TMS320C6000 Instruction Set Simulator

Technical Reference

Literature Number: SPRU600F
April 2005

Texas Instruments

Contents

List of Tables

Read This First

About This Manual

This manual provides the following information:

- Names the TMS320C6000™ (C6000™) digital signal processors (DSPs) that are supported by configurations of the TMS320C6000 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 bench marking 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

TMS320C6000 CPU and Instruction Set Reference Guide (SPRU189) describes the architecture, pipeline, instruction set, and interrupts for the TMS320C6000 DSPs.

TMS320C6000 Peripherals Reference Guide (SPRU190) describes common peripherals available on the TMS320C6000 DSPs. This book includes information on the internal data and program memories, the external memory interface (EMIF), the host port, serial ports, direct memory access (DMA), enhanced direct memory access (EDMA), expansion bus (XBUS), clocking and phase-locked loop (PLL), and the power-down modes.

TMS320C6000 Code Composer Studio Online Help is accessible through the Code Composer Studio™ Integrated Development Environment (IDE).

Trademarks

TMS320C6000, C6000, Code Composer Studio, TMS320C62x, TMS320C64x, TMS320C67x, C62x, C64x, C67x, RTDX, DSP/BIOS are trademarks of Texas Instruments.

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

Introduction to the TMS320C6000 Simulator

The TMS320C6000™ Instruction Set Simulator, available within the Code Composer Studio™ Integrated Development Environment (IDE) for TMS320C6000, simulates the TMS320C62x™, TMS320C64x™, and TMS320C67x™ generations of devices. This chapter lists the TMS320C6000 devices supported by the simulator and provides information to help in the selection of the best simulator configuration.

Topic **Page**

1.1 Features

The C6000™ simulators support the following features:

- TMS320C6000 CPU full instruction set architecture execution
- Support for C62x™, C64x™, and C67x™. Support for all devices is based on these cores.
- Port Connect, which also supports external peripheral simulation
- Pin Connect, which also supports external interrupt simulation
- Analysis events support for debug and analysis
- RTDX™ support
- DSP/BIOS™ real-time analysis support
- CPU resource conflict detection
- Rewind support for C6x CPU simulators

1.2 Supported Processors and Simulator Configurations

Table 1-1 lists the devices supported by the C6000 Instruction Set Simulator with the corresponding configuration to be selected under the Factory Board menu of Code Composer Studio Setup.

Table 1-1. Processors Supported by the C6000 Simulator

Processor	Code Composer Studio IDE Import Configuration(1)	CPU Modeled
TMS320C62x	C62xx CPU Cycle Accurate Simulator	TMS320C62x
TMS320C64x	C64xx CPU Cycle Accurate Simulator	TMS320C64x
TMS320C67x	C67xx CPU Cycle Accurate Simulator	TMS320C67x
TMS320C6211	C6211 Device Cycle Accurate Simulator	TMS320C62x
TMS320C6411	C6411 Device Cycle Accurate Simulator	TMS320C64x
TMS320C6412(2)	C6412 Device Cycle Accurate Simulator	TMS320C64x
TMS320C6414	C6414 Device Cycle Accurate Simulator	TMS320C64x
TMS320C6415	C6415 Device Cycle Accurate Simulator	TMS320C64x
TMS320C6416(2)	C6416 Device Cycle Accurate Simulator	TMS320C64x
TMS320C6711	C6711 Device Cycle Accurate Simulator	TMS320C67x
TMS320C6712	C6712 Device Cycle Accurate Simulator	TMS320C67x
TMS320C6713(2)	C6713 Device Cycle Accurate Simulator	TMS320C67x
TMS320DM642(2)	DM642 Device Cycle Accurate Simulator	TMS320C64x
TMS320C6201	C6201 Device Simulator(3)	TMS320C62x
TMS320C6202	C6202 Device Simulator(3)	TMS320C62x
TMS320C6203	C6203 Device Simulator(3)	TMS320C62x
TMS320C6204	C6204 Device Simulator(3)	TMS320C62x
TMS320C6205	C6205 Device Simulator(3)	TMS320C62x
TMS320C6701	C6701 Device Simulator(3)	TMS320C67x

Note:

Multiple C6000 simulator configurations cannot be imported in Code Composer Studio Setup.

(1) Big Endian and Little Endian (default) configurations are available in the product.
(2) Device Functional Simulators are also available for these Device Cycle Accurate Simulators and can be used by changing the Simulator Type entry in the Code Composer Studio Setup processor properties window from Cycle Accurate to Functional.
(3) Map 0 and Map 1 (default) configurations are available.

1.3 Considerations for Choosing a Simulator

The different simulators provide a tradeoff between the functionality modeled, cycle accuracy of the simulation, and simulation performance. Different configurations can be selected based on the needs of the application, in terms of the target functionality needed and the levels of cycle accuracy required. You can choose CPU Cycle Accurate Simulators or Device Cycle Accurate Simulators from the Factory Boards menu in Code Composer Studio Setup. For C6x1x simulators, you can create simulator configurations for Device Functional Simulators by changing the Simulator Type entry in the Code Composer Studio Setup processor properties window from Cycle Accurate to Functional.

The CPU cycle-accurate simulator configurations can be selected if your primary interest is in optimizing core algorithms. Here, you are concerned with the accuracy of the core and do not need full device simulation. Any accesses outside the core will be handled by a flat memory; the cycles measured will not account for any memory access latencies.

The device functional simulator configurations model the functionality of some of the key peripherals, with minimal impact on simulator performance. The peripherals in these simulators are modeled functionally to support the programmer view; they are not cycle-accurate. These configurations can be used when you are interested in the features of the device that are supported in the functional simulator but not present in the core simulator, and when cycle accuracy is not important.

For example, the C6713 Device Functional Simulator can be used for application development where cycle accuracy is not needed. The C6713 Device Functional Simulator runs faster than the C6713 Device Cycle Accurate Simulator.

The device cycle-accurate simulator configurations model most of the peripherals of the devices. The peripherals are cycle accurate, as in the silicon. These simulators can be used to get an indication of the cycle behavior of an application. For more details on the accuracy of a specific simulator configuration, see Chapter 3.

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 on simulator configurations, see Chapter 3.

1.4.1 CPU

The CPU model is cycle accurate. Cycle effects are modeled.

1.4.2 Memory

For each of the device simulators and device cycle-accurate simulators, the cache and internal memory models match the cache and memory architecture specifications described in the *TMS320C6000 Peripherals Reference Guide* (SPRU190). They support standard cache behavior such as: LRU line replacement, direct mapping, set associativity, cache protocols for hit/miss service, snoops, and victims. The allocation policies (allocate on read miss, write-back on write miss, etc.) are also modeled to match the device specifications. A flat memory model is hooked up to the EMIF in order to allow for external memory accesses.

For the CPU cycle-accurate simulators and device functional simulators, the whole memory range is modeled as a flat memory.

Note:
Setting up memory maps using either the GUI or GEL scripts is not supported.

1.4.3 Peripherals

The following tables show the peripherals supported under each simulator configuration. For more information on these peripherals, refer to the *TMS320C6000 Peripherals Reference Guide* (SPRU190).

Table 1-2. TMS320C6000 CPU Cycle Accurate Simulator Configurations

	C62x	C64x	C67x
CPU	Yes	Yes	Yes
Flat Memory	Yes	Yes	Yes
Timer	Yes	Yes	Yes

Table 1-3. TMS320C6201, C6202, C6203, C6204, C6205, and C6701 Simulators

	C6201	C6202	C6203	C6204	C6205	C6701
CPU	Yes	Yes	Yes	Yes	Yes	Yes
Internal Memory/Cache Model	Yes	Yes	Yes	Yes	Yes	Yes
DMA	Yes	Yes	Yes(1)	Yes(1)	Yes(1)	Yes
EMIF	Yes	Yes	Yes	Yes	Yes	Yes
Interrupt Selector	Yes	Yes	Yes	Yes	Yes	Yes
McBSP	Yes	Yes	Yes	Yes	Yes	Yes
Timer	Yes	Yes	Yes	Yes	Yes	Yes
GPIO	No	No	No	No	No	No
HPI/XBUS/PCI	No	No	No	No	No	No

(1) The DMA functionality modeled on this device is the same as that of the C6202 simulator. The enhancements have not been modeled.

