TMS320C6000
Instruction Set Simulator

User's Guide

Literature Number: SPRU546
September 2001

IMPORTANT NOTICE

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

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

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

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

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

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

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

This document is the user's guide for the TMS320C6000™ instruction set simulator, available within the Code Composer Studio IDE. This document describes the basic capabilities of the simulator and the features provided for configuring the it.

How to Use This Manual

This document contains the following chapters:

Chapter One discusses how to invoke the TMS320C6000 simulator, including choosing a simulation driver and configuration files. This chapter also describes the drivers supported by the C6000 simulator.

Chapter Two covers the devices and configurations supported by the C6000 simulator.

Chapter Three contains information on simulator features, simulation accuracy, and the known limitations.

The appendices contain such topics as the XBAR configuration file format, the pin connect file format, and how to optimize using the simulator.

Notational Conventions

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's. Examples use a **`bold version`** of the special typeface for emphasis; interactive displays use a **`bold version`** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

  Here is a sample program listing:

```
0011  0005  0001         .field    1, 2
0012  0005  0003         .field    3, 4
0013  0005  0006         .field    6, 3
0014  0006               .even
```

Here is an example of a system prompt and a command that you might enter:

```
C:  csr –a /user/ti/simuboard/utilities
```

- In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. 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. Here is an example of a directive syntax:

  .sect **"***section name***"**

  .sect is the directive. This directive has one parameter, indicated by *section name*. When you use .sect, the first parameter must be a section name, enclosed in double quotes.

- Square brackets (**[** and **]**) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you don't enter the brackets themselves. Here's an example of an instruction that has an optional parameter:

  ADD *[src,] dst*

  The ADD instruction has two parameters. The first parameter, *src*, is optional. The second parameter, *dst*, is required. As this syntax shows, if you use the optional first parameter, you must add a comma before the second parameter.

- Braces ({ and }) indicate a list. The symbol **|** (read as *or*) separates items within the list. Here's an example of a list:

  ```
  { * | *+ | *– }
  ```

  This provides three choices: `*`, `*+`, or `*–`.

  Unless the list is enclosed in square brackets, you must choose one item from the list.

- Some directives can have a varying number of parameters. For example, the .byte directive can have up to 100 parameters. The syntax for this directive is:

  .byte $value_1$ *[, ... ,* $value_n$*]*

  This syntax shows that .byte must have at least one value parameter, but you have the option of supplying additional value parameters, separated by commas.

Related Documentation From Texas Instruments

The following books describe the TMS320C6x devices and related support tools.

Code Composer User's Guide (literature number SPRU328) explains how to use the Code Composer development environment to build and debug embedded real-time DSP applications.

Trademarks

TMS320C6x and Code Composer Studio are trademarks of Texas Instruments.

Contents

Figures

Tables

Chapter 1

How To Invoke the TMS320C6000 Simulator

This chapter describes how to set up the TMS320C6000™ Simulator, including choosing a driver and configuration file, and specifying endianness.

Topic **Page**

1.1 Choosing a Simulator Driver

Code Composer Studio™ Setup must be used to specify the simulator you will use. This information is called the system configuration and it consists of a device driver that handles communication with the target plus other information and files that describe the characteristics of your target. Without this information, the Code Composer Studio IDE cannot establish communication with your target system and cannot determine which tools to use to build your application. For example, the compiler cannot know what special instructions are present, the linker cannot know how much memory is present, and the DSP/BIOS™ kernel cannot know what peripherals are present.

Each standard and custom system configuration is stored in a specially formatted file. The current system configuration is recorded in the System Registry where it can be retrieved by both Code Composer Studio IDE and Setup. When CCS Setup™ records a new configuration in the System Registry, it replaces the previous one. If you will be using more than one target in your project (for example, if you will be switching between a simulator and a target board) then you will need a separate configuration for each target and you will have to use Setup to switch between them when necessary.

You normally use the Setup utility to select (import) a standard configuration and record (save) it into the System Registry. If no standard configuration is quite right, you may be able to modify one. You can also create a custom configuration from scratch. Third parties who provide target boards or emulators may provide predefined configurations or instructions on creating appropriate configurations. All configurations must include a suitable device driver; creating device drivers is not within the scope of Code Composer Studio IDE or Setup.

1.1.1 Using Setup

When the Setup utility is invoked, the current configuration is displayed and the Import Configuration dialog box appears. This dialog lists predefined configurations that you can select. If you do not want to select a new configuration now, click Close on the Import Configuration dialog box and you will see the current configuration.

The displayed configuration is called the working configuration. Changes made to it do not become permanent until it is saved as the current configuration or until it is exported to a configuration file.

- To clear the working configuration, select Clear from the File menu.
- To import a configuration file, select Import from the File menu.

- To save the working configuration as the current configuration, select Save from the File menu.
- To export the working configuration to a file, select Export from the File menu.

To create a custom configuration, follow the steps in Custom Setup. You must perform a custom setup if you do not have a predefined configuration to import. When you create a custom configuration you can export it to a file and import it later.

The Code Composer Studio Setup interface is divided into three panes:

- System Configuration
- Available Board/Simulator Types (Available Processor Types)
- Setup Commands/Information

1.1.2 System Configuration

The C6000 Simulator components comprise the simulators for the following configurations:
C64xx™, C62xx™, C67xx™, C6201, C6202, C6203, C6204, C6205, C6211, C6711, C6701, C6414, C6415, C6416, and C64xc.

Choose the C6xxx Simulator as the board (in cc_setup.exe) if you wish to simulate any C6000 device. If a C6xxx Simulator board does not already exist, then click on the ”Install Device Driver” option and choose the file tisimC6xxx.dvr as the driver.

Also, simulator-specific gel files have been supplied with specific programming settings for the EMIF. The EMIF settings are intended to facilitate fast simulation.

In order to select the specific device simulator, select the corresponding Configuration (.cfg) file as shown in Table 1–1.

Table 1–1. Configuration Files and GEL Files

Configuration File	Device Selected	GEL File Supported
sim6201_simulator.cfg	6201	init6201sim.gel
sim6201_simulator_map0.cfg	6201	init6201_map0sim.gel
sim6202_simulator.cfg	6202	init6202sim.gel
sim6202_simulator_map0.cfg	6202	init6202_map0sim.gel
sim6203_simulator.cfg	6203	init6203sim.gel
sim6203_simulator_map0.cfg	6203	init6203_map0sim.gel
sim6204_simulator.cfg	6204	init6204sim.gel
sim6204_simulator_map0.cfg	6204	init6204_map0sim.gel
sim6205_simulator.cfg	6205	init6205sim.gel
sim6205_simulator_map0.cfg	6205	init6205_map0sim.gel
sim6701_simulator.cfg	6701	init6701sim.gel
sim6701_simulator_map0.cfg	6701	init6207_map0sim.gel
sim6211_simulator.cfg	6211	init6211sim.gel
sim6711_simulator.cfg	6711	init6711sim.gel
sim62xx_fast_simulator.cfg	62xx	init62xx_fastsim.gel
sim67xx_fast_simulator.cfg	67xx	init67xxsim.gel
sim64xx_fast_simulator.cfg	64xx	init64xxsim.gel
sim64xx_csim_simulator.cfg	64xx	init64xx_csim.gel
sim64xx_csim_simulator.cfg	64xc	init64xxsim.gel
sim6414_simulator.cfg	6414	init6414sim.gel
sim6415_simulator.cfg	6415	init6415sim.gel
sim6416_simulator.cfg	6416	init6416sim.gel

1.1.2.1 Simulator Feature Set

The following tables characterize the functionality of each of these simulators. Yes means the corresponding functionality is modeled, No means it is not, and na means it is not applicable.

Table 1–2. Simulator Feature Sets

	62xx	67xx	6201	6202	6203	6204	6205	6211	6711	6701
CPU Core	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Flat Memory	Yes	Yes	na	na	na	na	na	na	na	na
Internal Memory/ Cache Model	na	na	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DMA	na	na	Yes	Yes	Yes*	Yes*	Yes*	na	na	Yes
EDMA	na	na	na	na	na	na	na	Yes	Yes	No
EMIF	na	na	Yes	Yes	Yes	Yes	Yes	Yes+	Yes+	Yes
McBSP	na	na	Yes	Yes	Yes	Yes	Yes	No	No	Yes
Timer	na	na	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HPI/XBUS	na	na	no	No	No	No	No	No	No	No

Table 1–2. Simulator Features Sets (Continued)

	62xx	67xx	6201	6202	6203	6204	6205	6211	6711	6701
Supports Pin Connect	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Supports Port Connect	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TCP/VCP	na	na	na	na	na	na	na	na	na	na

+ - For the 6211/6711 simulators, the EMIF simulator comprises Async, SDRAM and SBSRAM memory models. Async and SBSRAM memory model simulator have been benchmarked to be around 85% cycle accurate. The SDRAM memory model simulator is functionally accurate.

* - The DMA functionality modeled on these devices is the same as that of the C6202 simulator. The enhancements have not been modeled.

Table 1–3. TMS320C64x Simulator Feature Sets

	C64xx (fastsim)	C64xx (cachesim)	C6414	C6415	C6416
CPU Core	Yes	Yes	Yes	Yes	Yes
Flat Memory	Yes	na	No	No	No
Internal Memory/ Cache Model	na	Yes	Yes	Yes	Yes
DMA	na	na	Yes	Yes	Yes
EDMA	na	na	Yes	Yes	Yes
EMIF	na	No	Yes	Yes	Yes
McBSP	na	na	Yes	Yes	Yes
Timer	No	No	Yes	Yes	Yes
HPI/XBUS	na	No	Yes	Yes	Yes
Supports Pin Connect	No	No	Yes	Yes	Yes
Supports Port Connect	No	No	Yes	Yes	Yes
TCP/VCP	na	na	No	No	Yes
Utopia	na	na	No	No	No
PCI	na	na	No	No	No

For the 64xx simulator, three variants are provided:

- ❑ CPU core only simulator - models the 64xx CPU
- ❑ CPU + two level cache simulator - models the CPU core along with the L1I, L1D and the L2 caches.
- ❑ CPU + two level cache simulator + EDMA + EMIF +TCP/VCP

1.1.3 Multiple Processor Configurations

The most common configurations include a single simulator or a single target board with a single CPU. However, for more complicated configurations, you may:

- Connect multiple emulators to your computer, each with its own target board
- Connect more than one target board to a single emulator, using special hardware to link the scan paths on the boards
- Have multiple CPUs on a single board, and the CPUs can be all of the same kind or they can be of different types (DSPs and microcontrollers, for example)

Although a Code Composer Studio configuration is represented as a series of boards, in fact each “board” is really either a single CPU simulator or a single emulator scan chain that could be attached to one or more boards with multiple processors. The device driver associated with the board must be able to comprehend all the CPUs on the scan chain.

In multiprocessor configurations, invoking Code Composer Studio IDE starts a special control known as the Parallel Debug Manager (PDM).

The PDM allows you to open a separate Code Composer Studio session for each target device. Activity on the specified devices can be controlled in parallel using the PDM control. For further details on the PDM, see the Code Composer Studio online help (Select Help -> Contents -> Using CCS IDE -> Parallel Debug Manager(PDM) -> Opening an Individual Parent Window). Although multiple devices are present, it is not necessary for you to use all of them. You can work within a single session.

1.1.4 Available Board/Simulator Types (Available Processor Types)

The title of this middle pane in the Setup display changes depending on the icon that is selected in the left System Configuration pane.

When the My System icon is selected in the System Configuration pane, the middle pane is titled **Available Board/Simulator Types** and lists all available target boards and simulators. Each of these entries represents the device driver for that target. All driver files that are shipped with the product are preinstalled in Code Composer Studio Setup and will appear in the pane.

When a target board or simulator is selected in the System Configuration pane, the middle pane is titled **Available Processor Types** and lists the processors that are available for that target.

You can drag-and-drop items from the Available Board/Simulator Types (Available Processor Types) pane to the System Configuration pane. The Set-

up utility prevents you from creating an unsupported processor or target board configuration.

After adding a target board or simulator you will have the opportunity to change the properties of that device. You can specify:

- board name
- board data file
- board properties
- processor configuration (for target boards that allow multiple CPUs)
- startup gel file(s)

1.1.5 Setup Commands/Information

The third (rightmost) pane provides information describing the target board or simulator that is highlighted in the Available Board/Simulator Types (Available Processor Types) pane.

The third pane also contains the following commands:

Import a Configuration File. Load a standard configuration file using the Import Configuration dialog. This is the same as selecting Import from the File menu.

Install a Device Driver. Install a Code Composer Studio device driver. The driver is displayed in the Available Board/Simulator Types pane.

Uninstall. Uninstall a Code Composer Studio device driver. The driver is removed from the Available Board/Simulator Types pane.

Add to System. Add a device driver to the system configuration using the Board Properties dialog.

The commands that are displayed depend on the items highlighted in the System Configuration and Available Board/Simulator Types (Available Processor Types) panes. For example, when a target board or simulator is selected in the Available Board/Simulator Types pane, the Uninstall and Add to System commands appear in the Setup Commands/Information pane.

1.2 Configuration Files

While configuring the Board Properties tab for a simulator, you may be asked to specify a simulator configuration file. Although the name is the same, a simulator configuration file is but a small piece of the larger system configuration of which the simulator is a part.

To Specify a Simulator Configuration File:

1) Click on the property Simulator Config File. In the Value field, a browse button ([..]) will be displayed.

2) Click on the browse button to open the Configuration File dialog box.

3) Select the appropriate simulator configuration file (.cfg).

4) Click on the Open button to accept your selection and close the Configuration File dialog box.

Please refer to Table 1–1 on page 1-4 for the list of simulator config files.

