking Data for C5500 Simulators

APPLICATION/KERNEL USED FOR BENCHMARKING	REFERENCE HARDWARE	SIMULATOR CONFIGURATION	% VARIANCE IN CYCLE NUMBERS	REMARKS
On-Chip Memory Accesses (CPU + SARAM + DARAM)				
Picture Filtering - 3x3 Correlation with Rounding	C5510 DSK	C5510 Device Simulator	0	
Complex Fast Fourier Transform - CFFT	C5510 DSK	C5510 Device Simulator	-5.33	
Discrete Cosine Transform - 8x8 DCT	C5510 DSK	C5510 Device Simulator	-9.74	
On-Chip and Off-Chip Memory Accesses (CPU + SARAM + DARAM + EMIF)				
Picture Filtering - 3x3 Correlation with Rounding	C5510 DSK	C5510 Device Simulator	-8.77	
Complex Fast Fourier Transform - CFFT	C5510 DSK	C5510 Device Simulator	-2.32	
Discrete Cosine Transform - 8x8 DCT	C5510 DSK	C5510 Device Simulator	-3.42	
Inverse Discrete Cosine Transform - 8x8 IDCT	C5510 DSK	C5510 Device Simulator	-3.61	
Full Applications With DMA (CPU + SARAM + DARAM + DMA + EMIF)				
Picture Filtering - 3x3 Correlation with Rounding	C5510 DSK	C5510 Device Simulator	1.20	
Complex Fast Fourier Transform - CFFT	C5510 DSK	C5510 Device Simulator	1.20	
Discrete Cosine Transform - 8x8 DCT	C5510 DSK	C5510 Device Simulator	0.61	
Inverse Discrete Cosine Transform - 8x8 IDCT	C5510DSK	C5510 Device Simulator	0.72	

In addition to the above, the C55x CPU simulator has been validated for cycle accuracy with respect to the RTL design using the design verification test suite.

The cycle variances for these measurements are found to be within 0% to 2% for a test suite consisting of more than 2000 tests.