Table 1-4. TMS320C6411, C6412, C6414, C6415, C6416, and DM642 Device Cycle Accurate; C6412, C6416 and DM642 Functional Simulators

	C6411	C6412	C6414	C6415	C6416	DM642	C6412 FUNC	C6416 FUNC	DM642 FUNC
CPU	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Internal Memory/ Cache Model	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
QDMA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EDMA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
EMIF	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Interrupt Selector	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
McBSP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Timer	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
GPIO	No	No	No	No	No	No	No	No	No
HPI	No	No	No	No	No	No	No	No	No
TCP/VCP	N/A(1)	N/A(1)	N/A(1)	N/A(1)	Yes	N/A(1)	N/A(1)	No	N/A(1)
Utopia	N/A(1)	N/A(1)	N/A(1)	No	No	N/A(1)	N/A(1)	No	N/A(1)
PCI	No	No	No	No	No	No	No	No	No
McASP	N/A(1)	N/A(1)	N/A(1)	N/A(1)	N/A(1)	Yes	N/A(1)	N/A(1)	No
Videoport	N/A(1)	N/A(1)	N/A(1)	N/A(1)	N/A(1)	No	N/A(1)	N/A(1)	No

(1) N/A - Does not exist in the hardware.

Table 1-5. TMS320C6211, C6711, C6712, and C6713 Device Cycle Accurate, and C6713 Device Functional Simulators

	C6211	C6711	C6712	C6713	C6713 FUNC
CPU	Yes	Yes	Yes	Yes	Yes
Internal Memory/Cache Model	Yes	Yes	Yes	Yes	Yes
QDMA	Yes	Yes	Yes	Yes	Yes
EDMA	Yes	Yes	Yes	Yes	Yes
EMIF	Yes	Yes	Yes	Yes	Yes
Interrupt Selector	Yes	Yes	Yes	Yes	Yes
McBSP	Yes	Yes	Yes	Yes	Yes
McASP	No	No	No	Yes	No
Timer	Yes	Yes	Yes	Yes	Yes
GPIO	No	No	No	No	No
HPI	No	No	No	No	No

Supported Simulation Features

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

Topic **Page**

2.1 External Event and Data Simulation

In real hardware, the DSP interacts with many external entities. The simulator provides features to simulate these interactions. 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, the serial port of the DSP in an audio device 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 into the Data Values category.

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

2.1.1 Pin Connect

The Pin Connect tool allows you to simulate and monitor signals from external interrupts. For taking in external interrupts/triggers, some pins are simulated in the TMS320C6000 devices. Any file with the specified format can be connected to those pins.

The pins being simulated are of two types: pulse and waveform. Pulse pins are sensitive to rising edges and Waveform pins are sensitive to both rising and falling edges. For example, CPU interrupt pins are sensitive to rising edges, while the clock input pins of the serial port (CLKX, CLKR) of the TMS320C6000 devices are sensitive to rising and falling edges. Refer to the following tables for details on the various pins supported for the different simulator configurations.

Table 2-1. Available Memory Ranges for Port Connect for TMS320C6000 Devices

Pin	Description	Type
NMI	Non-maskable interrupt	Pulse
INT4	General purpose external interrupt pin	Pulse
INT5	General purpose external interrupt pin	Pulse
INT6	General purpose external interrupt pin	Pulse
INT7	General purpose external interrupt pin	Pulse
TINP0	Timer0 input pin	Waveform
TINP1	Timer1 input pin	Waveform
FSX0	Transmit frame synchronization pin for McBSP0	Waveform
FSR0	Receive frame synchronization pin for McBSP0	Waveform
CLKX0	Transmit clock pin for McBSP0	Waveform
CLKR0	Receive clock pin for McBSP0	Waveform
CLKS0	External clock input for McBSP0	Waveform
FSX1	Transmit frame synchronization pin for McBSP1	Waveform
FSR1	Receive frame synchronization pin for McBSP1	Waveform
CLKX1	Transmit clock pin for McBSP1	Waveform
CLKR1	Receive clock pin for McBSP1	Waveform
CLKS1	External clock input for McBSP1	Waveform

Table 2-2. Available Pins for Configuration of the TMS320C6713 Device

Pin	Description	Type
NMI	Non-maskable interrupt	Pulse
INT4	General purpose external interrupt pin	Pulse
INT5	General purpose external interrupt pin	Pulse
INT6	General purpose external interrupt pin	Pulse
INT7	General purpose external interrupt pin	Pulse
TINP0	Timer0 input pin	Waveform
TINP1	Timer1 input pin	Waveform
FSX0	Transmit frame synchronization pin for McBSP0	Waveform
FSR0	Receive frame synchronization pin for McBSP0	Waveform
CLKX0	Transmit clock pin for McBSP0	Waveform
CLKR0	Receive clock pin for McBSP0	Waveform
CLKS0	External clock input for McBSP0	Waveform
FSX1	Transmit frame synchronization pin for McBSP1	Waveform
FSR1	Receive frame synchronization pin for McBSP1	Waveform
CLKX1	Transmit clock pin for McBSP1	Waveform
CLKR1	Receive clock pin for McBSP1	Waveform
CLKS1	External clock input for McBSP1	Waveform
ACLKX0	Clock pin for transmit section of McASP0	Pulse
ACLKR0	Clock pin for receive section of McASP0	Pulse
AHCLKX0	Clock pin at higher frequencies for transmit section of McASP0	Pulse
AHCLKR0	Clock pin at higher frequencies for receive section of McASP0	Pulse
AFSX0	Frame sync pin for transmit section of McASP0	Pulse
AFSR0	Frame sync pin for receive section of McASP0	Pulse
ACLKX1	Clock pin for transmit section of McASP1	Pulse
ACLKR1	Clock pin for receive section of McASP1	Pulse
AHCLKX1	Clock pin at higher frequencies for transmit section of McASP1	Pulse
AHCLKR1	Clock pin at higher frequencies for receive section of McASP1	Pulse
AFSX1	Frame sync pin for transmit section of McASP1	Pulse
AFSR1	Frame sync pin for receive section of McASP1	Pulse

Table 2-3. Available Pins for Configuration of TMS320C6202, C6203, C6412, C6414, C6415, C6416 and DM642 Devices

Pin	Description	Type
NMI	Non-maskable interrupt	Pulse
INT4	General purpose external interrupt pin	Pulse
INT5	General purpose external interrupt pin	Pulse
INT6	General purpose external interrupt pin	Pulse
INT7	General purpose external interrupt pin	Pulse
TINP0(1)	Timer0 input pin	Waveform
TINP1(1)	Timer1 input pin	Waveform
FSX0	Transmit frame synchronization pin for McBSP0	Waveform
FSR0	Receive frame synchronization pin for McBSP0	Waveform
CLKX0	Transmit clock pin for McBSP0	Waveform
CLKR0	Receive clock pin for McBSP0	Waveform
CLKS0	External clock input for McBSP0	Waveform
FSX1	Transmit frame synchronization pin for McBSP1	Waveform
FSR1	Receive frame synchronization pin for McBSP1	Waveform
CLKX1	Transmit clock pin for McBSP1	Waveform
CLKR1	Receive clock pin for McBSP1	Waveform
CLKS1	External clock input for McBSP1	Waveform
FSX2(2)	Transmit frame synchronization pin for McBSP2	Waveform
FSR2(2)	Receive frame synchronization pin for McBSP2	Waveform
CLKX2(2)	Transmit clock pin for McBSP2	Waveform
CLKR2(2)	Receive clock pin for McBSP2	Waveform
CLKS2(2)	External clock input for McBSP2	Waveform
ACLKR0(3)	Clock pin for receive section of McASP0	Pulse
ACLKX0(3)	Clock pin for transmit section of McASP0	Pulse
AHCLKR0(3)	Clock pin at higher frequencies for receive section of McASP0	Pulse
AHCLKX0(3)	Clock pin at higher frequencies for transmit section of McASP0	Pulse
AFSR0(3)	Frame sync pin for receive section of McASP0	Pulse
AFSX0(3)	Frame sync pin for transmit section of McASP0	Pulse

(1) This pin is applicable only to the C6202 and C6203 Device Simulators.
(2) This pin is not applicable to the DM642 Device Cycle Accurate Simulator, as it has only two McBSPs.
(3) This pin is applicable only to the DM642 Device Cycle Accurate Simulator.