Notice that the prefix identifies the processor being simulated. Simulator configuration files with prefixes such as “sim6201” or “sim6205” simulate all the features of the specified processor. For the C64xx device, the sim64xx_lsim supports all the features of the processor. Simulator configuration files with prefixes such as “sim62xx” or “c55xx” simulate that processor families’ core. Features that are only available on a specific processor might not be available; however, these simulators usually run faster.

1.3 Specifying Endianness

Endianness refers to which bytes are most significant in multi-byte data types. In big-endian architectures, the leftmost bytes (those with a lower address) are most significant. In little-endian architectures, the rightmost bytes are most significant.

HP, IBM, Motorola 68000, and SPARC systems store multi-byte values in big-endian order, while Intel 80x86, DEC VAX, and DEC Alpha systems store them in little-endian order. Internet standard byte ordering is also big-endian. The TMS320C6000 is bi-endian because it supports both systems.

Chapter 2

Supported Devices and Configurations

This chapter describes the devices and configurations supported by the TMS320C6000™ simulator.

Topic **Page**

2.1 CPU

The C6000 simulator family supports the following CPU types: C62 (fixed point), C67 (floating point) and the C64 (fixed point with enhanced instruction set and register file). Please refer to the *TMS320C6000 CPU and Instruction Set Reference Guide* (literature number SPRU189) for information on the CPU architecture.

Table 2–1 lists the simulator configurations and the corresponding CPU types they support.

Table 2–1. Simulator Configurations and Corresponding CPU Types Supported

Configuration File	CPU Type
sim62xx_fast_simulator.cfg	C62
sim67xx_fast_simulator.cfg	C67
sim64xx_fast_simulator.cfg	C64
sim64xx_csim_simulator.cfg	C64
sim6414_simulator.cfg	C64
sim6415_simulator.cfg	C64
sim6416_simulator.cfg	C64
sim6201_simulator.cfg	C62
sim6202_simulator.cfg	C62
sim6203_simulator.cfg	C62
sim6204_simulator.cfg	C62
sim6205_simulator.cfg	C62
sim6201_simulator_map0.cfg	C62
sim6202_simulator_map0.cfg	C62
sim6203_simulator_map0.cfg	C62
sim6204_simulator_map0.cfg	C62
sim6205_simulator_map0.cfg	C62
sim6211_simulator.cfg	C62
sim6701_simulator.cfg	C67
sim6701_simulator_map0.cfg	C67
sim6711_simulator.cfg	C67

A brief description of the feature set of the CPU simulator is given below:

1. **Instruction Set**:

All the instructions described in the CPU Reference Guide (SPRU189) are supported. This includes support of all registers, addressing modes, branch conditions, parallel instructions, floating point instructions etc.

2. **Resource Conflict Detection**:

All the simulators report resource conflict errors if any appear in the running executable program. All documented resource constraints (such as functional unit access, register reads and writes, cross-path access, conditional register access, load/store data path, etc.) are verified during the course of the simulation.

Example 1:

In this example, the sample code fragment is:

MPY.M1 A1, A2, A3

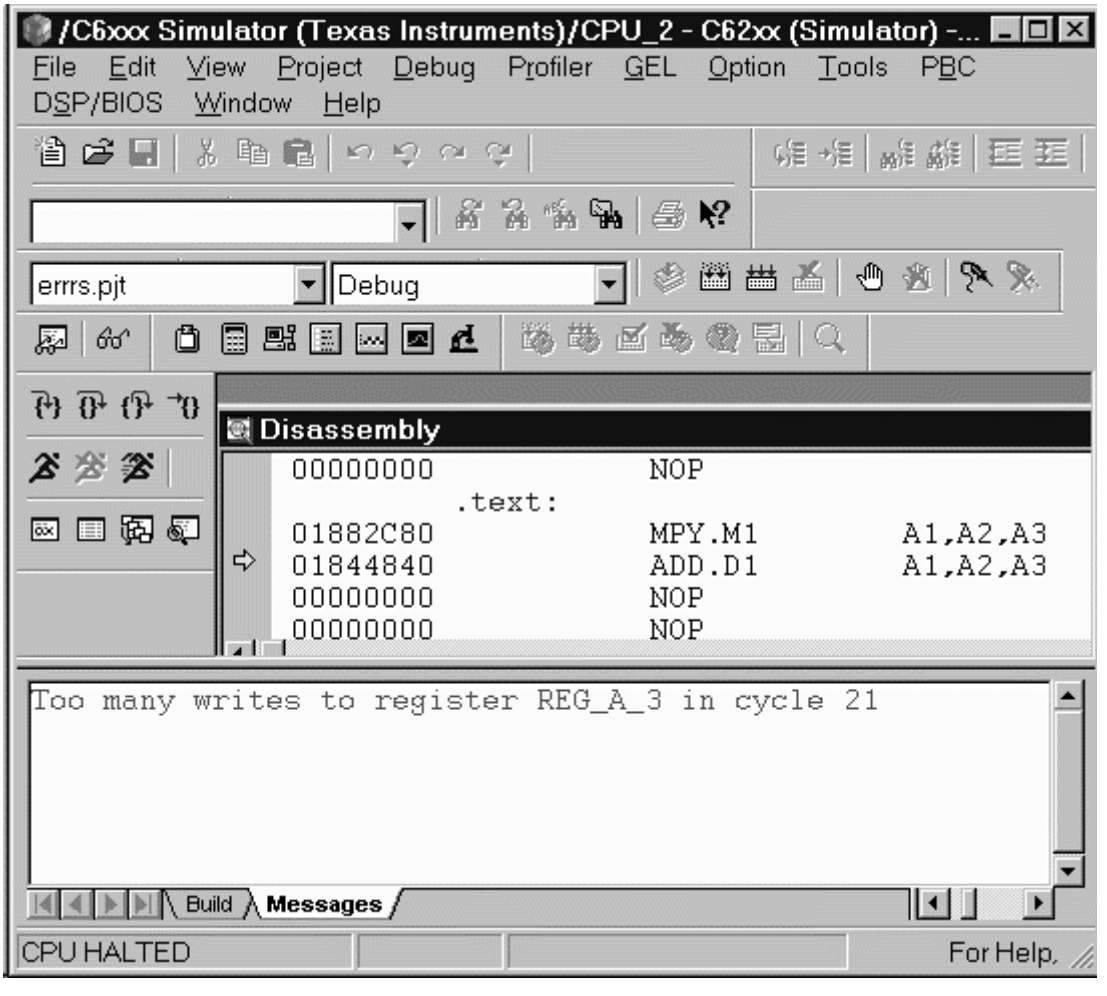
ADD.S1 A1, A2, A3

Clearly, there is a conflict in writing the results to A3 since MPY has a single cycle delay slot while the ADD has no delay slots.

When the above code fragment is run on the simulator (in any device configuration), the following error message is generated:

”Too Many Writes to Register REG_A_3 in cycle ...”

Figure 2–1. Register Conflict: Too Many Writes to a Register



Example 2:

In this example, the sample code fragment is:

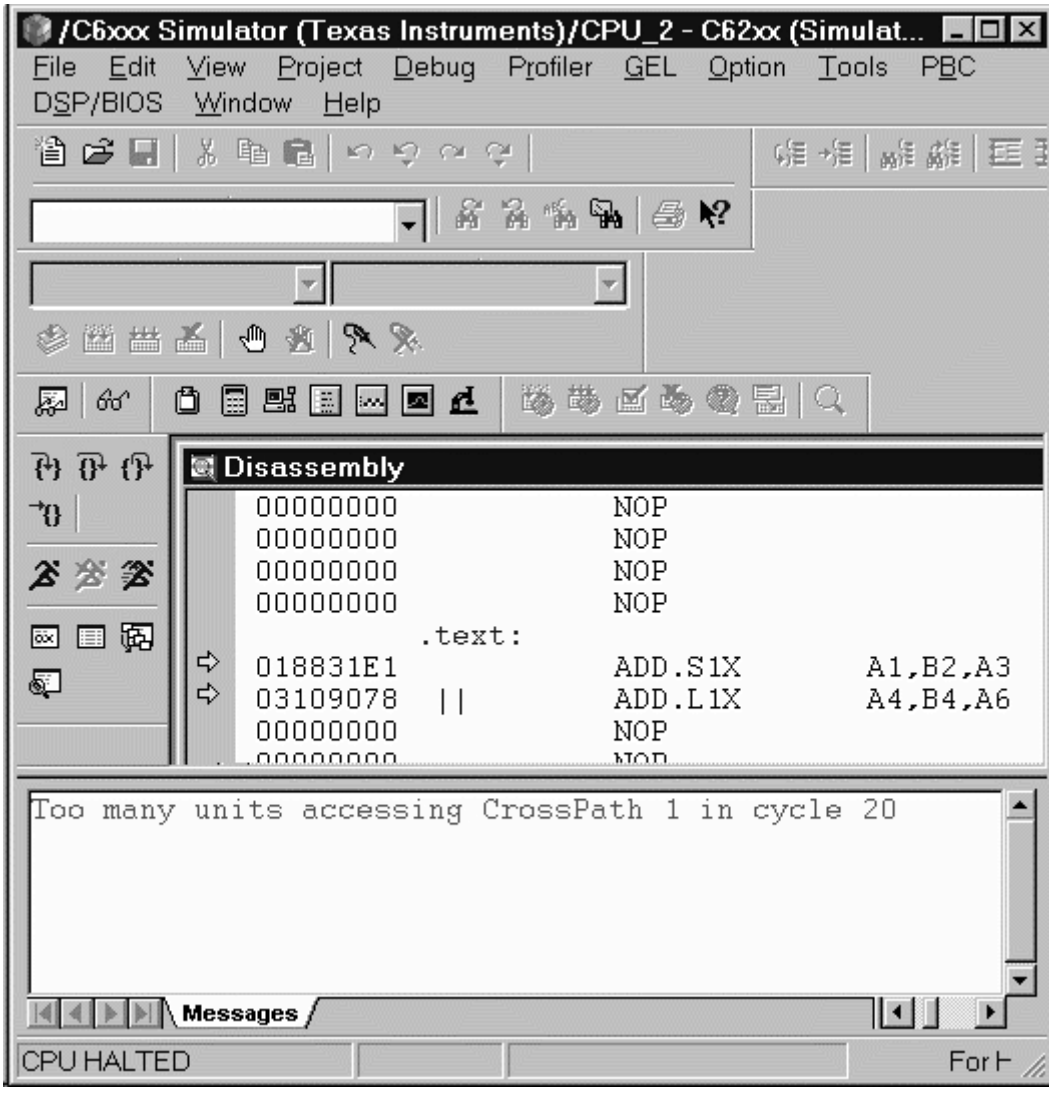
```
  ADD.S1X A1, B2, A3
|| ADD.L1X A4, B4, A6
```

Clearly, this has a cross-path conflict since both the instructions attempt to use the same cross path in the same cycle.

When the above code fragment is run on the simulator (in any device configuration), the following error message is generated:

”Too Many Units accessing cross path 1 in cycle ...”

Figure 2–2. Register Conflict: Cross Path Conflict



3. **Registers**:

Each simulator configuration supports all the user-visible registers (the data register sets A and B, and all the control registers). In case of control registers, the read-only bits are also modeled as such.

4. **Interrupts**:

All the documented interrupts - Reset, NMI and Intr4 to Intr 15 are supported. The interrupt behavior is modeled as documented in the *TMS320C6000 CPU and Instruction Set Reference Guide* (SPRU189).

5. **Accuracy**:

The CPU simulation is 100% cycle accurate for each of the supported CPU types. The overall device simulation accuracy is determined by the accuracy of the device model, comprising the cache model, external memory model, DMA and so on.

6. **Endianness**

All the simulator configurations support both endian modes: little endian and big endian.

2.2 Cache

This section describes the cache models supported in the device simulators. For detailed descriptions of the cache architecture, please refer to the *TMS320C6000 Peripherals Reference Guide* (SPRU190).

In general, all the cache models match the cache architecture specifications described in the C6000 Peripherals Reference Guide. They support standard cache behavior: LRU line replacement, direct mapping, set associativity, cache protocols for hit/miss service, snoops, victims etc. The allocation policies (allocate on read miss, write-back on write miss etc.) are also modeled to match the device specifications.

2.2.1 Notation

The term C620x refers to all the devices - C6201, C6202, C6203, C6204 and C6205.

The term C6x0x refers to all the devices - C6201, C6202, C6203, C6204, C6205 and C6701.

The terms L1P, L1D and L2 denote Level 1 program cache, Level 1 data cache and Level 2 unified memory respectively.

2.2.2 Notes

1) All the C6x0x simulators support arbitration between CPU and DMA for internal program memory (alternately cache) and internal data memory accesses. None of the C6x0x devices have any on-chip data cache.

2) All the simulators support both endian modes: little and big.