Table 2-4. Available Pins for Configuration of TMS320C64x, C62x, and C67x CPU Cycle Accurate Simulators

Pin	Description	Type
NMI	Non-maskable interrupt	Pulse
INT4	General purpose external interrupt pin	Pulse
INT5	General purpose external interrupt pin	Pulse
INT6	General purpose external interrupt pin	Pulse
INT7	General purpose external interrupt pin	Pulse

Table 2-5. Available Pins for Configuration of TMS320C6412, C6416, C6713, and DM642 Device Functional Simulators

Pin	Description	Type
NMI	Non-maskable interrupt	Pulse
FSX0	Transmit frame synchronization pin for McBSP0	Pulse
FSR0	Receive frame synchronization pin for McBSP0	Pulse
CLKX0	Transmit clock pin for McBSP0	Pulse
CLKR0	Receive clock pin for McBSP0	Pulse
CLKS0	External clock input for McBSP0	Pulse
FSX1	Transmit frame synchronization pin for McBSP1	Pulse
FSR1	Receive frame synchronization pin for McBSP1	Pulse
CLKX1	Transmit clock pin for McBSP1	Pulse
CLKR1	Receive clock pin for McBSP1	Pulse
CLKS1	External clock input for McBSP1	Pulse
FSX2(1)	Transmit frame synchronization pin for McBSP2	Pulse
FSR2(1)	Receive frame synchronization pin for McBSP2	Pulse
CLKX2(1)	Transmit clock pin for McBSP2	Pulse
CLKR2(1)	Receive clock pin for McBSP2	Pulse
CLKS2(1)	External clock input for McBSP2	Pulse

(1) This pin is available only on the C6416 Device Functional Simulator.

The Pin Connect feature can be enabled through the Code Composer Studio command window, GEL commands, or through the Pin Connect plug-in. See Section 4.9 for more information on the Pin Connect file format. See Code Composer Studio online help for more information on how to connect and disconnect a specific pin.

2.1.2 Port Connect

The Port Connect feature allows you to feed in external data to the simulator and send simulator data out to an external entity. External data is fed into and out of the simulator (memory or serial port) through Port Connect files (the file format is discussed in Section 4.10). This feature can be used to setup an input or output data stream to the simulator at the supported address. Whenever a file is connected to a memory (port) address for reads, data from the file is accessed whenever there is a read to the memory in the device. Similarly, whenever a file is connected to a memory (port) address for writes, all data writes to that address will be written to the file. The simulator provides Port Connect for all processor configurations.

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 the serial port receive register in read mode. Available memory ranges on the TMS320C6000 devices, to which a file can be connected for reading/writing, are given in Table 2-6.

Table 2-6. Available Memory Ranges for Port Connect for TMS320C6000 Devices

Configuration File (1)	Available Memory Range
C6201 Device Simulator, Map 1	0x0040 0000 - 0x017F FFFF DXR0, DRR0, DXR1, DRR1(2)
C6201 Device Simulator, Map 0	0x0000 0000 - 0x013F FFFF DXR0, DRR0, DXR1, DRR1(2)

(1) For each configuration listed, Little Endian (default) and Big Endian versions are provided with the same available memory range for the serial port connection.

(2) To connect to these peripheral registers, use the memory-mapped addresses. Port connects to the McBSP can be done only to the following locations: 0x3000 0000 for DX0 and DR0, 0x3400 0000 for DX1 and DR1, and 0x3800 0000 for DX2 and DR2.

Table 2-6. Available Memory Ranges for Port Connect for TMS320C6000 Devices (continued)

Configuration File [1]	Available Memory Range
C6202 Device Simulator, Map 1	0x0040 0000 - 0x017F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6202 Device Simulator, Map 0	0x0000 0000 - 0x013F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6203 Device Simulator, Map 1	0x0040 0000 - 0x017F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6203 Device Simulator, Map 0	0x0000 0000 - 0x013F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6204 Device Simulator, Map 1	0x0040 0000 - 0x017F FFFF DXR0, DRR0, DXR1, DRR1[2]
C6204 Device Simulator, Map 0	0x0000 0000 - 0x013F FFFF DXR0, DRR0, DXR1, DRR1[2]
C6205 Device Simulator, Map 1	0x0040 0000 - 0x017F FFFF DXR0, DRR0, DXR1, DRR1[2]
C6205 Device Simulator, Map 0	0x0000 0000 - 0x013F FFFF DXR0, DRR0, DXR1, DRR1[2]
C6411 Device Cycle Accurate Simulator	0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1[2]
C6412 Device Cycle Accurate Simulator	0x0800 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1[2]
C6412 Device Functional Simulator	0x8000 0000 - 0xBFFF FFFF
C6414 Device Cycle Accurate Simulator	0x6000 0000 - 0x6FFF FFFF 0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6415 Device Cycle Accurate Simulator	0x6000 0000 - 0x6FFF FFFF 0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6416 Device Cycle Accurate Simulator	0x6000 0000 - 0x6FFF FFFF 0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6416 Device Functional Simulator	0x6000 0000 - 0x6FFF FFFF 0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2[2]
C6701 Device Simulator, Map 1	0x0040 0000 - 0x017F FFFF DXR0, DRR0, DXR1, DRR1[2]
C6701 Device Simulator, Map 0	0x0000 0000 - 0x013F FFFF DXR0, DRR0, DXR1, DRR1[2]
C6211 Device Cycle Accurate Simulator	0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1[2]
C6711 Device Cycle Accurate Simulator	0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1[2]
C6713 Device Cycle Accurate Simulator	0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1, McASP0 TXBUF0-15, McASP0 RXBUF0-15, McASP1 TXBUF0-15, McASP1 RXBUF0-15[2]
C6713 Device Functional Simulator	0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1[2]
DM642 Device Cycle Accurate Simulator	0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1, McASP0 TXBUF0-15, McASP0 RXBUF0-15[2]
DM642 Device Functional Simulator	0x8000 0000 - 0xBFFF FFFF DXR0, DRR0, DXR1, DRR1[2]
C62xx CPU Cycle Accurate Simulator	0x0000 0000 - 0xFFFF FFFF
C64xx CPU Cycle Accurate Simulator	0x0000 0000 - 0xFFFF FFFF
C67xx CPU Cycle Accurate Simulator	0x0000 0000 - 0xFFFF FFFF

2.1.2.1 Read Modes

These two modes are used to connect a Port Connect file:

- **Rewind mode** is the default connection mode. 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 further read accesses. For example, if the sample file below is used for address 0x2000 in read mode, at the time of the sixth read access to that address, 0x12346666 is read.
- In **No-Rewind mode** read accesses made after end-of-file do not result in file reads. The data read is whatever is present in the memory. For example, if the sample file below is used for address 0x2000 in read mode, at the time of the sixth read access to that address, the value read would be that in the memory, which is 0x89897f7f (from the fifth access).

A sample file for a C6000 target (word length, 4 bytes):

```
12346666
33449999
cb56aaaa
5656cccc
89897f7f
```

2.1.2.2 Reset

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

Port Connect can occur through the Code Composer Studio command window, GEL commands, or through 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 Reserved Memory Access Detection

Accesses by DSP applications to reserved memory locations of the device can result in undefined program behavior. The simulators can help detect such violations by flagging an error message whenever the application accesses reserved memory address for that device.

If a peripheral is modeled in the simulator, any access to this peripheral is also treated as a reserved memory access. Any write access by the simulator to such locations have no effect and reads return zero; for instance, TCP/VCP memory map addresses on a C6146 device functional simulator.

2.2.1 Supported Configurations

This feature is available on all C6x1x simulators.

It is not supported on any of the C6x0x simulators.

CPU simulators do not have any reserved memory modeled.

2.2.2 Configuration Options

The simulators can be configured for reserved memory access detection from Code Composer Studio Setup in the following modes:

- **YES**. All reserved memory accesses are detected and flagged as errors, along with the address of access. Simulation is halted.
- **NO**. No reserved memory access detection is used. Writes to the reserved memories have no effect, while reads return 0.
- **Create Log**. All reserved memory accesses are detected and logged in a text file, along with the address of the access. Simulation is not halted.

See Section 4.2 for more information.

2.2.3 Error Reporting Format

If reserved memory access detection is turned on, the errors are reported/logged in the following formats:

Access to Reserve Memory Ranges:

Memory Map Error: {**READ** | **WRITE**} access to address *AAAAAAAA*, which is RESERVED in Hardware.

For example:

```
Memory Map Error: WRITE access to address 0xc7fffc, which is RESERVED in Hardware.
```

Access to Unsupported Memory Ranges:

Memory Map Error: {**READ** | **WRITE**} access to address *AAAAAAAA*, which is NOT SUPPORTED by the simulator.

For example:

```
Memory Map Error: READ access to address 0x1880000, which is NOT SUPPORTED in Simulator.
```

2.2.4 Limitations

This feature does not support the addition of any new reserved area ranges (through the Code Composer Studio Memory Map feature) in addition to the ones as that are reserved in the hardware. Neither does it allow for selective removal of any such reserved memory regions.

2.3 CPU Resource Conflict Detection

The C6000 CPUs have four functional units on each of the two sides, A and B. These units offer enhanced parallelism and allow for execution of up to eight parallel instructions. These units, along with the Data Access Paths and the CPU registers, primarily constitute the resources of a C6000 CPU.

These resources have various constraints on their simultaneous use by different instructions. If these constraints are violated by the executing code, the application behavior is not guaranteed.

Resource conflict detection in the simulators is crucial because many of these violations manifest only at run time, making it impossible to detect them at compile time.

2.3.1 Supported Configurations

All C6000 simulators support resource conflict detection.

2.3.2 Types of Conflict Detected

Simulators detect the following types of conflict:

- Unit overusage (S, M, L, and D units)
- XPath and T unit overusage
- C67x multi-cycle unit usage constraints
- Multiple register writes
- Multiple multi-cycle NOPs in the same cycle
- Multiple branches in the same cycle

2.3.3 Types of Conflict Not Detected

Multiple (more than four) simultaneous reads to one register.

2.3.4 Configuration Options

The simulators can be configured for resource conflict detection from Code Composer Studio Setup in the following three modes:

- **YES**. All conflicts are detected and flagged as errors. Simulation is halted.
- **NO**. No resource conflict detection is used.
- **Create Log**. All conflicts are detected and error messages redirected to a file. Simulation is not halted.

See Section 4.1 for more information.

2.3.5 Error Reporting Format

On detection of resource conflicts, the simulator generates an error string which is either displayed in the error window or logged onto a file, depending on the configured mode. The error strings are reported in the following general format:

error type ******* *error message* **at PC =** *XXXXXXXX*
Resources YYY, *YYY* and *YYY***in***TypeOf***Conflict in***ZZZZ***phase.**
Ref *literature number* **Sec** *ABC*

For example:

```
Error Running Target CPU ***Runtime error at PC = 00000030
Register(s) B7 in Multiple Write Conflict in E1 phase.
Ref SPRU189 Sec 3.7, Sec 5.6
```

- The phase indicated, *ZZZZ*, is with respect to the instruction at the PC =*XXXXXXXX*.
- The section information displayed gives only a broad reference and depends on the device type.

Table 2-7 lists the valid Resource Names.

Table 2-7. Resources That Can Appear in the Resource Conflict Error Message

L Unit/Read Port	T Access Path	32 MSBPATH Load Path	Multi-cycle Nop
M Unit/Read Port	32 LSBPATH Store Path	L Unit Long Write Port	Control Register File
D Unit/Read Port	32 MSBPATH Store Path	M Unit Long Write Port	Shared 32 LSB Store Path/L/S Unit Long Write Port
S Unit/Read Port	L Unit Write Port	S Unit Long Write Port	Shared 32 LSB Store Path/L/S Unit Long Write Port
L Unit Long Read Port	M Unit Write Port	Internal M Unit Resource V	Shared L/S Unit Long Write Port Registers A# and B#
M Unit Long Read Port	S Unit Write Port	Internal M Unit Resource U	Invalid Resource
S Unit Long Read Port	32 LSBPATH Load Path	XPath	

2.4 Simulator Analysis

The TMS320C6000 Simulator Analysis allows you to set up and monitor the occurrence of specific events. Some of the simulated events are program cache miss, program cache hit, program fetch, program access block 0, and program access block 1. The Simulator Analysis plug-in reports the occurrence of particular system events so that you can accurately monitor and measure the performance of the program. The events can be set up either to increment a counter or to halt the execution when they occur.

The capability to halt execution on an event can be used to debug the execution of the application. For example, by setting up a C6211 target to halt when a Timer0 sync event to EDMA is triggered, you can debug if the sync event is happening correctly or not.

The ability to count events over a period of execution will give you an overview of program behavior during the 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 for the list of analysis events available for each of the configurations and how to enable them through the Simulator Analysis plug-in.

2.5 RTDX

Real-Time Data Exchange (RTDX) is supported when running on the simulator. To run an RTDX application that uses the simulator, you must link applications with the RTDX Simulator Target library. It is easy to switch applications from running on the simulator to running on real hardware. For more information on RTDX, see the Code Composer Studio IDE online help.

2.6 DSP/BIOS

All applications using DSP/BIOS can be run on all the C6000 simulators. In order to enable real-time analysis for these applications, one needs to ensure that the RTDX Mode in the configuration is set to simulator. Please refer to the Code Composer Studio IDE online help topics on DSP/BIOS for more information.

2.7 Bootload

The Bootload is a process that copies a finite number of words (the exact number differs for C620x/C670x and C621x/C64x devices) from an address specified by the bootmode to address 0x0. Bootload happens in the simulator only if it is enabled by specifying a valid Bootmode through a simulator base configuration file. See Section 4.3 for more information.

2.8 Application Memory Usage Detection

This feature enables the detection of large memory usage by applications. The simulator reports an error if the application memory usage goes beyond a specified limit, the default value being 64 MB. You can override this default limit through a base configuration file-based option. See Chapter 4 for more information.

Note:
Application memory usage detection is not available on C6x0x simulators.

2.9 EMIF Clock Configuration

The following simulators support the programmability of the EMIF and CPU clock speeds through Code Composer Studio Setup:

- C6211 Device Cycle Accurate Simulator
- C6711/C6712/C6713 Device Cycle Accurate Simulators
- C6411/C6412/C6414/C6415/C6416/DM642 Device Cycle Accurate Simulators

For more details, see Section 4.4.

2.10 Rewind

The following 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.

- C62xx CPU Cycle Accurate Simulator
- C64xx CPU Cycle Accurate Simulator
- C67xx CPU Cycle Accurate Simulator

See Section 4.5 for details on enabling this feature on CPU simulators.

Detailed Capabilities of Individual Configurations

This chapter describes the capabilities and known limitations of each simulator configuration.

Topic **Page**

3.1 C62x/C64x/C67x CPU Cycle Accurate Simulators

In the CPU cycle-accurate simulator configurations, only the CPU core is modeled, along with device timer support and a flat memory model for the full addressable space. These simulators can be used for algorithmic verification if the functionality of the device peripherals is not needed. These configurations are also faster than the other simulator configurations available. These simulators also support Rewind (see Section 4.5), which is helpful for quicker debugging of applications.