3) All the simulators detect and report data bank conflicts.The data bank conflicts are reported to the user by means of the Simulator Analysis Event mechanism. Please refer Section 3.3.3 for a detailed description of the Simulator Analysis Event mechanism. Briefly, the user may invoke this feature from the Code Composer Studio tools menu (available as C6000 Simulator Analysis). This opens up the simulator analysis window inside the main CCStudio application window. The user may right click the mouse inside the analysis window and select the item ”Analysis Setup ...” to choose one or more events to track. The data bank conflict event is available in the following simulator configurations:

Simulator Configuration	Event Name
sim6201_simulator.cfg	Data Bank Conflict
sim6201_simulator_map0.cfg	Data Bank Conflict
sim6202_simulator.cfg	Data Bank Conflict
sim6202_simulator_map0.cfg	Data Bank Conflict
sim6203_simulator.cfg	Data Bank Conflict
sim6203_simulator_map0.cfg	Data Bank Conflict
sim6204_simulator.cfg	Data Bank Conflict
sim6204_simulator_map0.cfg	Data Bank Conflict
sim6205_simulator.cfg	Data Bank Conflict
sim6205_simulator_map0.cfg	Data Bank Conflict
sim6211_simulator.cfg	(L1D) Data Access Conflict, L2 Bank Conflict
sim6711_simulator.cfg	(L1D) Data Access Conflict, L2 Bank Conflict
sim64xx_csim_simulator.cfg	L1D Data Bank Conflict
sim6414_simulator.cfg	L1D Data Bank Conflict
sim6415_simulator.cfg	L1D Data Bank Conflict
sim6416_simulator.cfg	L1D Data Bank Conflict

Example:

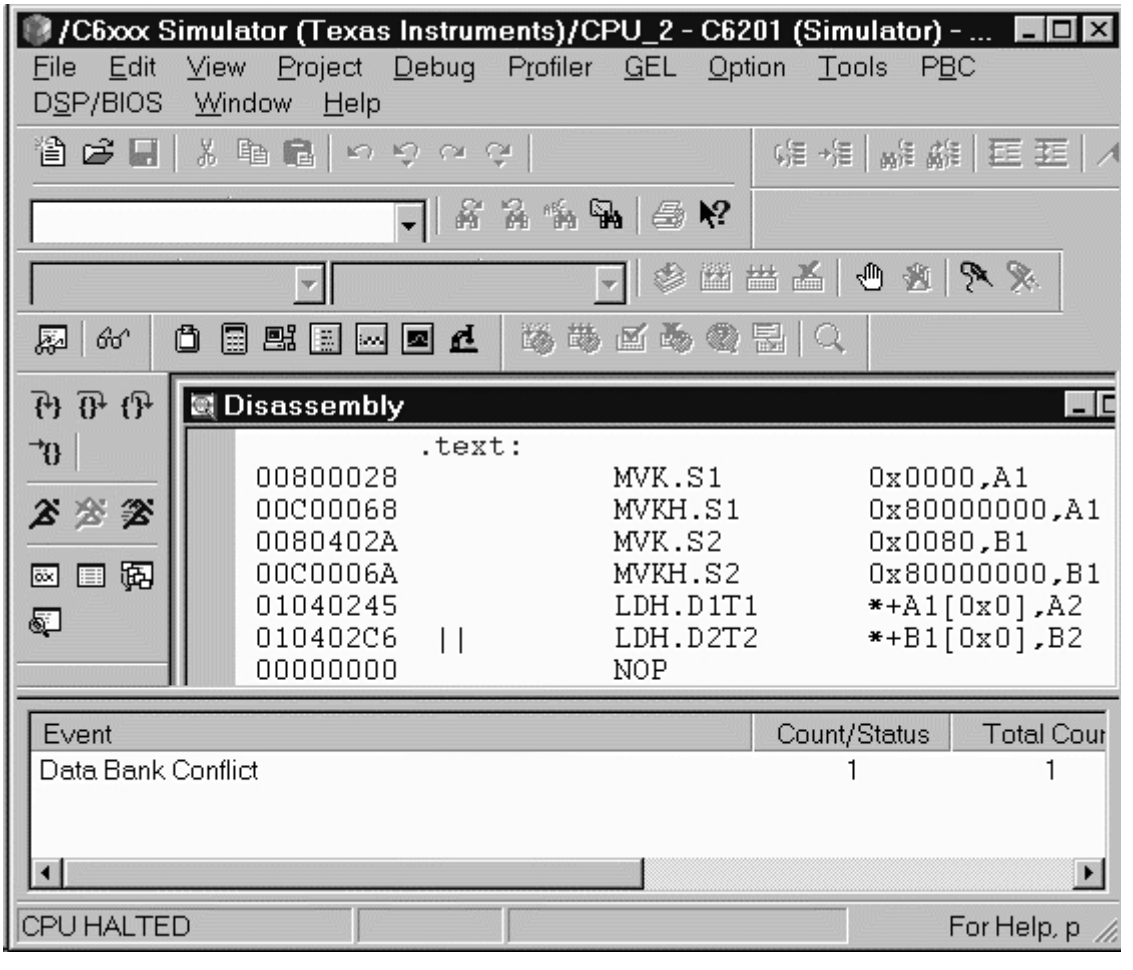
As an illustration, let us choose the sim6201_simulator configuration.

The following code fragment is run on this simulator after enabling the Data Bank Conflict Event for count.

```
  MVK.D1 0x80000000, A1

  MVKH.D1 0x80000000, A1

  MVK.D2 0x80000080, B1

  MVKH.D2 0x80000080, B1

  LDH.D1 *A1, A2

|| LDH.D2 *B1, B2
```

This causes a data bank conflict to occur since both the LDH instructions access the same bank in the same cycle, as seen in the following figure:

Figure 2–3. Data Bank Conflict

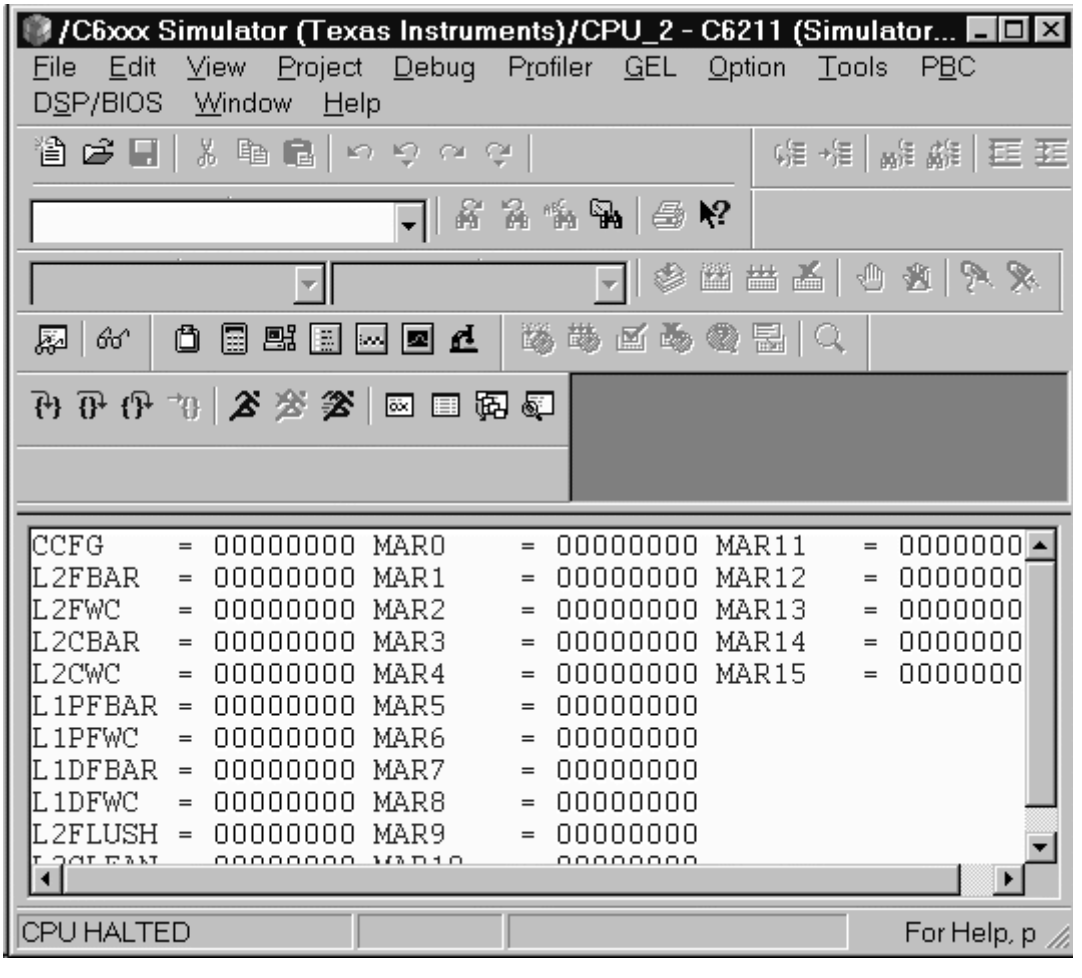


4) All user-visible control registers are supported, e.g. the L2CFG register in the 6211 simulator.

Example:

Choose the 6211 simulator configuration. Click View ->CPU Registers ->L2 Registers to bring up a register display window showing all the user programmable L2 registers, as seen in Figure 2–4.

Figure 2–4. User Programmable L2 Registers



5) This section describes the functional support of all the cache models.

For information related to the cycle accuracy of these models, see section 3.6, Simulation Accuracy, found on page 3-33 of this document.

The following table highlights the characteristics of cache models supported in the C6x0x device simulators:

Table 2–2. Cache Models Supported in the C6x0x Device Simulators

Simulator Configuration	Program Cache	Data Cache	Known Limitations
sim6201_simulator.cfg	64kBytes Mapped Cache Enable Cache Freeze Cache Bypass DMA Access	64kBytes Memory 2 32kByte Blocks Each block of 4 4k 16-bit Banks DMA Access	
sim6201_simulator_map0.cfg	64kBytes Mapped Cache Enable Cache Freeze Cache Bypass DMA Access	64kBytes Memory 2 32kByte Blocks Each block of 4 4k 16-bit Banks DMA Access	
sim6701_simulator.cfg	64kBytes Mapped Cache Enable Cache Freeze Cache Bypass DMA Access	64kBytes Memory 2 32kByte Blocks Each block of 8 2k 16-bit Banks DMA Access	
sim6701_simulator_map0.cfg	64kBytes Mapped Cache Enable Cache Freeze Cache Bypass DMA Access	64kBytes Memory 2 32kByte Blocks Each block of 8 2k 16-bit Banks DMA Access	
sim6202_simulator.cfg	128KBytes Mapped/Cache 128KBytes Mapped Cache Enable Cache Freeze Cache Bypass DMA Access	128kBytes Mapped 2 64kByte Blocks Each block of 4 8k 16-bit Banks DMA Access	

Table 2–2. Cache Models Supported in the C6x0x Device Simulators (Continued)

Simulator Configuration	Program Cache	Data Cache	Known Limitations
sim6202_simulator_map0.cfg	128KBytes Mapped 128KBytes Cache Cache Enable Cache Freeze Cache Bypass DMA Access	128KBytes Mapped 2 64KByte Blocks Each block of 4 8k 16-bit Banks DMA Access	
sim6203_simulator.cfg	256KBytes Mapped 128KBytes Cache Cache Enable Cache Freeze Cache Bypass DMA Access	512KBytes Mapped 2 64KByte Blocks Each block of 4 8k 16-bit Banks DMA Access	
sim6203_simulator_map0.cfg	256KBytes Mapped 128KBytes Cache Cache Enable Cache Freeze Cache Bypass DMA Access	512KBytes Mapped 2 64KByte Blocks Each block of 4 8k 16-bit Banks DMA Access	
sim6204_simulator.cfg	64KBytes Mapped 128KBytes Cache Cache Enable Cache Freeze Cache Bypass DMA Access	64KBytes Mapped 2 64KByte Blocks Each block of 4 8k 16-bit Banks DMA Access	
sim6204_simulator_map0.cfg	64KBytes Mapped 128KBytes Cache Cache Enable Cache Freeze Cache Bypass DMA Access	64KBytes Mapped 2 64KByte Blocks Each block of 4 8k 16-bit Banks DMA Access	

Table 2–2. Cache Models Supported in the C6x0x Device Simulators (Continued)

Simulator Configuration	Program Cache	Data Cache	Known Limitations
sim6205_simulator.cfg	64KBytes Mapped 128KBytes Cache Cache Enable Cache Freeze Cache Bypass DMA Access	64KBytes Mapped 2 64KByte Blocks Each block of 4 8k 16-bit Banks DMA Access	
sim6205_simulator_map0.cfg	64KBytes Mapped 128KBytes Cache Cache Enable Cache Freeze Cache Bypass DMA Access	64KBytes Mapped 2 64KByte Blocks Each block of 4 8k 16-bit Banks DMA Access	

The following table summarizes the characteristics of the cache models supported in the C6211, C6711 and C64xx device simulators:

Table 2–3. Cache Models Supported in C6211,6711, and C64xx Device Simulators

Simulator Configuration	L1P	L1D	L2	Known Limitations
sim6211_simulator.cfg	4kBytes Direct Map 64Bytes line size DMA Access	4kBytes 2-way 32Bytes line size DMA Access	64kBytes 1-4 way 128Bytes line size Mapped DMA Access	
sim6711_simulator.cfg	4kBytes Direct Map 64Bytes line size DMA Access	4kBytes 2-way 32Bytes line size DMA Access	64kBytes 1-4 way 128Bytes line size DMA Access	
sim64xx_csim_simulator.cfg	16kBytes Direct Map 32Bytes line size DMA Access	16kBytes 2-way 64Bytes line size DMA Access	0k-4MB up to 256KBytes cache (in sizes of 32kB, 64kB, 128kB, or 256kB) 4-way 128Bytes line size DMA Access	This simulator does not support Cache Control Operations registers (XXBAR and XXWC registers).
sim6414_simulator.cfg	16kBytes Direct Map 32Bytes line size DMA Access	16kBytes 2-way 64Bytes line size DMA Access	0k-4MB up to 256KBytes cache (in sizes of 32kB, 64kB, 128kB, or 256kB) 4-way 128Bytes line size DMA Access	

Simulator Configuration	L1P	L1D	L2	Known Limitations
sim6415_simulator.cfg	16kBytes Direct Map 32Bytes line size DMA Access	16kBytes 2-way 64Bytes line size DMA Access	0k-4MB up to 256KBytes cache (in sizes of 32kB, 64kB, 128kB, or 256kB) 4-way 128Bytes line size DMA Access	
sim6416_simulator.cfg	16kBytes Direct Map 32Bytes line size DMA Access	16kBytes 2-way 64Bytes line size DMA Access	0k-4MB up to 256KBytes cache (in sizes of 32kB, 64kB, 128kB, or 256kB) 4-way 128Bytes line size DMA Access	

2.3 DMA

This section describes about the DMA (Direct Memory Access) models supported in C6x device simulators. For details regarding DMA architecture and programming please refer the *TMS320C6000 Peripherals Reference Guide* (literature number SPRU190).