All the instructions described in the *TMS320C6000 CPU and Instruction Set Reference Guide* (SPRU189) are supported. This includes support for all registers, addressing modes, branch conditions, parallel instructions, and floating point instructions. The CPU core model is cycle accurate and complete pipeline effects are modeled. Each simulator configuration supports all the user-visible registers (the data register sets A and B, and all the control registers). All of the interrupts are supported.

The CPU cycle accurate simulators have a cycle-accurate Timer model. They also support simulating applications that use RTDX. It is possible to run DSP/BIOS-based applications on these simulators and use the real-time analysis capabilities for debug and analysis of the application. Additionally, the Pin Connect and the Port Connect features are available to set up external stimuli required by the application for simulation.

The list of pins supported and the ranges of memory that can be connected to a file for these configurations are specified in Chapter 2. Using the Simulator Analysis plug-in, these simulator configurations can track many of the target events and allow you to count or break on these events. Please refer to the Code Composer Studio IDE online documentation for more details on the list of events and how to use them.

3.2 C6416/C6713/C6412/DM642 Device Functional Simulators

The device functional simulators model the functionality of the peripherals without modeling their full cycle behavior. This allows these simulators to be faster than their corresponding cycle-accurate models, while functionally still allowing you to run applications that make use of these peripherals. These simulators can be used for verifying the application functionally and for measuring the clock cycles that the application would require.

3.2.1 Supported Features

The DM642, C6416, and C6713 Device Functional Simulators have the cycle-accurate CPU; cycle-accurate Timer module; functional models of the L1P, L1D, and L2 caches; a functional model of the Interrupt Selector; functional model of the McBSP; functional model of the EMIF; and a functional model of the EDMA. All memory beyond cache is modeled as flat memory.

The C6416 Device Functional Simulator can be used to simulate C6414/15 devices functionally. The C6713 Device Functional Simulator can be used to simulate the C6711, C6712, and C6211 configurations functionally. The C6412 Device Functional Simulator can be used to simulate the C6411 configuration functionally.

Applications that use RTDX and DSP/BIOS real-time analysis can be run on these simulators. Additionally, the Pin Connect and the Port Connect features are available to set up external stimuli required by the application for simulation.

The list of pins supported and the ranges of memory that can be connected to a file for these configurations are specified in Chapter 2. These simulator configurations can track many of the target events and allow you to count or break on these events. Please refer to the Code Composer Studio IDE online documentation for more details on the list of events and how to use them.

3.2.2 Known Limitations

The simulators model only the Timer, L1 and L2 Caches, Interrupt Selector, EMIF, McBSP, and EDMA peripherals.

- McBSP Limitations
  - Functionality related to the FREE & SOFT field in the SPCR registers is not supported, since these are emulation features.
  - The enabling of the extra delay for the DX turn-on time is not available. The TRISTATE indication feature is not available in the Functional Device Simulator. Therefore, the functionality of the DXENA register field in the SPCR register is not supported.
  - The notion of rising/falling edge triggered activities is not available in the functional device simulator. Only the pulse type of input is supported; therefore, the functionality of the following register fields is not supported: FSXP, FSRP, CLKXP, CLKRP (all in the PCR register), and CLKSP (in the SRGR register).
  - The General Purpose I/O mode is not supported. Therefore, the functionality of the following register fields is not supported: XIOEN, RIOEN, CLKS_STAT, DX_STAT, and DR_STAT (all in the PCR register).
  - The concept of Data Delay is not supported. Therefore, the functionality of the RDATDLY in the RCR register and XDATDLY in the XCR register is not available. Data transmission/reception will begin immediately following the arrival of FSX/FSR, respectively.
- EMIF Limitations
  - All the data transfer operations in the DSP are modeled as one-time transfers in the device functional simulator, that is, all data transfers are treated as though they are happening with ZERO cycle read latency. Therefore, the functionality of all register fields in the SDEXT register is not available.
  - The use of the COUNTER field in the SDTIM register as a general-purpose timer is not supported.
  - The refreshing of the SDRAM is not supported in the device functional simulator. Therefore, the functionality of all register fields pertaining to SDRAM refresh operation is not available.
- Cache Limitations
  - The cache in the device functional simulator is modeled as a *dataless cache*; that is, the L1P, L1D, and L2 caches do not store data, but only tag information. Any modifications to L1D and L2 will be directly reflected in actual memory. Therefore, results of cache operations such as cache clean and cache invalidate can be different. It is always recommended to use the reset-reload sequence followed by the restart command to see the correct behavior.
- Handling of Reserved Spaces
  - In the device functional simulator, whenever you write to a reserved location it will be treated as writing of 0x0000 0000 to the location. Similarly, whenever you read the data value from a reserved location, only 0x0000 0000 will be read.
- ROM Access Behavior
  - The 6713 Device Functional Simulator does not generate an error message when you write to the area designated as ROM (0x0186 0000 - 0x0187 FFF0).

As mentioned previously, simulations on these configurations are not expected to be cycle accurate.

3.3 C6201/C6202/C6203/C6204/C6205/C6701 Device Simulators

These simulators model the CPU core, the Internal Program (alternate Cache) and Data memory controllers, DMA, EMIF, Interrupt Selector, McBSPs, and Timer. Arbitration between the CPU and the DMA for internal program memory (alternate cache) and internal data memory accesses, is modeled. The DMA supports all features as present in the device, such as four channels, resource arbitration and priority configuration, split channel operation, and DMA access to peripheral configuration range. The external stimuli of the McBSP can be set up using either the Pin Connect feature or the McBSP XBAR. For more information, see Section 4.9 and Section 4.6, respectively.

3.3.1 Supported Features

The C6x0x Device Simulators have the cycle-accurate CPU and the cycle-accurate Timer modules. The L1P, internal program and data memories, and EMIF subsystem are also modeled accurately. The DMA subsystem is accurate for the C6201 Device Simulator.

Pin Connect, Port Connect, and Analysis event features are supported.

The list of pins supported and the ranges of memory that can be connected to a file for these configurations are specified in Section 2.1. Using the Simulator Analysis plugin, these simulator configurations can track many of the target events and allow you to count or break on these events. Please refer to the Code Composer Studio IDE online documentation for more details on the list of events and how to use them.

3.3.2 Known Limitations

- General
  - The HPI is not modeled.
  - XBUS is not modeled.
  - GPIO is not modeled.
- DMA
  - The auxiliary channel (channel 5), meant to service HPI requests, is not supported.
  - In C6203, C6204, and C6205, the DMA FIFO to hold data coming from a high-performance source (e.g., internal memory) is shared among all four channels (whereas it is meant to be dedicated for each channel).
  - The synchronization events, DSPINT and SDRAM_INT, are not supported.
  - The synchronization events are captured only if they are enabled.
- EMIF
  - The SDRAM interrupt is not modeled for these configurations.
- Cycle Accuracy
  - Total cycles are not guaranteed to be accurate.

3.4 C6211/C6711/C6712/C6713 Device Cycle Accurate Simulators

These simulators model the CPU core, L1P and L1D Caches, L2 Cache/SRAM memory, EDMA, EMIF, Interrupt selector, McBSPs, and Timer. The C6713 simulator additionally models the McASP modules. The memory external to the EMIF has been modeled to mimic the expected memory behavior based on the EMIF settings for the corresponding CE space. The C6713 simulator also models, in addition to the above, S2, the ROM, and ROM patching.

3.4.1 Supported Features

The bootload feature is supported by the simulator.

The C6x1x Device Cycle Accurate Simulators have the cycle-accurate CPU and the cycle-accurate Timer module. The L1 and L2 are also modeled accurately. The EDMA subsystem and the EMIFs, although not accurate, will take cycles proportional to what will be taken on design, depending on transfer types, memory parameters, etc.

Applications that use RTDX and DSP/BIOS real-time analysis can be run on these simulators. Additionally, the Pin Connect and the Port Connect features are available to set up external stimuli required by the application for simulation.

The list of pins supported and the ranges of memory that can be connected to a file for these configurations are specified in Section 2.1. These simulator configurations can track man y of the target events and allow you to count or break on these events. Please refer to the Code Composer Studio IDE online documentation for more details on the list of events and how to use them.