2.3.1 C6x0x DMA

In C6x0x device simulators, the DMA model is designed to match hardware specifications The C6x0x DMA supports all features - e.g. Four channels, resource arbitration and priority configuration, split channel operation, and DMA access to config space. For details regarding DMA architecture and programming refer to chapter 4 in the *TMS320C6000 Peripheral Reference Guide* (literature noumber SPRU190). Here are a few important notes regarding the DMA model for the 6x0x category:

- C6201/C6701/C6202 have a common FIFO to hold data from high performance source (e.g. internal memory) which is shared among all four DMA channels. In C6203/C6204/C6205 the FIFO is meant to be dedicated separately for each channel, but this feature is not supported in the device simulators.
- The auxilary (5th) channel to service HPI requests is not modelled.
- The sync events SD_INT (EMIF SDRAM timer interrupt event 3) and DSPINT (Host Processo-to-DSP interrupt event 16) are not modelled.
- The sync events are captured only if they are enabled. If an event comes when it is disabled it will not be captured, in order that the transfer corresponding to it could be triggerd in the future by enabling that event.

2.3.2 C6x1x DMA

The DMA in C6x1x devices is enhanced (known as EDMA) with many new features such as 16 channels, programmable priority and the ability to link and chain data transfers. Regarding details of the structure and programming, C6x1x EDMA refers to chapter six of the *TMS320C6000 Peripheral Reference Guide* (literature number SPRU190). Here are a few important notes on C6x1x EDMA:

- The QDMA (quick DMA module) for triggering simple fast DMAs is not modelled in C6x1x device simulators.
- The sync events SD_INT (EMIF SDRAM timer interrupt event 3) and DSPINT (Host Processor to DSP interrupt event 0) are not modelled.

2.3.3 C64x EDMA

The C64x EDMA is much more enhanced, with features such as support for 64 channels, transfers on all channels chainable/linkable, and the ability to submit transfers on all four priorities. For details of structure and programming of C64x devices EDMA refer to chapter six of the *TMS320C6000 Peripheral Reference Guide* (literature number SPRU190). Here are a few major points about EDMA in C64x device simulators:

- The event polarity registers are not modelled. All the events are taken only in active-high status.
- The Push Data Transfer feature is not modelled. Therefore PDTD and PDTS (bit numbers two and three, respectively, of the option field) are ignored.
- The sync events DSPINT (Host-to-DSP interrupt event 0), GPIOINT (16 events 4-11 and 48-55), SD_INTA (EMIFA SDRAM timer interrupt event 3), SD_INTB (EMIFB SDRAM timer interrupt event 20), PCI event (PCI wakeup interrupt event 21), UREVT (Utopia Receive event 32) and UXEVT (Utopia Transmit event 40) are not modelled.

Chapter 3

Simulator Features

This chapter describes the simulation features of the TMS320C6000™ Simulator, as well as the simulation accuracy and known limitations.

Topic **Page**

3.1 Port Connect

It might be required that during the execution of certain applications, the content of some memory locations have to change and the change is external to the execution of the application, i.e. the application is not making those changes. For example in TMS320C6x0x devices, the Data Transmit Register (DXR) of the serial ports may be connected to a file to which it dumps the data to be transmitted. Similarly the Data Receive Register (DRR) may receive a series of data from a file as though the inputs were obtained by sampling the serial port receive pins.

The Port Connect feature in the simulator enables performing this operation. A file can be connected to a memory address (port) and data can be read from/written to the file.You can use the Port Connect tool to access a file through a memory address. Then, by connecting to the memory (port) address, you can read data in from a file and/or write data out to a file.

3.1.1 Available Memory Ranges

For the TMS320C6000 devices, the memory range to which a file can be connected for reading/writing are given in Table 3–1 on page 3-2.

Table 3–1. Available Memory Ranges

Configuration File	Available Memory Range
c6201_ltl_endian_sim_map1.ccs, c6201_big_endian_sim_map1.ccs	0x0040 0000 – 0x017F FFFF
c6201_ltl_endian_sim_map0.ccs, c6201_big_endian_sim_map0.ccs	0x0000 0000 – 0x013F FFFF
c6202_ltl_endian_sim_map1.ccs, c6202_big_endian_sim_map1.ccs	0x0040 0000 – 0x017F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2
c6202_ltl_endian_sim_map0.ccs, c6202_big_endian_sim_map0.ccs	0x0000 0000 – 0x013F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2
c6203_ltl_endian_sim_map1.ccs, c6203_big_endian_sim_map1.ccs	0x0040 0000 – 0x017F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2
c6203_ltl_endian_sim_map0.ccs, c6203_big_endian_sim_map0.ccs	0x0000 0000 – 0x013F FFFF DXR0, DRR0, DXR1, DRR1, DXR2, DRR2
c6204_ltl_endian_sim_map1.ccs, c6204_big_endian_sim_map1.ccs	0x0040 0000 – 0x017F FFFF DXR0, DRR0, DXR1, DRR1
c6204_ltl_endian_sim_map0.ccs, c6204_big_endian_sim_map0.ccs	0x0000 0000 – 0x013F FFFF DXR0, DRR0, DXR1, DRR1
c6205_ltl_endian_sim_map1.ccs, c6205_big_endian_sim_map1.ccs	0x0040 0000 – 0x017F FFFF DXR0, DRR0, DXR1, DRR1

Table 3–1. Available Memory Ranges(Continued)

Configuration File	Available Memory Range
c6205_ltl_endian_sim_map0.ccs, c6205_big_endian_sim_map0.ccs	0x0000 0000 – 0x013F FFFF DXR0, DRR0, DXR1, DRR1
c6701_ltl_endian_sim_map1.ccs, c6701_big_endian_sim_map1.ccs	0x0040 0000 – 0x017F FFFF DXR0, DRR0, DXR1, DRR1
c6701_ltl_endian_sim_map0.ccs, c6701_big_endian_sim_map0.ccs	0x0000 0000 – 0x013F FFFF DXR0, DRR0, DXR1, DRR1
c6211_ltl_endian_sim.ccs, c6211_big_endian_sim.ccs	0x8000 0000 – 0xBFFF FFFF
c6711_ltl_endian_sim.ccs, c6711_big_endian_sim.ccs	0x8000 0000 – 0xBFFF FFFF
c62xx_Ltl_Endian_Sim.ccs, c62xx_Big_Endian_Sim.ccs	No port connect available
c67xx_Ltl_Endian_Sim.ccs, c67xx_Big_Endian_Sim.ccs	No port connect available
c64xx_ltl_endian_sim.ccs, c64xx_big_endian_sim.ccs	0x6000 0000 – 0x6FFF FFFF 0x8000 0000 – 0xBFFF FFFF
c64xx_cachesim_ltl_endian_sim.ccs, c64xx_cachesim_big_endian_sim.ccs	0x6000 0000 – 0x6FFF FFFF 0x8000 0000 – 0xBFFF FFFF
c6414_ltl_endian_sim.ccs, c6414_big_endian_sim.ccs	0x6000 0000 – 0x6FFF FFFF 0x8000 0000 – 0xBFFF FFFF
c6415_ltl_endian_sim.ccs, c6415_big_endian_sim.ccs	0x6000 0000 – 0x6FFF FFFF 0x8000 0000 – 0xBFFF FFFF
c6416_ltl_endian_sim.ccs, c6416_big_endian_sim.ccs	0x6000 0000 – 0x6FFF FFFF 0x8000 0000 – 0xBFFF FFFF

Note: DXR(i) - Data Transmit Register of Serial Port i
DRR(i) - Data Receive Register of Serial Port i

3.1.2 Connecting a File

To connect a memory (port) address to a data file, follow these steps:

1) From the Tools menu, select Port Connect. This action displays the Port Connect window and starts the Port Connect tool.

2) Click the Connect button. This action opens the Connect dialog box.

3) In the Port Address field, enter the memory address. This parameter can be an absolute address, any C expression, the name of a C function, or an assembly language label. If you want to specify a hex address, be sure to prefix the address number with 0x; otherwise, Code Composer Studio IDE treats the number as a decimal address.

4) In the Length field, enter the length of the memory range. The length can be any C expression.

5) In the Page field, choose type of memory (program or data) that the address occupies:

To identify this page...	Use this value as the page parameter...
Program Memory	Prog
Data Memory	Data

6) In the Type field, select the Write or Read radio button.

7) Click OK. This action displays the Open Port File window.

8) Select the data file to which you want to connect and click Open.

The file is accessed during an assembly language read or write of the associated memory address. Any memory address can be connected to a file. A maximum of one input and one output file can be connected to a single memory address; multiple addresses can be connected to a single file.

3.1.3 Disconnecting a File

To remove some address from the memory map, the file that was connected to that memory address must be disconnected. To disconnect a memory address from a data file, follow these steps:

1) Select the line that contains the file and port address you want to disconnect. This action activates the Disconnect button.

2) Press the Disconnect button or double click the port address. This action disconnects the memory address from the data file and removes that information from the Port Connect window.

3.1.4 File Format for the Port Connect

The file that is connected to some memory address through Port Connect must have data in hexidecimal format with a 0x preceding the data. For example, the sample data file connected to a memory address may look like:

0x00000004
0x000000a6
0x00002fea

The above file will transfer three 32-bit data to the memory address connected to it.

NOTE: In the case of C64xx simulators, the port connect feature works for memory regions which have been programmed as cacheable.

3.2 Pin Connect

Pin Connect enables 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 having a specific format (please refer to appendix B) can be connected to those pins. The pins being simulated are of two types - Pulse and Waveform. The pins simulated as Pulse pins are sensitive to rising edges and the pins simulated as Waveform pins are sensible to both the edges. For example, CPU interrupt pins are sensitive to rising edges and the clock input pins of the serial port (CLKX , CLKR) of the TMS320C6000 devices are sensitive to both the edges.

3.2.1 Available Pins for Configuration of TMS320C6000 Devices

The available pins for the configurations of TMS320C6000 devices are :

(a) TMS320C6201, TMS320C6204, TMS320C6205 and TMS320C6701

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

(b) TMS320C6202 and TMS320C6203

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
FSX2	Transmit frame synchronization pin for McBSP2	Waveform
FSR2	Receive frame synchronization pin for McBSP2	Waveform
CLKX2	Transmit clock pin for McBSP2	Waveform
CLKR2	Receive clock pin for McBSP2	Waveform
CLKS2	External clock input for McBSP2	Waveform

(c) TMS320C621x, TMS320C671x and TMS320C64xx

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

TMS320C62x Fast Simulator Pin Connect not available

TMS320C67x Fast Simulator Pin Connect not available

3.2.2 Simulating External Interrupts

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:

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

3.2.3 Connecting a Data File

After creating your data file, you must connect the file to an external interrupt pin.

To Connect a Data File to an External Interrupt Pin

1) Select Tools®Pin Connect from the Code Composer Studio menu bar. The Pin Connect tool appears. The available external interrupt pins are listed in the Pin Name column.
2) Click on the Pin Name that you wish to use. Notice that the Filename column displays the status ”Not Connected”.

3) Click the Connect button (or double-click on the selected Pin Name). The Open Pin File dialog box appears.

4) Specify the name of your data file in the File name field.

5) Click the Open button.

The selected pin is connected to your data file. Notice that the selected data file is now listed in the Filename column.

3.2.4 Creating a Data 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 occurs at the clock cycle as specified in the file.

For details about the possible formats of the pin connect file please refer to Appendix B.

3.2.5 Disconnecting a Data File from an Interrupt

To end the interrupt simulation:

1) Click on the pin name that you wish to disconnect.

2) Click on the Disconnect button or double-click on the Pin name.

The pin is disconnected from your data file. Notice that the Filename column displays the status ”Not Connected”.

3.3 Analysis Events

TMS320C6000 Simulator Analysis allows you to setup and monitor the occurrence of events. Simulated events can include program cache miss, program cache hit, program fetch, program access block 0, program access block 1, and memory stall.

The C6000 Simulator Analysis tool reports the occurrence of particular system events so you can accurately monitor and measure the performance of your program**.**

To Use the C6000 Simulator Analysis Tool:

1) Load your program using the Code Composer Studio IDE.
2) Start the analysis tool. Select Tools and then C6000 Simulator Analysis from the Code Composer Studio menu.
3) Enable analysis.
4) Specify your analysis parameters (count events or halt events).
5) Use the Code Composer Studio IDE to run or step through your program.
6) Analyze the output of the analysis tool using the analysis window.

3.3.1 Events Supported

TMS320C6201/C6701 events

The TMS320C6201 and TMS320C6701 simulators support the following system events:

Table 3–2. TMS320C6201 and C6701 Simulator System Events

Event Name	Event Action
Data Access Block 0	Data read Block 0.
Data Bank Conflict	Two data accesses occur in the same memory bank resulting in a CPU stall.
External Data Access	A data memory access resulting in an external memory access.
External Program Access	A program memory access resulting in an external memory access. A single program fetch from external memory results in eight off-chip memory accesses.
Memory Stall	The number of cycles lost due to memory stalls. A memory stall occurs when the CPU is stalled because the system is unable to provide the data as soon as it is requested.
Program Access Block 0	Program access block 0.
Program Cache Hit	Any program memory access resulting in a cache hit. Occurs only when cache is enabled.
Program Cache Miss	Any program memory access resulting in a cache miss. Occurs only when cache is enabled.
Program Fetch	Any program fetch.

TMS320C6202 events

The TMS320C6202 simulator supports the following system events:

Table 3–3. TMS320C6202 Simulator System Events

Event Name	Event Action
Data Access Block 0	Data read Block 0.
Data Access Block 1	Data read Block 1.
Data Bank Conflict	Two data accesses occur in the same memory bank resulting in a CPU stall.
External Data Access	A data memory access resulting in an external memory access.
External Program Access	A program memory access resulting in an external memory access. A single program fetch from external memory results in eight off-chip memory accesses.
Memory Stall	The number of cycles lost due to memory stalls. A memory stall occurs when the CPU is stalled because the system is unable to provide the data as soon as it is requested.
Program Access Block 0	Program access block 0.
Program Access Block 1	Program access block 1.
Program Cache Hit	Any program memory access resulting in a cache hit. Occurs only when cache is enabled.
Program Cache Miss	Any program memory access resulting in a cache miss. Occurs only when cache is enabled.
Program Fetch	Any program fetch.

TMS320C6211/C6711events

The TMS320C6211/C6711 simulator supports the following system events:

Table 3–4. TMS320C6211/C6711 Simulator System Events

(a) Data Events

Event Name	Event Action
Data Access Conflict	Parallel requests from the CPU trying to access the same line.
Data Memory Access	A CPU access to L1D (Read/ Write). Maximum of 2 per cycle. = Data Read Events + Data Write Events.
Data Read	CPU Reads. Maximum of 2 per cycle. = Data Read Hit Events + Data Read Miss Events.
Data Read Hit	CPU Reads that resulted in a Cache hit. Maximum of 2 per cycle.
Data Read Miss	CPU Reads that resulted in a Cache miss. Maximum of 2 per cycle.
Data Write	CPU Writes. Maximum of 2 per cycle. = Data Write Hit Events + Data Write Miss Events.
Data Write Hit	CPU Writes that resulted in a Cache hit. Maximum of 2 per cycle.
Data Write Miss	CPU Writes that resulted in a Cache miss. Maximum of 2 per cycle.

(b) L1 Cache Events

Event Name	Event Action
L1 Cache Bypass Read Misses	This event counts the sum total of L1D Cache Bypass Read Misses and L1I Cache Bypass Read Misses.
L1 Cache Bypass Snoop Hits	This event counts the sum total of L1D Cache Bypass Snoop Hits and L1I Cache Bypass Snoop Hits.
L1 Cache Bypass Snoop Misses	This event counts the sum total of L1D Cache Bypass Snoop Misses and L1I Cache Bypass Snoop Misses.
L1 Cache Freeze Read Hits	This event counts the sum total of L1D Cache Freeze Read Hits and L1I Cache Freeze Read Hits.
L1 Cache Freeze Read Misses	This event counts the sum total of L1D Cache Freeze Read Misses and L1I Cache Freeze Read Misses.
L1 Cache Freeze Snoop Hits	This event counts the sum total of L1D Cache Freeze Snoop Hits and L1I Cache Freeze Snoop Hits.
L1 Cache Freeze Snoop Misses	This event counts the sum total of L1D Cache Freeze Snoop Misses and L1I Cache Freeze Snoop Misses.
L1 Cache State Freeze Read Hits	This event counts the sum total of L1D Cache State Freeze Read Hits and L1I Cache State Freeze Read Hits.
L1 Cache State Freeze Read Misses	This event counts the sum total of L1D Cache State Freeze Read Misses and L1I Cache State Freeze Read Misses.
L1 Cache State Freeze Snoop Hits	This event counts the sum total of L1D Cache State Freeze Snoop Hits and L1I Cache State Freeze Snoop Hits.
L1 Cache State Freeze Snoop Misses	This event counts the sum total of L1D Cache State Freeze Snoop Misses and L1I Cache State Freeze Snoop Misses.

(c) L1D Events

Event Name	Event Action
L1D Cache Bypass Read Misses	While L1D is in Cache Bypass Mode, this event counts the number of read misses.
L1D Cache Bypass Write Misses	While L1D is in Cache Bypass Mode, this event counts the number of write accesses.
L1D Cache Bypass Snoop Hits	While L1D is in bypass mode, this event counts the number of hits for snoop requests from L2.
L1D Cache Bypass Snoop Misses	While L1D is in bypass mode, this event counts the number of misses for snoop requests from L2.
L1D Cache Freeze Read Misses	While L1D is in Cache Freeze Mode, this event counts the number of read misses.
L1D Cache Freeze Write Misses	While L1D is in Cache Freeze Mode, this event counts the number of write misses.
L1D Cache Freeze Snoop Hits	While L1D is in Cache Freeze Mode, this event counts the number of snoop requests that successfully hit an L1D cache line.
L1D Cache Freeze Snoop Misses	While L1D is in Cache Freeze Mode, this event counts the number of snoop requests that failed to hit an L1D cache line.
L1D Cache State Freeze Read Hits	While L1D is in Cache State Freeze Mode, this event counts the number of read hits.
L1D Cache State Freeze Read Misses	While L1D is in Cache State Freeze Mode, this event counts the number of read misses.
L1D Cache State Freeze Write Hits	While L1D is in Cache State Freeze Mode, this event counts the number of write hits.
L1D Cache State Freeze Write Misses	While L1D is in Cache State Freeze Mode, this event counts the number of write misses.
L1D Cache State Freeze Snoop Hits	While L1D is in Cache State Freeze Mode, this event counts the number of snoop requests from L2 that successfully hit an L1D cache line.
L1D Cache State Freeze Snoop Misses	While L1D is in Cache State Freeze Mode, this event counts the number of snoop requests from L2 that failed to hit an L1D cache line (that is, the snoop request did not match any L1D line).

Event Name	Event Action
L1D Dirty Line Replace	Line replaces as above that resulted in lines that have been modified in L1D. Such lines need to be written out to L2 before replacement (Victim lines).
L1D Line Replace	Cache accesses in L1D that resulted in a line being replaced in L1D.
L1D Write Buffer Full	CPU had to stall because of the write buffer being full. The write buffer in L1D is 4 deep. If the number of pending writes exceeds this, the CPU stalls until at least one slot is available for the new write to occur.

(d) L1I Events

Event Name	Event Action
L1I Line Flush	Snoop requests from L2 result in the line being flushed from L1I.
L1I Line Replace	Program accesses in L1I that resulted in a line being replaced in L1I.

(e) L2 Bank Event

Event Name	Event Action
L2 Bank Conflicts	L2 data Banks do not allow more than 1 read/write at any time. When different requests need to access the same bank, this results in a bank conflict and one of the requests stalls.

(f) L2 Cache Events

Event Name	Event Action
L2 Cache Bypass Reads	While L2 is in Cache Bypass Mode, this event counts the number of reads from L2.
L2 Cache Bypass Writes	While L2 is in Cache Bypass Mode, this event counts the number of writes to L2.
L2 Cache Freeze Read Hits	While L2 is in Cache Freeze Mode, this event counts the number of reads from L2 which are hits.
L2 Cache Freeze Reads	While L2 is in Cache Freeze Mode, this event counts the total number of reads from L2 (that is, the sum total of read hits in L2 and read misses in L2).
L2 Cache Freeze Write Hits	While L2 is in Cache Freeze Mode, this event counts the number of writes to L2 that are hits.
L2 Cache Freeze Writes	While L2 is in Cache Freeze Mode, this event counts the total number of writes to L2 (that is, the sum total of write hits in L2 and write misses in L2).

(g) L2 Data Events

Event Name	Event Action
L2 Data Access (SRAM)	L1D Accesses to the SRAM locations.
L2 Data Access Hit	Data from L1D to L2 access hit.
L2 Data Access Miss	Data from L1D to L2 accesses qualified by hits/ misses. Can be further split into Read Hits, Read Misses, Write Hits, and Write Misses.
L2 Data Read	L2 Data Reads. Maximum of 2 per cycle. = L2 Data Access Hit Events + Data Read Miss Events.
L2 Data Read (SRAM)	L1D read accesses to SRAM locations.
L2 Data Read Hit	Reads that resulted in an L2 cache hit.
L2 Data Read Miss	Reads that resulted in an L2 cache miss.
L2 Data Request	L1D access, this event is the total of L2 Data Read, L2 Data Write, and L2 Data Victim.
L2 Data Victim	Any L1D access
L2 Data Victim (SRAM)	L1D accesses to SRAM locations qualified by read, write, victim.
L2 Data Write	L2 cache write, total of L2 Data Write Hit and L2 Data Write Miss.
L2 Data Write (SRAM)	L1D write accesses to SRAM locations.
L2 Data Write Hit	Writes that resulted in an L2 cache hit.
L2 Data Write Miss	Writes that resulted in an L2 cache miss.

(h) L2 Line Replace Event

Event Name	Event Action
L2 Line Replace	An L2 Cache line being replaced with a new line.

(i) L2 Long Distance Events

Event Name	Event Action
L2 Long Distance Access	Non-cacheable external memory accesses.
L2 Long Distance Read	Non-cacheable read accesses.
L2 Long Distance Write	Non-cacheable write accesses.

(j) L2 Program Events

Event Name	Event Action
L2 Parallel Prog/Data Access	An L1I and L1D request came in to L2 in the same cycle.
L2 Program Access (SRAM)	L1I accesses that happen to SRAM locations (when the L2 has been configured in partial SRAM and partial cache mode).
L2 Program Access Hit	L2 reads that resulted in a Cache hit. Maximum of 2 per cycle.
L2 Program Access Miss	L1I accesses. These cater only to the cacheable addresses in L2. (Note:An address could be cacheable in L1, but not in L2.)
L2 Program Request	L1I access, caters only to the cacheable addresses in L2. (Note: An address could be cacheable in L1, but not in L2.)

(k) L2 to L1D Events

Event Name	Event Action
L2 to L1D Snoop	An L2 Snoop request to L1D.= L2 Snoop (Dirty) Event + L2 Snoop (Clean/Miss) Events.
L2 to L1D Snoop (Clean/Miss)	An L2 Snoop request to L1D that either did not exist in L1D or was to a line that was not modified in L1D. No data is returned.
L2 to L1D Snoop (Dirty)	An L2 Snoop request to L1D that hit a line that was dirty in L1D. Data is returned from L1D to L2 in this case.

(l) L2 to L1I Events

Event Name	Event Action
L2 to L1I Snoop Hit	An L2 Snoop request to L1D that hit a line that was dirty in L1D. Data is returned from L1D to L2 in this case.
L2 to L1I Snoop Miss	An L2 Snoop request to L1D that either did not exist in L1D or was to a line that was not modified in L1D. No data is returned.
L2 to L1I Snoop Request	An L2 Snoop request to L1D.

(m) Miscellaneous L2 Events

Event Name	Event Action
L2 TR Bus Request	When data is not found in L2, it is fetched from the external world. Using the TR bus, L2 contacts the TC for this fetch. This event occurs in situations such as:Read/Write miss in L2 requiring a fetch; Long distance reads/writes; Victim lines in L2 being evicted; L2 Cache being flushed/written back.
L2 TR Bus Read Request	L2 TR requests for reading data from the external world.
L2 TR Bus Write Request	L2 TR requests for write data to the external world.
L2 Victim Buffer Full	When a dirty line gets evicted in L2, it goes into the victim buffer where it is sent out to the external world. If, in the meanwhile, another dirty line gets evicted, the second one stalls until the first gets fully written out. This event detects such a situation.
L2 Victim Write	L2 cache replacing a line that was dirtied.

(n) Miscellaneous Events

Event Name	Event Action
Non-Cacheable Data Read	CPU read request to a location that cannot be cached (either peripheral registers, or to regions for which MARn register was off).
Non-Cacheable Program Read	CPU read request to a location that cannot be cached (either peripheral registers, or to regions for which MARn register was off).
Parallel Data Request	CPU made a parallel request via D1 and D2 units.
Program Read	CPU Reads. Maximum of 2 per cycle.= Program Read Hit Events + Program Read Miss Events.
Program Read Hit	CPU Reads that resulted in a Cache hit. Maximum of 2 per cycle.
Program Read Miss	CPU Reads that resulted in a Cache miss. Maximum of 2 per cycle.

Table 3–5. TMS320C6414/C6451/C6416 Simulator Events

(a) General Events

Event Name	Event Action
Branch	Branch instruction executed.
NOP	NOP instruction executed.
PIPE_STALL	L1 pipe stall.
XPATH_Forwarding Stall	Stall due to use of crosspath in C64xx.

(b) L1D Cache Events

Event Name	Event Action
L1D Data Bank Conflict	One or more data banks being accessed simultaneously.
L1D Stall	L1D stalled and cannot make more requests to L2.
L1D Victim	Clean line being victimised in L1D.
L1D Dirty Victim	Dirty line being victimised in L1D, line will be written out.
L1D Miss B Side	L1D Cache miss for request coming out of B side of CPU.
L1D Miss A Side	L1D Cache miss for request coming out of A side of CPU.
L1D Hit B Side	L1D Cache hit for request coming out of B side of CPU.
L1D Hit A Side	L1D Cache hit for request coming out of A side of CPU.
L1D Read Hit	L1D Cache hit for CPU read request.
L1D Read Miss	L1D Cache miss for CPU read request.
L1D Write Hit	L1D Cache hit for CPU write request.
L1D Write Miss	L1D Cache miss for CPU write request.
L1D Snoop Dirty	Cache line being snooped in L1D is dirty.
L1D Snoop Clean	Cache line being snooped in L1D is clean.
L1D Write Merge	Consecutive write requests to SRAM got merged.
L1D Store Q Full	Maximum number of write requests pending from L1D.

(c) L1P Cache Events

Event Name	Event Action
L1P Stall	Instruction Fetch unit stalled.
L1P Miss	Fetch packet request from CPU not present in L1P.
L1P Hit	Fetch packet request from CPU not present in L1P.