3.4.2 Known Limitations

- General
  - The HPI is not modeled.
  - GPIO is not modeled.
- DMA
  - The synchronization events, DSPINT and SDRAM_INT, are not supported.
- EMIF
  - There is no support for the SDRAM interrupt.
- Cache
  - The functionality of the L2CLEAN register is not supported.

3.5 C6411/C6412/C6414/C6415/C6416/DM642 Device Cycle Accurate Simulators

These simulators model the CPU core, L1P and L1D Caches, L2 Cache/SRAM, EDMA, EMIFA, EMIFB, Interrupt selector, McBSPs and Timer. The memory external to the EMIF has been modeled to mimic the expected memory behavior based on the EMIF settings for the corresponding CE space. EMIFB and TCP/VCP support is not available for C6411/DM642 Device Cycle Accurate Simulators.

3.5.1 Supported Features

The C641x Device Cycle Accurate Simulators have the cycle-accurate CPU module, cycle-accurate Timer module. The L1 and L2 Cache models are modeled accurately. The EDMA subsystem and the EMIF will consume cycles proportional to what they consume on design for different parameters.

Applications that use RTDX and DSP/BIOS real-time analysis can be run on these simulators. Additionally, the Pin Connect and the Port Connect features are available to setup up external stimuli required by the application for simulation.

The list of pins supported and the ranges of memory that can be connected to a file for these configurations are specified in Section 2.1. These simulator configurations can track many of the target events and allow you to count or break on these events. Please refer to the Code Composer Studio IDE online documentation for more details on the list of events and how to use them.

3.5.2 Known Limitations

- General
  - The HPI/PCI is not modeled.
  - Utopia is not modeled.
  - GPIO is not supported.
- EDMA
  - Event polarity in XDMA is not supported. Therefore, events are taken only in active high state.
  - Push Data Transfer feature is not supported.
  - The synchronization events DSPINT, GPIOINT, SD_INTA, SD_INTB, PCI, UREVT, and UXEVT are not modeled.
- Cache
  - L2 does not submit Transfer Requests (TR) on all priorities, only on priority 0.
  - The cycle effects of L1P pipelined misses to L2SRAM are not accurate.
- TCP/VCP Coprocessors
  - No support for Pause/UnPause conditions.

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 4.11.)

Topic **Page**

4.1 Setting the Resource Conflict Detection Mode

CPU Resource Conflict Detection is a feature that allows you to find any resource problem with the use of CPU registers and functional units. If you are confident that there is no resource conflict problem in the code, this feature can be turned off to further enhance simulation speed.

This feature can be configured through Code Composer Studio Setup by setting one of the following options from the Detect CPU Resource Conflicts entry in the processor properties window:

- **Yes**. The simulator detects all resource conflicts and flags them as errors to you. Whenever an error occurs, the simulation is halted and control returns to you. You can then choose to correct the error, reset the program and run again; or to proceed. If you proceed, a precautionary error message is displayed, click on the Run option to resume simulation.

  Note:
  YES is the default mode. If nothing is specified in the base configuration file (see Section 4.11), simulators come up in this mode of execution.

- **No**. The simulator does not detect any resource conflicts.
- **Create Log**. The simulator detects all resource conflicts but does not flag them as errors to you. Instead, they are written to a file. To specify the file location, use the Log File entry.
- **Log File** : *Fully qualified filename*. If no filename is specified, the errors generated in File mode are logged in a file named resource_errors.log, located in the CCStudio\drivers directory.

  Whenever the Code Composer Studio IDE is re-invoked, the error log file is overwritten. However, if multiple errors occur during one or more runs in one invocation of CCStudio, all errors are written to the file.

4.2 Setting the Reserved Memory Access Detection Mode

By default, the detection of reserved memory access is enabled on all simulators. This feature can be configured through Code Composer Studio Setup by setting one of the following options from the Detect Reserved Memory Access entry in the processor properties window:

- **Yes**. The simulator halts whenever the program accesses a reserved memory location. It displays an error indicating the nature and address of the access. This is the default choice.
- **No**. The simulator does not report any reserved memory accesses. In this mode, writes to the reserved memories have no effect, while reads return 0.

  Note:
  Any debug accesses to reserved memory locations (through Memory Windows) will not cause errors. Debug writes will have no effect and debug reads will always return 0.

- **Create Log**. The simulator routes all error messages to a file without halting simulation. To specify the file to save the errors into, use the Log File entry.
- **Log File**: *Fully qualified filename*. If no filename is specified, the errors generated in File mode are logged in a file named res_mem_access_errors.log, located in the CCStudio\drivers directory.

  Whenever the Code Composer Studio IDE is re-invoked, the error log file is overwritten. However, if multiple errors occur during one or more runs in one invocation of CCStudio, all errors are written to the file.

4.3 Setting the Bootload

4.3.1 Bootload in C6x0x Device Simulators

The following are the Bootmodes supported in the C6x0x Device Simulators:

- ROM_8BIT
- ROM_16BIT
- ROM_32BIT

The Bootmode for C6x0x Device Simulators is not configurable via the Code Composer Studio Setup interface. You must manually add a BOOTSRC entry in the desired simulator's base configuration file (see Section 4.11) to use the Bootload feature.

For example, to choose ROM_8BIT Bootmode in the C6202 Device Simulator, the base configuration file must include the following in order to enable the bootload:

```
MODULE C6202;
...
    BOOTSRC ROM_8BIT;
END C6202;
```

If the BOOTSRC flag is absent or set to NONE, no bootload will happen. If Bootmode is set to any of the three values listed above, DMA will copy 64K bytes from EMIF CE1 (which is address 0x01000000 in MAP0 and is address 0x01400000 in MAP1) to address 0. Apart from Bootmode, the type of memory sitting at address 0 can also be configured through the simulator base configuration file. To do so, the base configuration file should have the MEM0 switch set to a valid memory type, chosen from the following available values:

- ONCHIP
- SDRAM_8BIT
- SDRAM_16BIT
- SDRAM_32BIT
- SBSRAM_CLK2
- SBSRAM_CLK1

If not specified, the memory at address 0 is set to ONCHIP by default.

4.3.2 Bootload in C64x Device Cycle Accurate/Device Functional Simulators

In C64x Device Simulators, the process of bootload (if enabled from Code Composer Studio Setup) copies 1K byte (256 words) into the memory at address 0 (i.e., internal memory in this case).

For enabling bootload on C6411/DM642/C6414/C6415/C6416, BOOT Mode in Code Composer Studio Setup should be set to either of the following values:

- NONE
- EMIFB

If not specified, or set to NONE, or to any other invalid value through the base configuration file (see Section 4.11), BOOTMODE is set to NONE by default. This means that no Bootload will happen. Specifying EMIFB as the BOOTMODE causes the Bootload process to copy 256 words from the EMIFB CE1 space (0x64000000) to internal memory address 0 through EDMA.

Bootmode QUICK does a simple memcopy (not through EDMA, like Bootmodes EMIFB) of 256 words from EMIFA CE1 space (i.e., 0x90000000) to internal memory. QUICK Bootmode is therefore faster.

4.4 Setting the EMIF and CPU Clocks

The EMIF to CPU clock ratio on C6000 simulators must be set up based on the respective DSP developer board (DSK/TEB/EVM) characteristics. This is necessary to ensure that the cycles on the developer board match those on the simulator.

You must specify correct EMIF and CPU clock speeds in Code Composer Studio Setup.

For instance, consider a C6416 DSK whose CPU clock frequency is 600 MHz and EMIF clock frequency is 100 MHz. In order to match the characteristics of the C6416 Device Cycle Accurate Simulator with this DSK, the EMIF Clock (MHz) and CPU Clock (MHz) should be set to 100 and 600, respectively, from Code Composer Studio Setup.

Note:

If this step is not done, cycles will differ considerably between the simulator and the developer board for external memory accesses.

4.5 Enabling 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.

4.6 Setting Up the McBSP XBAR

The simulator provides a mechanism by which you can interconnect two McBSPs to test and validate code written for serial transfer. The XBAR (crossbar) serves as a test bench which can be programmed to set up this desired connectivity.