(d) L2 Cache Events

Event Name	Event Action
L2 Victim	Clean line in L2 being victimised.
L2 Dirty Victim	Dirty line in L2 being victimised, needs to be written out.
L2 DMA Access	DMA access to internal L2 SRAM.
L2 Miss	L2 Cache miss for request coming from L1D or L1P.
L2 Hit	L2 Cache hit for request coming from L1D or L1P.
L2 Read Hit	L2 Cache hit for read request coming from L1D or L1P.
L2 Read Miss	L2 Cache miss for read request coming from L1D or L1P.
L2 Write Hit	L2 Cache hit for write request coming from L1D or L1P.
L2 Write Miss	L2 Cache miss for write request coming from L1D or L1P.
L2 DMA Read	DMA Read access to internal L2 SRAM.
L2 DMA Write	DMA Write access to internal L2 SRAM.

(e) EDMA Events

Event Name	Event Action
TC Channel 0 Request	Priority 0 request packet received by TC Queue manager.
TC Channel 1 Request	Priority 1 request packet received by TC Queue manager.
TC Channel 2 Request	Priority 2 request packet received by TC Queue manager.
TC Channel 3 Request	Priority 3 request packet received by TC Queue manager.
QDMA Request	QDMA submitted a TRP.
EDMA Request	XDMA submitted a TRP.

(f) Peripheral Events

Event Name	Event Action
Timer 0 Interrupt	Timer 0 generated an interrupt.
Timer 1 Interrupt	Timer 1 generated an interrupt.
Timer 2 Interrupt	Timer 2 generated an interrupt.
VCP XEVT	VCP ready to read data.
VCP REVT	VCP ready with data to be read.
VCP Error	VCP Internal error.
TCP XEVT	TCP ready to read data.
TCP REVT	TCP ready with data to be read.
TCP Error	TCP Internal error.
McBSP 0 Transmit Ready	McBSP 0 transmit event
McBSP 0 Receive Ready	McBSP 0 receive event
McBSP 1 Transmit Ready	McBSP 1 transmit event
McBSP 1 Receive Ready	McBSP 1 receive event
McBSP 2 Transmit Ready	McBSP 2 transmit event
McBSP 2 Receive Ready	McBSP 2 receive event

3.3.2 User Options

3.3.2.1 Enable/disable analysis

When you start the analysis tool, analysis is enabled by default. To confirm that analysis is enabled, right-click in the C6000 Simulator Analysis window. In the drop-down menu, notice that a check mark appears next to the Enable Analysis command.

To disable analysis, right-click in the C6000 Simulator Analysis window and select Enable Analysis from the drop-down menu. The check mark next to the Enable Analysis command disappears, indicating that analysis has been disabled. To re-enable analysis, select Enable Analysis again.

In the Analysis Setup dialog, you can enable and disable analysis by selecting and deselecting the Enable analysis check box.

When analysis is disabled, all events that you previously enabled remain unchanged. You can simply re-enable analysis and use the events that are already defined.

3.3.2.2 Count the occurrence of selected events

The analysis tool can count the number of times a selected system event occurs during the execution of your program. A system event that is configured for counting is called a “count event”. Any event that is supported by the simulator can be configured as a count event. You can define as many count events as you choose.

Each event that you select to count is displayed in the C6000 Simulator Analysis window. Two event counters are defined for each event:

Count/Status: For a count event, counts the number of events between each step or breakpoint.

Total Count: Tallies the number of occurrences of an event since the beginning of your analysis session, or the last count reset.

The analysis tool supports all of the debugger’s step and run commands when counting the number of occurrences of each selected event.

To Select Events for Counting:

1) Right-click in the C6000 Simulator Analysis window. Select Analysis Setup from the drop-down menu. The Analysis Setup dialog box appears. The analysis tool automatically polls the simulator and displays the supported events in the Analysis Setup dialog.

2) From the Event Type drop-down list, select Count.

3) From the Setup Events list, select the event(s) you want to count. Notice that the event's Status is Not Configured.

4) Click the Add button (or double-click on the event). The event's Status changes to Enabled and the event is added to the C6000 Simulator Analysis window.

5) Select more events to count, or click the Close button.

The event counters are incremented every time the event occurs. The event counters can be reset at any time.

To Reset Event Counters:

1) Right-click in the C6000 Simulator Analysis window.

2) From the drop-down menu select Reset All Counters.

3.3.2.3 Halt execution whenever a selected event occurs

The analysis tool can halt execution when selected system events occur during the execution of your program. A system event that is configured to halt execution is called a "break event." Any event that is supported by the simulator can be configured as a break event. You can set as many break events as you choose. The analysis tool halts execution on the occurrence of any break event.

Each break event is displayed in the C6000 Simulator Analysis window. The following information is displayed:

Count/Status: For a break event, displays the Status of the event: Configured or Triggered.

Break Address: The address at which execution halts.

Routine: The name of the routine in which execution halts.

To Select Break Events:

1) Right-click in the C6000 Simulator Analysis window and select Analysis Setup from the drop-down menu. The Analysis Setup dialog box appears. The analysis tool automatically polls the simulator and displays the supported events in the Analysis Setup dialog.

2) From the Event Type drop-down list, select Break.

3) From the Setup Events list, select the event(s) to be configured as a break event. Notice that the event's Status is Not Configured.

4) Click the Add button (or double-click on the event). The event's Status changes to Enabled and the event is added to the C6000 Simulator Analysis window.

5) Select more events, or click the Close button.

3.3.2.4 Delete count or break events

During a single analysis session, you may want to change your analysis parameters several times. For example, you may want to define new parameters such as counting program read hits or tracking data writes.

You can delete the currently defined parameters that you no longer want to track and then select new count or break events.

To Delete a Defined Count or Break Event:

1) Right-click in the C6000 Simulator Analysis window and select Analysis Setup from the drop-down menu. The Analysis Setup dialog box appears.

2) From the Setup Events list, select the event that you want to remove from analysis. Notice that the event's Status is Enabled.

3) Click the Delete button (or double-click on the event). The event's Status changes to Not Configured and the event is removed from the C6000 Simulator Analysis window.

4) Remove other events, or select the Close button.

3.3.2.5 Create a log file

In addition to viewing the analysis data in the C6000 Simulator Analysis window, you may choose to create a log file to record the output of the analysis tool.

NOTE that the log file only logs count events. No break events will be logged.

To Create a Log File:

1) Right-click in the C6000 Simulator Analysis window and select Analysis Setup from the drop-down menu. The Analysis Setup dialog box appears.

2) Click the Open Log button. The Select Log File dialog box appears.

3) Specify a name for your log file in the File name field.

4) Click Open.

To stop logging the occurrence of events, return to the Analysis Setup dialog and click the Close Log button.

3.3.3 Configurations Not Supporting Analysis Events

C62xx_big_endian_sim.ccs

C62xx_little_endian_sim.ccs

C67xx_big_endian_sim.ccs

C67xx_little_endian_sim.ccs

sim62xx_fast_simulator.cfg

sim67xx_fast_simulator.cfg

3.4 RTDX/BIOS Support

RTDX is supported when running inside the simulator. To run inside the simulator, you must link your application with the RTDX Simulator Target Library. It is easy to switch applications from running inside the simulator to running on real hardware. All that is required is to link the target application with the Target Library that matches your target environment. To run under the simulator, link with RTDX Target Simulator Library. No recompile of your target application is necessary.

Please note that, as expected, applications run more slowly on the simulator than on real hardware. As a result, your RTDX clients may frequently receive the return status code ' ENoDataAvailable' from calls into the RTDX host interface, due to increased latency between messages that are received. You must make sure that your client application is carefully constructed to deal with these timing issues. Also, be aware that in the present RTDX simulator implementation, there is no target buffering of messages, as with the emulator versions. Therefore, there is no need for you to reconfigure your buffer to accommodate different message sizes.

You can find more information about RTDX/BIOS support in the Code Composer Studio IDE online help.

NOTE: RTDX/BIOS is not supported on Solaris.

3.5 BootLoad

The Bootload is a process that copies a definite number of words (the exact number differs for C620x/C670x and C621x/C64x devices) from a certain address (specified by the bootmode selected through configuration file) to address 0x0. The Bootload happens only if it is enabled by specifying a valid Bootmode through a simulator configuration file. This section describes the-Bootload process in different C6x device simulators. For more details regarding Boatload please refer to chapter 11 of the *TMS320C6000 Peripherals Reference Guide* (literature number SPRU190).

3.5.1 Bootload in C6x0x Device Simulators

In this category of device simulators, the Bootload occurs by way of DMA as it copies 64K bytes from the EMIF CE1 space to address 0. C6x0x devices have two memory maps, MAP0 and MAP1. The maps differ in that MAP0 has external memory mapped at address 0, whereas MAP1 has internal memory mapped at address 0. The following are the Bootmodes supported in the C6x0x device simulators:

- ROM_8BIT
- ROM_16BIT
- ROM_32BIT

The Bootmode can be specified in the simulator configuration file. For example, to choose ROM_8BIT Bootmode in the C6202 device simulator, the config 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 simulator config file. To do so the config file should have 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.

3.5.2 Bootload in C6x11 Device Simulators

The Bootload feature is not supported in c6x11 device simulators.

3.5.3 Bootload in C64x Device Simulators

In C64x device simulators the process of bootload, if enabled by specifying a valid Bootmode in the simulator config file, copies 1K byte (256 words) into the memory at address 0 (i.e. internal memory in this case). For enabling bootload, the ”BOOTMODE” flag in the simulator config file should be set any of the following values:

- QUICK
- EMIFA
- EMIFB

For example, in the C64x simulator, in order to choose EMIFA Bootmode the simulator config file should have following section specified:

```
MODULE C64xx;

    BOOTMODE EMIFA

END C6xx;
```

If not specified, or set to ”NONE” or to any other invalid value through the config file, BOOTMODE by default is set to ”NONE”. This means that no Bootload will happen.

The difference between the these three Bootmodes listed above is that choosing EMIFA as BOOTMODE causes Bootload process to copy 256 words from the EMIFA CE1 space (i.e. 0x90000000) to internal memory address 0 through EDMA, whereas choosing EMIFB as BOOTMODE causes Bootload process to copy 256 words from the EMIFB CE1 space (i.e. 0x64000000) to internal memory address 0 through EDMA. Bootmode QUICK does a simple memcopy (not through EDMA like Bootmodes EMIFA and EMIFB do) of 256 words from EMIFA CE1 space (i.e. 0x90000000) to internal memory. QUICK Bootmode is therefore faster and doesn’t result in simulation cycles.

3.6 Simulation Accuracy

Simulation accuracy is how close the simulator is to the design in terms of cycle counts and in terms of event counts. Event counts generally do not differ from hardware a great deal and most of the simulation accuracy is concerned with cycle count matching.

3.6.1 Fast Simulator Configurations

The fast simulator configurations can be used when one is only interested in the accuracy of the core and does not need full device feature simulation.

Table 3–6. Fast Simulator Configurations

Configuration	Device Cores Simulated
sim62xx_fast_simulator.cfg	C6201, C6202, C6203, C6204, C6205, C6701, C6211, C6711
sim64xx_fast_simulator.cfg	C6414, C6415, C6416

The sim62xx_fast_simulator.cfg simulates the core of the C6201, C6202, C6203, C6204, C6205, C6701, C6211, and C6711 devices accurately (100%). Any accessess outside core will be handled by a flat memory and hence these accesses will be inaccurate.

The sim64xx_fast_simulator.cfg simulates the core of the C6414, C6415, and C6416 devices accurately (100%). Any accesses outside the core will be handled by a flat memory and hence these accesses will be inaccurate.

3.6.2 Cache Simulator Configurations

The cache simulator configuration should be used when the user cares about the accuracy of the core and the cache system and does not plan to use the full device features.

Table 3–7. Cache Simulator Configurations

Configuration	Device Cores Simulated
sim64xx_csim_simulator.cfg	C6414, C6415, C6416

The sim64xx_csim_simulator.cfg simulates the core and the L1 and L2 cache accurately. While the core is 100% accurate, the cache is not totally accurate. When accuracy numbers were gathered for the simulator, cache was found to

have an accuracy greater than 85% when run on the design applications. Any accesses beyond the cache will be handled by a flat memory and hence will be inaccurate.

More information regarding the accuracy of the core and cache for C64xx can befound in section 3.6.4 on page 3-36.

3.6.3 Device Simulator Configurations

The device simulator configuration should be used when the user wants to use the features of the device (peripheral set supported) and which are not present in core or cache simulators.

Table 3–8. Device Simulatior Configurations

Device	Configuration
C6201	c6201_ltl_endian_sim_map0.ccs c6201_big_endian_sim_map0.ccs c6201_ltl_endian_sim_map1.ccs c6201_big_endian_sim_map1.ccs
C6202	c6202_ltl_endian_sim_map0.ccs c6202_big_endian_sim_map0.ccs c6202_ltl_endian_sim_map1.ccs c6202_big_endian_sim_map1.ccs
C6203	c6203_ltl_endian_sim_map0.ccs c6203_big_endian_sim_map0.ccs c6203_ltl_endian_sim_map1.ccs c6203_big_endian_sim_map1.ccs
C6204	c6204_ltl_endian_sim_map0.ccs c6204_big_endian_sim_map0.ccs c6204_ltl_endian_sim_map1.ccs c6204_big_endian_sim_map1.ccs
C6205	c6205_ltl_endian_sim_map0.ccs c6205_big_endian_sim_map0.ccs c6205_ltl_endian_sim_map1.ccs c6205_big_endian_sim_map1.ccs
C6701	c6701_ltl_endian_sim_map0.ccs c6701_big_endian_sim_map0.ccs c6701_ltl_endian_sim_map1.ccs c6701_big_endian_sim_map1.ccs
C6211	c6211_ltl_endian_sim.ccs c6211_big_endian_sim.ccs
C6711	c6711_ltl_endian_sim.ccs c6711_big_endian_sim.ccs
C6414	c6414_ltl_endian_sim.ccs c6414_big_endian_sim.ccs
C6415	c6415_ltl_endian_sim.ccs c6415_big_endian_sim.ccs
C6416	c6416_ltl_endian_sim.ccs c6416_big_endian_sim.ccs

The C6201, C6202, C6203, C6204, C6205, C6701 device simulators model the core with 100% accuracy.

Timers and McBSP are modeled with100% accuracy.

The C6211 and C6711 device simulators model the core with 100% accuracy. The L1 and L2 are also modeled 100% 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. No

proper measures were taken to find out exact accuracy numbers for EDMA and EMIF. Timers are the only peripherals with 100% accuracy.

The cores of the C6414, C6415, and C6416 device simulators are modeled accurately. The L1 and L2 Cache models are not completely accurate, both in terms of cycles and also in terms of some event counts.

For more on the cache accuracy refer to section 3.6.4. Here again, timers are the only peripherals modeled with 100% accuracy. Others, like the EDMA subsystem and the EMIF, will consume cycles proportional to what they consume on design for different parameters. EMIF ASYNC Controller was modeled accurately, but may not able to see this due to inaccuracies in the path to the EMIF (EDMA subsystem).

L1P has the biggest inaccuracies of the three cache components. The cycle difference in the L1P occurs as a result of differences between the way the instruction fetch requests are modeled and how the occur in the design.

3.6.4 C64xx Cache Event Count Accuracy

The main cache events, which must be as accurate as possible for on chip accesses, are the requests that are issued between the L1 to the L2. These will have the most amount of time delays. From L1P we have only the L1P instruction read misses. L1D will make three requests, data read misses, data write misses, and victims.

These four events were compared for event count accuracy on the simulator versus design using the design kernels and applications. The number of L1D to L2 victims was the same between design and simulator. The L1D read miss numbers were also 100% accurate. The number of L1D write misses were on an average 95% accurate.

3.6.5 C64xx Cache Cycle Count Accuracy

Some of the basic scenarios of L1P,L1D reads and L1D writes are presented in this section, depicting the difference in cycles between the C64xx design and C64xx cache simulator. For the measurements given below both Code and Data reside in internal SRAM.

Table 3–9. C64xx L1D Read Cycle Difference in Some Scenarios

Scenario	Design	ISS	Difference
single load miss to SRAM	13	13	0
ld miss || ld hit (same line) to SRAM	13	14	1

Table 3–9. C64xx L1D Read Cycle Difference in Some Scenarios(Continued)

Scenario	Design	ISS	Difference
ld miss || ld miss to SRAM	15	15	0
30 pipelined || LD misses (15 EP)	85	85	0
31 pipelined || LD misses (16 EP)	87	88	1
ld hit || ld hit DBC	8	8	0
ld hit || ld hit no DBC	7	7	0
30 pipelined LD hits with DBC (15 EP)	36	36	0
30 pipelined load misses with DBC (15 EP)	70	76	6
40 pipelined hit || miss with DBC, hit A side –> hit B side (20 EP, 10 + 10)	96	96	0

Table 3–10. C64xx L1D Write Cycle Difference in Some Scenarios

Scenario	Design	ISS	Difference
1 ST miss	2	2	0
30 pipelined store misses	16	16	0
30 pipelined load || store hits (15 EP) with Data bank conflicts	28	31	3
30 pipelined || dword stores going serialy	38	33	–5
30 pipelined || word stores going serialy	23	22	–1
30 pipelined word stores going serialy	38	37	–1

Table 3–11. C64xx L1P Cycle Difference in Some Scenarios

Scenario	Design	ISS	Difference	Remark
L1P 80 1 ep wide instructions	89	80	–9	All L1P misses
L1P 101 2 ep wide instructions	113	153	40	All L1P misses
L1P 101 3 ep wide instructions	167	203	36	All L1P misses
L1P 101 4 ep wide instructions	245	245	0	All L1P misses
L1P 101 5 ep wide instructions	280	303	23	All L1P misses
L1P 101 6 ep wide instructions	336	348	12	All L1P misses
L1P 101 7 ep wide instructions	397	398	1	All L1P misses
L1P 101 8 ep wide instructions	500	450	–50	All L1P misses

3.7 Known Limitations

This section describes features that are not modeled in C6000 device simulators.

3.7.1 Known Limitations in C6x0x Device Simulators

General

- The HPI is not modeled.
- XBUS 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 sync. events, DSPINT and SDRAM_INT, are not supported.
- The sync. events are captured only if they are enabled.

3.7.2 Known Limitations in C6x11 Device Simulators

General

- No support for HPI.
- No support for McBSP.
- The BootLoad feature is not supported.

DMA

- QDMA for simple and quick data transfers is not supported.
- Sync events DSPINT and SDRAM_INT are no supported.

3.7.3 Known Limitations in C64x Device Simulators

General

- No support for HPI/PCI.
- No support for Utopia.
- No support for GPIO.

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 sync events DSPINT, GPIOINT, SD_INTA, SD_INTB, PCI, UREVT and UXEVT are not modeled.

Cache

- L2 does not submit TR (Transfer Request) on all priority. It does so 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.

Appendix A

Serial Port XBAR Configuration File Format

Topic Page

A.1 XBAR Connectivity

The Instruction Set Simulator™ provides a mechanism by which the user can interconnect two McBSPs to test and validate code written for serial transfer. The XBAR (crossbar) serves as a testbench which can be ”programmed” to set up this desired connectivity.

Typically, the user programs one McBSP for transmission and another McBSP for reception. Thus, DX and DR pins of the corresponding McBSPs have to be hooked up to each other. Similarly, clock and frame synchronization signals also have to be interconnected. The XBAR component in the ISS allows specification and implementation of this connectivity.

A.1.1 How To Use This Feature ?

The XBAR connectivity is specified in a XBAR data file. The section below describes the file format in detail. The path and file name of the XBAR data file must be included in a configuration file (.cfg) which is selected in cc_setup. Subsequently, when cc_app is invoked, the XBAR functionality will be in effect.

A.1.2 How to write a XBAR file ?

1. Supported Pin Names are:

DX0, DX1, DX2 – denote DX pins of McBSP0, McBSP1 and McBSP2 respectively.

DR0, DR1, DR2 – denote DR pins of McBSP0, McBSP1 and McBSP2 respectively.

CLKX0, CLKX1, CLKX2 – denote CLKX pins of McBSP0, McBSP1 and McBSP2 respectively.

CLKR0, CLKR1, CLKR2 – denote CLKR pins of McBSP0, McBSP1 and McBSP2 respectively.

FSX0, FSX1, FSX2 – denote FSX pins of McBSP0, McBSP1 and McBSP2 respectively.

FSR0, FSR1, FSR2 – denote FSR pins of McBSP0, McBSP1 and McBSP2 respectively.

2. Connectivity is specified by means of (source pin, destination pin) pair specifications. Each pin name must be on a separate line. The source pin name is specified first (on one line), followed by the name of the destination pin (on a second line).

For instance, to connect DX0 and DR1, the XBAR file entry is:

DX0

DR1

3. NO COMMENTS ARE SUPPORTED BY THE FILE SYNTAX. NO BLANK LINES SHOULD BE PRESENT BETWEEN SPECIFICATION LINES.

4. The connecitivity can be mentioned in any order but please make sure that each pair specifies a driver-driven connection e.g. to mention DX1 ->DR0 , CLKX1 ->CLKR0 , FSX1 ->FSR0 connectivity please follow the valid XBAR connectivity :

/*valid XBAR connecitivity*/	/*Invalid XBAR connectivity*/
DX1	DX1
DR0	CLKX1
CLKX1	FSX1
CLKR0	DR0
FSX1	CLKR0
FSR0	FSR0

5. Please ensure that there are no blank lines at the END OF THE FILE. This might cause problems.

A.1.3 Format of the Config File to be Picked Up

In the configuration file that is going to be picked up by the simulator dll (via cc_setup), add entry MCBSP_XBAR_FILE.

In case of TMS320C6x0x devices add the entry MCBSP_XBAR_FILE as :

```
MODULE TB;
   DYNAMIC_LIB  libtbc6x0x.dll; //this should already be
present
   MCBSP_XBAR_FILE <path_of_xbar_file_name>;
   CSSI_INIT_FUNC NewTestBench;
END TB;
```

On the other hand, in case of TMS320C64xx devices add the entry MCBSP_XBAR_FILE as:

```
MODULE C64xx;

      MCBSP_XBAR_FILE  <path of xbar_file_name> ; // path
of xbar data file.

END C64xx;
```

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

```
MCBSP_XBAR_FILE  /ccs/drivers/mcsp_xbar.dat ;
```

Appendix B

Pin Connect File Format

Topic Page

B.1 Pin Connect File Format

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:

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

B.1.1 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.

Your 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 (e.g. FSX0 pin in C6201 simulator). This value (0 or 1) must be used to force the pin value to low or high at the corresponding cycle. A logic value of 0 causes the pin value to go low, and a logic value of 1 causes it to go high. For example, `[12,1][56,0][78,1]` If connected to the FSX0 pin in C6201, this will cause the pin to go high at the twelth 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.

B.1.2 Absolute Clock Cycle

To use an absolute clock cycle, your cycle value must represent the actual CPU clock cycle where you want to simulate an interrupt. For example:

```
12 34 56
```

Interrupts are simulated at the 12th, 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.

B.1.3 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 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. You can mix both relative and absolute values in your data file.

B.1.4 Repetition of Patterns for a Specified Number of Times

You can format your 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 5th CPU cycle, then the 15th (5 + 10), 35th (15 + 20), 45th (35 + 10), and 65th (45 + 20) CPU clock cycles.

B.1.5 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 simulated at the 10th CPU cycle, then the 15th (10 + 5), 35th (15 + 20), 40th (35 + 5), 60th (40 + 20), 65th (60 + 5), and 85th (65 + 20) CPU cycles, continuing in that pattern until the end of simulation.

Appendix C

How to Debug Using the Simulator

This section gives a brief overview of the some methods that can be adopted to debug an application being run on C6000 device simulators.

Topic **Page**

C.1 Using Steps/Breakpoints and Watch Windows

If the application being run is written on C, then CCStudio watch windows can be used to inspect the values of the variables at various points of execution by single stepping through the code. The value of a variable at a specific point can also be viewed by putting a breakpoint at that place and running the code up to that point.

To watch the value a variable using watch window:

1) Highlight the variable to be inspected.

2) Right click on it and select ”Add to watch window”

This should show the variable and its value in the watch window in the bottom of the CCStudio IDE. The radix of the variable’s value can be changed through the Radix drop down menu, available in watch window. Double-clicking on the variable in the watch window during the execution and providing a new value can also change the value of the variable.

In case of an application in assembly, CPU registers can be viewed at various points in the execution either by stepping through the code or by putting in breakpoints and running the code. To view CPU registers select View ->CPU Registers ->CPU registers from the CCStudio menu. In this case as well the value of a register can be changed by double-clicking on the register and providing the new value.

C.2 Using Memory Windows

In some cases you might want to view contents of a memory location. For example, consider an application doing a DMA transfer. In debugging such an application you might want to match source and destination memory to make certain they have the same contents for a specified number of elements. In order to accomplish this, the memory windows can be opened for both source and destination addresses and their contents can be viewed/compared. To open a memory window select View ->Memory from the CCStudio IDE menu and enter the address whose contents you wish to view in the address input.

C.3 Using Analysis Events

The analysis events also serve as a nice tool to debug many applications. The various types of analysis events, their applicability to various configurations of C6000 device simulators and how to set them up is very well described in section 3.3 of this document, beginning on page 3-11.