Typically, you program one McBSP for transmission and another McBSP for reception. Thus, the DX and DR pins of the corresponding McBSPs must be hooked up to one another. Similarly, clock and frame synchronization signals have to be interconnected. The XBAR component in the simulator allows specification and implementation of this connectivity.

The XBAR connectivity is specified in a XBAR data file. Section 4.6.2 describes the file format in detail. The path and file name of the XBAR data file must be included in a simulator base configuration file (see Section 4.11), which is selected in Code Composer Studio Setup. Subsequently, when Code Composer Studio IDE is invoked, the XBAR functionality will be effective.

4.6.1 How to Write an XBAR File

The supported pin names for an XBAR file are listed in Table 4-1:

Table 4-1. XBAR File Pin Names

Pin Name	Denoted Pins	Register
MCBSP0:DX	DX	McBSP0
MCBSP1:DX	DX	McBSP1
MCBSP2:DX	DX	McBSP2
MCBSP0:DR	DR	McBSP0
MCBSP1:DR	DR	McBSP1
MCBSP2:DR	DR	McBSP2
MCBSP0:CLKX	CLKX	McBSP0
MCBSP1:CLKX	CLKX	McBSP1
MCBSP2:CLKX	CLKX	McBSP2

Table 4-1. XBAR File Pin Names (continued)

Pin Name	Denoted Pins	Register
MCBSP0:CLKR	CLKR	McBSP0
MCBSP1:CLKR	CLKR	McBSP1
MCBSP2:CLKR	CLKR	McBSP2
MCBSP0:FSX	FSX	McBSP0
MCBSP1:FSX	FSX	McBSP1
MCBSP2:FSX	FSX	McBSP2
MCBSP0:FSR	FSR	McBSP0
MCBSP1:FSR	FSR	McBSP1
MCBSP2:FSR	FSR	McBSP2

1. Connectivity is specified by means of source pin-destination pin pair specifications. The driver-driven entity relation is specified on the same line as follows:
   – MCBSP0:FSX > MCBSP1:FSR
   – MCBSP0:DX > MCBSP1: DR
   – MCBSP0:CLKX > MCBSP1:CLKR
2. No comments are supported by the file syntax. No blank lines should be present between specification lines.
3. The connectivity can be mentioned in any order, but make sure that each pair specifies a driver-driven connection. For example, to mention MCBSP1:DX → MCBSP0:DR, MCBSP1:CLKX → MCBSP0:CLKR, MCBSP1:FSX → MCBSP0:FSR connectivity, please follow the valid XBAR connectivity:

Valid XBAR Connectivity	Invalid XBAR Connectivity
MCBSP1:DX > MCBSP0:DR	MCBSP1:DX > MCBSP1:CLKX
MCBSP1:CLKX > MCBSP0:CLKR	MCBSP1:FSX > MCBSP0:DR
MCBSP1:FSX > MCBSP0:FSR	MCBSP0:CLKR > MCBSP0:FSR

4. Ensure that there are no blank lines at the end of the file, as this might cause problems.

Note:
The old McBSP XBAR file format is supported for backward compatibility, but users are encouraged to use the new format.

4.6.2 Format of the Configuration File to be Picked Up

McBSP XBAR is not configurable via the Code Composer Studio Setup interface. You must manually add a MCBSP_XBAR_FILE entry in the desired simulator's base configuration file (see Section 4.11.)

In the case of C6x0x devices, add the entry MCBSP_XBAR_FILE as follows:

```
MODULE TB;
    MCBSP_XBAR_FILE <path_of_xbar_file_name>;
END TB;
```

On the other hand, in case of C64x devices, add the entry MCBSP_XBAR_FILE as follows:

```
MODULE C64xx;
...
    MCBSP_XBAR_FILE <path_of_xbar_file_name>; // path of xbar data file
END C64xx;
```

The value <path_of_xbar_file_name>.dat denotes the path and file name of the XBAR data file. For example:

```
MCBSP_XBAR_FILE C:\ccstudio\drivers\mcbsp_xbar.dat;
```

4.7 Setting Up the McASP XBAR

The simulator provides a mechanism for connecting two McASPs, just as it does for connecting two McBSPs. The format of the connections which make up the lines of the XBAR file is as follows:

SRC_DEV **:** *SRC_PIN* **>** *DST_DEV* **:** *DST_PIN*

Where

SRC_DEV is the device from which data goes out.

SRC_PIN is the pin that sends data.

DST_DEV is the device into which data is comes.

DST_PIN is the pin that receives data.

Here, the following limitations are to be noted:

- Peripheral supported: McASP0
- Pins supported: AXR0-AXR15, AFSR, AFSX
- Pins not supported: AMUTEIN, AMUTE, ACLKR, ACLKX, AHCLKR, AHCLKX

For example, the XBAR file could contain the following connections:

```
MCASP0:AXR8 > MCASP0:AXR0
MCASP0:AXR9 > MCASP0:AXR1
MCASP0:AXR10 > MCASP0:AXR2
MCASP0:AXR11 > MCASP0:AXR3
MCASP0:AXR12 > MCASP0:AXR4
MCASP0:AXR13 > MCASP0:AXR5
MCASP0:AXR14 > MCASP0:AXR6
MCASP0:AXR15 > MCASP0:AXR7
MCASP0:AFSX > MCASP0:AFSR
```

If no interconnections are desired, the file can be left blank.

4.7.1 Format of the Configuration File to be Picked Up

McASP XBAR is not configurable via the Code Composer Studio Setup interface. You must manually add a MCASP_XBAR_FILE entry in the desired simulator's base configuration file (see Section 4.11.)

In the case of C6x0x devices, add the entry MCASP_XBAR_FILE as follows:

```
MODULE TB;
    MCASP_XBAR_FILE <path_of_xbar_file_name>;
END TB;
```

On the other hand, in the case of C64x devices, add the entry MCASP_XBAR_FILE as follows:

```
MODULE C64xx;
...
    MCASP_XBAR_FILE <path_of_xbar_file_name>; // path of xbar data file
END C64xx;
```

The value <path_of_xbar_file_name> denotes the path and file name of the XBAR data file. For example:

```
MCASP_XBAR_FILE C:\ccstudio\drivers\mcasp_xbar.dat;
```

4.8 Setting the Maximum Memory Usage Limit

The default maximum memory usage limit of 64MB could be over-ridden by adding the following line in the base configuration file (see Section 4.11):

```
MEM_USAGE_LIMIT <max_mem_usage_in_MB>;
```

For example, in order to run a test case that uses between 50 and 100 MB of device memory space, the default limit of 64MB is not sufficient and needs to be over-ridden. For example, to set the maximum memory limit to100 MB for a C6416 Device Cycle Accurate Simulator configuration, go to the section MODULE C6416 in the base configuration file and add the entry MEM_USAGE_LIMIT 100 as shown below.

```
MODULE C6416;
...
    MEM_USAGE_LIMIT 100;
END C6416;
```

MEM_USAGE_LIMIT is not configurable via the Code Composer Studio Setup interface. You must manually add a MEM_USAGE_LIMIT entry in the desired simulator's base configuration file.

4.9 File Format for Pin Connect

The simulator allows you to simulate and monitor external interrupt signals.

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

To simulate external interrupts follow these steps:

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 the program.
5. Run the program.

4.9.1 Setting Up the Input File

To simulate external interrupts, you 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}]

Where

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., the FSX0 pin in the 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.

4.9.2 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.

4.9.3 Relative Clock Cycle

You 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. You can mix both relative and absolute values in the data file.

4.9.4 Repetition of Patterns for a Specified Number of Times

You 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.

4.9.5 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.

Note:
Comments are not supported in the pin connect file.

4.10 File Format for Port Connect

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 can be specified with a preceding 0x or without.

```
12346666
33449999
; This is a commented line
5655cccc ; This is a commented sentence
89897f7f
```

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

4.11 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 of certain features that are not configurable from Code Composer Studio Setup, such as advanced options for Rewind or settings for the Maximum Memory Usage Limit.