Consider a simple C64xx application which programs a DMA transfer in XDMA channel1, so that it gets triggered on sync event TINT0 (Timer 0 interrupt event connected to XDMA channel no. 1). Then timer 0 is programmed to count. So as timer 0 interrupt comes, XDMA channel 1's transfer gets initiated. The application will look like:

```
/* Filename: EDMADefines.h*/
/* Start of EDMADefine.h */
#ifndef  _EDMA_DEFINES_H
#define _EDMA_DEFINES_H
#define EDMA_CHANNEL_TINT0         0x01a00018
#define EDMA_REGISTER_START      0x01a0ff9c
#define TIMER0_REGISTER_START   0x01940000
/* EDMA parameters structure */
struct EdmaParams
{
    unsigned int options;
    unsigned int src_address;
    unsigned int frm_and_elem_count;
    unsigned int dst_address;
    unsigned int frm_and_elem_index;
    unsigned int elem_reload_and_link_address;
};
/* EDMA Registers structure */
struct EdmaRegisters{
    unsigned int EVT_POL_REG_HI;
    unsigned int reserved1;
    unsigned int XINT_SRC_HI;
    unsigned int XINT_SRC_EN_HI;
    unsigned int XCOMP_EN_HI;
    unsigned int EVT_REG_HI;
    unsigned int EVT_EN_REG_HI;
```

```
    unsigned int EVT_CLR_REG_HI;
    unsigned int EVT_SET_REG_HI;
    unsigned int QALLOC0_REG;
    unsigned int QALLOC1_REG;
    unsigned int QALLOC2_REG;
    unsigned int QALLOC3_REG;
    unsigned int reserved2;
    unsigned int reserved3;
    unsigned int reserved4;
    unsigned int EVT_POL_REG_LO;
    unsigned int CHNL_EMPTY;
    unsigned int XINT_SRC_LO;
    unsigned int XINT_SRC_EN_LO;
    unsigned int XCOMP_EN_LO;
    unsigned int EVT_REG_LO;
    unsigned int EVT_EN_REG_LO;
    unsigned int EVT_CLR_REG_LO;
    unsigned int EVT_SET_REG_LO;
};
/*Timer Registers structure */
struct TimerRegisters{
    unsigned int TIMER_CTRL_REG;
    unsigned int TIMER_PRD_REG;
    unsigned int TIMER_CNT_REG;
};
#endif /* _EDMA_DEFINES_H*/
/* End of EDMADefine.h */
/* Filename: EDMATint0.c */
/* Start of EDMATint0.c*/
#include <EDMADefines.h>
#include <stdio.h>
void main()
{
    /* Declarions */
    struct EdmaParams   *channel_tint0_params  = (struct EdmaParams*)EDMA_CHAN-
NEL_TINT0;
```

```
    struct EdmaRegisters  *edma_registers   = (struct EdmaRegisters*)EDMA_REG-
ISTER_START;
    struct TimerRegisters *timer0_registers = (struct TimerRegisters*)TIMER0_REG-
ISTER_START;


    /* Enable channel 1 by writing to corresponding EER (Event enable
     * register bit ) */
    edma_registers->EVT_EN_REG_LO = 0x2;
    /* Program EDMA channel 1 (for sync event TINT0 i.e. timer 0
     * interrupt) */
    channel_tint0_params->options = 0x41370001;


    /**********************Option field structure ***********************
     * This transfer will send back a completion code of 7
     *
     *     0x41370001 =  01000001001101110000000000000001b
     *
     *
-------------------------------------------------------------------------------
     *     Bit position         Field-name                        value
     *
-------------------------------------------------------------------------------
     *       31-29           priority                      2
     *       28-27           element size                  0 (Words)
     *       26              2D source                     0
     *       25-24           src addr update mode          1 (Increment)
     *       23              2D destination                0
     *       22-21           dst addr update mode          1 (Increment)
     *       20              tcint                         1
     *       19-16           tcc                           7
     *       15              reserved                      0
     *       14-13           tcc5-tcc4 (MSBs of tcc)       0
     *       12              actint                        0
     *       11              reserved                      0
     *       10-5            atcc                          0
     *       4               reserved                      0
     *       3               pdts                          0
```

```
     *       2               pdtd                    0
     *       1               link                    0
     *       0               frame synch             1
 ************************************************************************/

    channel_tint0_params->src_address                 = 0x90001000;
    channel_tint0_params->frm_and_elem_count          = 0x10;
    channel_tint0_params->dst_address                 = 0x90004000;
    channel_tint0_params->frm_and_elem_index          = 0x0;
    channel_tint0_params->elem_reload_and_link_address = 0x0;

    /* Program Timer 0 to count 0x100 (i.e. upto 0x100*8 cycles as
     * timer counts at cpu/8 clock ratio) */
    timer0_registers->TIMER_CTRL_REG = 0x2d3;
    timer0_registers->TIMER_PRD_REG   = 0x100;

    /* Poll for the channel 1's transfer (which will be triggered on
     * sync event TINT0) to get over, by looking at completion code set
     * by the transfer in XINT_SRC register*/
    while(!((edma_registers->XINT_SRC_LO) & (1 << 7)));

    printf("PASSED\n");
}
/* End of EDMATint0.c*/
```

In debugging this kind of application, the following steps would help:

(1) Open Tools–>C6000 Simulation Analysis.

(2) Right click on analysis window and select analysis setup.

(2) Select "Timer 0 interrupt" event.

(3) Select "EDMA Request" event.

(4) Configure them to "break".

NOTE: This application should be run using any of the following .ccs configurations (select the configuration from CCStudio Setup):

- c6416_ltl_endian_sim.ccs
- c6416_big_endian_sim.ccs
- c6414_ltl_endian_sim.ccs
- c6414_big_endian_sim.ccs

(For more information on selecting and configuring events please refer to section 3.3 on page 3–9.)

The above mentioned steps will cause the execution to stop each time either of the two events occur. Now run the application. The following behavior should occur:

(1) The execution should at first stop on the timer 0 interrupt.

(2) Running further should cause EDMA to receive TINT0 sync event.

(3) EDMA channel for sync event TINT0 should trigger its transfer since it is enabled.

(4) Application should again stop the execution on the ”EDMA Request” event, meaning that an EDMA request has been initiated. Run it further.

The source and destination addresses for this transfer (which are 0x90001000 and 0x90004000 respectively) can therefore be viewed using memory windows. The destination address should start getting updated with the contents of the source address. Finally, both source and destination should have same contents (for element count number of words).

Now suppose the ”Timer 0 interrupt” event doesn’t occur. This would mean that something is wrong in the programming timer, so this problem can be checked for. Similarly if ”Timer 0 interrupt event” occurs but ”EDMA Request” event doesn’t happen, then we should check whether channel TINT0 (i.e. EDMA channel 1) is enabled and is properly programmed with correct set of transfer parameters.

Appendix D

How to Optimize Using the Simulator

The simulator can be used to optimize code so that the application takes fewer cycles and runs for the least amount of time. However it does not give the user much information to help in optimizing his application for space. The main simulation tools that are used for detecting cycle consumption and time are analysis events and the profiler.

We will use the C64xx simulators in our discussion on how to optimize applications, and most of the methods discussed can be used in a similar manner across simulators. Also when we refer to functions we are talking about applications written in C.

Topic **Page**

D.1 Bringing Code and Data On Chip

One way of reducing cycles is by minimizing the amount of off-chip access and trying to put as much of code and data on chip (internal RAM). Before doing this, however, we need to see how the simulator can be used to detect where the cycles are being consumed. First, we need to identify functions or modules that take the biggest percentage of time. The time split up of a function/module is made up of two contributors - the code of which it is made up, and the data which the code accesses.

Whenever possible put both the code and the data onto the internal RAM. For large applications, however, this may not be always possible. Our first step is to profile the code. Note that it is not possible to actually profile data. Data profiles can be estimated to some extent in a function by using range-type profile points over pieces of code that access particular data. If possible, try to set profile points in all the functions of the code, or at least in the top functions in terms of time percentage expected. Documentation on how to profile can be found in the Code Composer Studio IDE v2.0 online help.

Here we will present a example which we will try to optimize using the profiler. The lnk.cmd file and the main.c (only file) look like this.

```
link.cmd
=======
 –c
–heap  0x2000
–stack 0x4000
–e c_int00

–l d:\ti\c6000\cgtools\lib\rts6400.lib
–l d:\ti\c6000\bios\lib\csl6400.lib

/* Memory Map 1 – the default */
MEMORY
{
        SRAM o = 00001000h   l = 0000200h
        PMEM:      o = 90000000h   l = 00010000h
        EXTM:      o = 80020000h   l = 00010000h
        DATA1:  o = 80000000h   l = 00010000h
        DATA2:  o = 80010000h   l = 00010000h
```

```
}
SECTIONS
{
    .text       >       PMEM

    .stack      >       EXTM
    .bss        >       EXTM
    .cinit      >       EXTM
    .cio        >       EXTM
    .const      >       EXTM
    .data       >       EXTM
    .switch     >       EXTM
    .sysmem     >       EXTM
    .far        >       EXTM
    .array_A >      DATA1
    .array_B >      DATA2
    .array_sum>     DATA1
}
```

The program file is given below

```
main.c
======
#include <stdio.h>

#define L2_LINE_SIZE  32 //words
#define ROWS L2_LINE_SIZE
#define COLS ROWS
#pragma DATA_SECTION (sum,".array_A");
#pragma DATA_SECTION (sum,".array_B");
#pragma DATA_SECTION (sum,".array_sum");
/*
#pragma CODE_SECTION (matrixAllElementsIncrement,".fn_ma-
trixAllElementsIncrement");
#pragma CODE_SECTION (matrixLeftHalfIncrement,".fn_matrix-
LeftHalfIncrement");
#pragma CODE_SECTION (matrixDiagonalElementsIncre-
ment,".fn_matrixDiagonalElementsIncrement");
```

```
#pragma CODE_SECTION (matrixDiagonalPairElementsIncre-
ment,".fn_matrixDiagonalPairElementsIncrement");
#pragma CODE_SECTION (matrixDiagonalQuadrupleElement-
sIncrement,".fn_matrixDiagonalQuadrupleElementsIncre-
ment");
#pragma CODE_SECTION (addFirstRows,".fn_addFirstRows");
*/
unsigned int a[ROWS][COLS], b[ROWS][COLS];
unsigned int sum[ROWS][COLS];


void matrixAllElementsIncrement( void );
void matrixLeftHalfIncrement( void );
void matrixDiagonalElementsIncrement( void );
void matrixDiagonalPairElementsIncrement( void );
void matrixDiagonalQuadrupleElementsIncrement( void );
void addFirstRows( void );
void main( void )
{
  int k;
  unsigned int * address, data;

  // CE space starting at 0x80000000
  address = (unsigned int *) 0x01800008;
  *address = 0x20528321;
  data = *address;

  // Set L2 to 256K Cache
  address = (unsigned int *) 0x01840000;
  *address = 0x7;
  data = *address;

  // MAR 90 set to Cacheable (0x80000000)
  address = (unsigned int *) 0x01848200;
  *address = 0x1;
  data = *address;
```

```
  matrixAllElementsIncrement();
  for(k=0;k<10;k++)
  {
    matrixLeftHalfIncrement();
    matrixDiagonalElementsIncrement();
    matrixDiagonalElementsIncrement();
    matrixDiagonalPairElementsIncrement();
    matrixDiagonalQuadrupleElementsIncrement();
  }
  addFirstRows();
}
//increment all the elements of the matrix
void matrixAllElementsIncrement( void )
{
  int i,j;

  for(j=0;j<COLS;j++)
  {
    for(i=0;i<ROWS;i++)
    {
      a[i][j]++;
    }
  }
}

void matrixLeftHalfIncrement( void )
{
  int i,j;

  for(i=0;i<ROWS;i++)
  {
    for(j=0;j<COLS/2;j++)
    {
      a[i][j]++;
```

```
    }
  }
}
void matrixDiagonalElementsIncrement( void )
{
  int i;

  for(i=0;i<ROWS;i++)
  {
    a[i][i]++;
  }
}
void matrixDiagonalPairElementsIncrement( void )
{
  int i;

  for(i=0;i<ROWS-1;i++)
  {
    a[i][i]++;
    a[i][i+1]++;
  }
}
void matrixDiagonalQuadrupleElementsIncrement( void )
{
  int i;

  for(i=0;i<ROWS-3;i++)
  {
    a[i][i]++;
    a[i][i+1]++;
    a[i][i+2]++;
    a[i][i+3]++;
  }
}
void addFirstRows( void )
```

```
{
  int i;

  for(i=0;i<COLS;i++)
  {
    sum[i][0] = a[i][0] + b[i][0];
  }
}
```

These two files can be loaded into a CCStudio project and a coff file can be created. Note that the linker command file has only 0x200 size for the SRAM region even though this is not true for the 64xx device and simulators. We deliberately choose these values so that we get a feel of the real life situation in which we will be having only limited SRAM.

Our task at hand is to see what functions we take on chip. We will do this by using the profiler. To enable profiling follow the following steps.

1) From the Profiler Menu click on ”Enable Clock”.

2) From the Profiler Menu select ”Start New Session”.

3) Type in the a session name. In the Profile window that pops up, select the ”Profile All Functions” icon.

4) Run the program

Once the program has finished the details of the profiling can be found in the profile window. There are many columns which will give different Inclusive (Function + All Descendants Measures) and Exclusive (Functions without Descendants Measures) numbers over the program run. A good measure for deciding which pieces of code we take on chip would be the Exclusive Total Count. In this example we get the following numbers:

Table D–1. Total Exclusive Counts by Profiler for All Functions

Function	Total Exclusive
matrixLeftHalfIncrement	67404550
matrixAllElementsIncrement	13347285
matrixDiagonalQuadrupleElementIncrement	6855290
matrixDiagonalElementIncrement	6712740
matrixDiagonalPairElementIncrement	4810110
addFirstRows	463708
matrixDiagonalPairElementIncrement	130430

The total amount of time that the application took was 100208721.

After obtaining the Data we try to fit as much of code on chip by starting with the functions taking the most number of cycles. In order to move different functions to different functions we need to compile in such a way that we support

- Far Calls (–ml1)
- Function Subsection (–mo)

Both these options can be set by selecting ”Build Options” and then clicking on ”Advanced” under the compiler Tab. Once we do this we rebuild all after changing our linker command file to locate our functions into internal SRAM. In this example due to our SRAM constraints we set, we are able to shift only the first two functions into SRAM. We do this by adding the following two lines to the linker command file.

```
.text:_matrixLeftHalfIncrement     >      SRAM
.text:_matrixAllElementsIncrement  >      SRAM
```

On running the new coff file we see that the application now only takes around 20 Million cycles compared to 100 Million it took when everything was in external memory.

This method of bringing Code on chip has a flaw that we took into Account not only the Code time per function but also the Data access time. There is no Data profiling at the moment, but we do have two ways of solving the problem.

1) By profiling the regions of code which make data accesses.

This is done by setting Range type profile points over code which make data accesses. These values can then be subtracted from the function’s numbers to give approximate Program miss numbers. This method is however not practical due to the following reasons.

a) Code could be large, needing to set a lot of range type profile points.

b) It is tough to know some of the non-code accesses, like function arguments, calls.

c) Cannot separate Code and Data contribution for a line of code.

2) By profiling using Other Analysis Events as counts.

A really powerful way in which the simulator helps in Profiling is in its ability to profile on any of the Analysis Events Supported. The Cache events can be used for deciding on L1P, L1D, L2 Data hit and Miss counts, which will give a good idea for the user on what to optimize. To profile on an event do the following:

i) Select ”Clock Setup” in the ”profiler menu.

ii) Click on the ”Count” Drop down list which has a list of the analysis events. Select the one you want to profile on, e.g. ”L1P miss”.

iii) Run the profiler

The counts obtained now are for the event.

Index

A

B

C

D

E

F

G

I

K

M

O

P

R

S

W

X