Performance Numbers

Table 5-1 shows the performance numbers of the simulator for different device configurations. These numbers were gathered on a PC with a 2.4-GHz Intel® Pentium® 4 processor and 512 MB of RAM.

Table 5-1. Performance Numbers of the C6000 Simulator

Simulator Configuration	Application	Speed Under Default Settings		Speed With Resource Conflict Detection Turned Off	
		KIPS(1)	KCPS(1)	KIPS(1)	KCPS(1)
C62xx/C67xx CPU Cycle Accurate Simulators	GSM Enhanced Full Rate Vocoder (GSMEFR)	3800	-	5600	-
C64xx CPU Cycle Accurate Simulator	GSMEFR	2700	-	3600	-
C641x/DM642 Device Functional Simulators	GSMEFR	2700	-	2750	-
C6713 Device Functional Simulator	GSMEFR	2600	-	3300	-
C6x0x Device Simulators	Reed-Solomon encoder and decoder	41	188	41	188
C6211/C6713 Device Cycle Accurate Simulators	GSMHR	74	644	74	644
	GSMEFR	9	195	9	195
	Performance Audio Application - PA3	100	294	102	311
C641x/DM642 Device Cycle Accurate Simulators	GSMHR	101	337	102	355
	GSMEFR	807	2344	892	2354
	Video & Imaging Application - MPEG2 Decoder	885	223	984	248
	Video & Imaging Application - H.263 Decoder	776	261	851	279

(1) KIPS = Kilo Instructions Per Second, KCPS = Kilo Cycles Per Second

Note:

- Performance numbers are given for more than one application when there are significant differences in performance data due to the nature of the application.
- For Device Cycle Accurate simulators, the CPU load significantly affects the cycles per second data. A lower CPU load means that for the same number of instructions, more cycles are spent in CPU stalls, thus increasing the cycles per second.

Cycle Accuracy

This chapter describes how the TMS320C6000 simulators have been validated for cycle accuracy using a benchmark suite of applications.

Topic **Page**

6.1 C6000 Simulators Benchmarking

The TMS320C6000 simulators have been validated for cycle accuracy using a benchmark suite of applications. The measurements on Device Cycle Accurate Simulators have been carried out in the following categories:

- CPU + L1Cache + SRAM
- CPU + L1Cache + L2Cache + EMIF
- CPU + L1Cache + L2Cache + EDMA + EMIF
- Full applications using all of the above models and peripherals

The details on the benchmarking data are provided in Table 6-1.

Table 6-1. Benchmarking Data for C6000 Simulators

Application/Kernel Used for Benchmarking	Referenced Hardware	Simulator Configuration	Percent Variance in Cycle Numbers
On-Chip Memory Accesses (CPU + L1Cache + SRAM)			
Image Analysis - Histogram Computation	C6416 DSK	C6416 Device Cycle Accurate Simulator	1.35
Picture Filtering - 3x3 Correlation with Rounding	C6416 DSK	C6416 Device Cycle Accurate Simulator	0
Image Analysis - Histogram Computation	C6713 DSK	C6713 Device Cycle Accurate Simulator	2.47
Picture Filtering - 3x3 Correlation with Rounding	C6713 DSK	C6713 Device Cycle Accurate Simulator	-0.37
Single Precision Fast Fourier Transform - SP_FFT	C6713 DSK	C6713 Device Cycle Accurate Simulator	-1.64
Single Precision Matrix Multiplication - SP_MATMUL	C6713 DSK	C6713 Device Cycle Accurate Simulator	-0.79
On-Chip and Off-Chip Memory Accesses (CPU + L1Cache + L2Cache + EMIF)			
Image Analysis - Histogram Computation	C6416 DSK	C6416 Device Cycle Accurate Simulator	1.56
Picture Filtering - 3x3 Correlation with Rounding	C6416 DSK	C6416 Device Cycle Accurate Simulator	-10.89
Image Analysis - Histogram Computation	C6713 DSK	C6713 Device Cycle Accurate Simulator	-8.83
Picture Filtering - 3x3 Correlation with Rounding	C6713 DSK	C6713 Device Cycle Accurate Simulator	12.66
Single Precision FIR Filtering (general purpose) - SP_FIR_GEN(L2SRAM)	C6713 DSK	C6713 Device Cycle Accurate Simulator	9.78
Single Precision FIR Filtering (general purpose) - SP_FIR_GEN(L2CACHE)	C6713 DSK	C6713 Device Cycle Accurate Simulator	-11.67
Single Precision Fast Fourier Transform - SP_FFT	C6713 DSK	C6713 Device Cycle Accurate Simulator	-35.31
Single Precision Matrix Multiplication - SP_MATMUL	C6713 DSK	C6713 Device Cycle Accurate Simulator	-9.76
Single Precision Weighted Vector Sum - SP_W_VEC(L2SRAM)	C6713 DSK	C6713 Device Cycle Accurate Simulator	11.29
Single Precision Weighted Vector Sum - SP_W_VEC(L2CACHE)	C6713 DSK	C6713 Device Cycle Accurate Simulator	-1.58

Table 6-1. Benchmarking Data for C6000 Simulators (continued)

Application/Kernel Used for Benchmarking	Referenced Hardware	Simulator Configuration	Percent Variance in Cycle Numbers
Memory Accesses using DMA (CPU + L1Cache + L2Cache + EDMA + EMIF)			
Image Analysis - Histogram Computation	C6416 DSK	C6416 Device Cycle Accurate Simulator	1.80
Picture Filtering - 3x3 Correlation with Rounding	C6416 DSK	C6416 Device Cycle Accurate Simulator	9.49
Image Analysis - Histogram Computation	C6713 DSK	C6713 Device Cycle Accurate Simulator	0.87
Picture Filtering - 3x3 Correlation with Rounding	C6713 DSK	C6713 Device Cycle Accurate Simulator	-0.55
Single Precision FIR Filtering (general purpose) - SP_FIR_GEN	C6713 DSK	C6713 Device Cycle Accurate Simulator	9.67
Single Precision Fast Fourier Transform - SP_FFT	C6713 DSK	C6713 Device Cycle Accurate Simulator	25.22
Single Precision Matrix Multiplication - SP_MATMUL	C6713 DSK	C6713 Device Cycle Accurate Simulator	9.58
Single Precision Weighted Vector Sum - SP_W_VEC	C6713 DSK	C6713 Device Cycle Accurate Simulator	6.64
Full Applications (CPU + L1Cache + L2Cache + EDMA + EMIF + Peripherals (McBSP, McASP))			
Video & Imaging Application - MPEG2 Decoder	C6416 DSK	C6416 Device Cycle Accurate Simulator	-0.12
Video & Imaging Application - H.263 Decoder	C6416 DSK	C6416 Device Cycle Accurate Simulator	1.46
Performance Audio Application - PA3	C6713 - PA3 Reference Board	C6713 Device Cycle Accurate Simulator	-4.66
Speech Application - PCM Voice Channel	Access Communication Processor Evaluation Platform (C6416 daughter card)	C6416 Device Cycle Accurate Simulator	1.02

6.2 Notes on Cycle Accuracy

When comparing cycles between a simulator and a developer board (DSK/EVM/TEB), ensure the following:

1. The Device Cycle Accurate Simulator chosen is appropriate for the desired target. For example, select the C6416 Device Cycle Accurate Simulator for a TMS320C6416 developer board. For more on the different categories of simulators, please refer to *Choosing the Appropriate Simulator Configuration in Code Composer Studio* (literature number SPRA864).
2. The ratio of the EMIF and CPU clocks on the simulator matches that on the respective developer board. See Section 4.4 for information on configuring the simulator with the correct EMIF to CPU clock ratio.
3. The EMIF settings of simulator match those of the developer board. Setting the EMIF registers with the same value across the simulator and developer board, either in the *emif_init* gel file or in the program, will ensure this.
4. Cycle differences can arise due to emulation breakpoint handling. To get more accurate numbers, use timers instead of profiler.