TMS320C54x Instruction Set Simulator

Technical Reference

Literature Number: SPRU598B
July 2002–Revised October 2003

TEXAS INSTRUMENTS

Contents

Preface **5**

1 Introduction to the TMS320C54x Simulator **7**
- 1.1 Features 8
- 1.2 Processors Supported by the C54x Simulator 8
- 1.3 Considerations for Choosing a Simulator 9
- 1.4 Supported Hardware Resources 9
  - 1.4.1 CPU 9
  - 1.4.2 Memory 10
  - 1.4.3 Peripherals 10

2 Supported Simulation Features **11**
- 2.1 External Event & Data Simulation 12
  - 2.1.1 Pin Connect 12
  - 2.1.2 Port Connect 16
- 2.2 Pipeline Latency Warning 17
- 2.3 Simulation Pipeline Modes 19
  - 2.3.1 Switching Between the Modes 19
  - 2.3.2 Effects of Mode Switching 20
  - 2.3.3 Recommendations on Simulation Mode 20
  - 2.3.4 Related Warnings 20
- 2.4 Simulator Analysis 21
- 2.5 RTDX 21
- 2.6 DSP/BIOS 21

3 Configuration Specifics **23**
- 3.1 Detailed Capabilities of Individual Configurations 24
  - 3.1.1 Peripherals 24
  - 3.1.2 IDLE Behavior in the Simulator 25
- 3.2 Configuring the Simulator 25
  - 3.2.1 Creating a Memory Map 26
- 3.3 Performance Numbers 26

List of Figures

2-1 Setup for Pipeline Latency Warning 18
2-2 Pipeline Latency Message 19

List of Tables

1-1 Processors Supported by the C54x Simulator 8
1-2 Configurations for Use With Unsupported C54x Devices 9
1-3 On-Chip Peripherals Supported by the C54x Simulator 10
2-1 Pins Supported by the Pin Connect Feature of the C54x Simulator 13
2-2 *hpi-function-specification* Parameter Values 15

3-1 Performance Numbers of the C54x Simulator 26

Read This First

About This Manual

This manual does the following:

- Names the TMS320C54x™ (C54x™) digital signal processors that are supported by configurations of the TMS320C54x Instruction Set Simulator
- Lists which modules and pin of each device are modeled
- Describes capabilities and limitations of the simulator
- Explains how to configure the simulator
- Provides some benchmarking data on cycle accuracy

Notational Conventions

This document uses the following conventions:

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

Related Documentation From Texas Instruments

TMS320C54x DSP CPU and Peripherals Reference Set, Volume 1 (literature number SPRU131) describes the TMS320C54x 16-bit fixed-point general-purpose digital signal processors. Covered are its architecture, internal register structure, data and program addressing, and the instruction pipeline.

TMS320C54x DSP Mnemonic Instruction Set Reference Set, Volume 2 (literature number SPRU172) describes theTMS320C54x DSP algebraic instructions individually. Also includes a summary of instruction set classes and cycles.

TMS320C54x DSP Algebraic Instruction Set Reference Set, Volume 3 (literature number SPRU179) describes theTMS320C54x DSP mnemonic instructions individually. Also includes a summary of instruction set classes and cycles.

TMS320C54x DSP Enhanced Peripherals Reference Set, Volume 5 (literature number SPRU302) describes the enhanced peripherals available on some TMS320C54x DSPs. Includes the multi-channel buffered serial ports (McBSPs), direct memory access (DMA) controller, interprocessor communications, and the HPI-8 and HPI-16 host port interfaces.

Trademarks

TEXAS INSTRUMENTS

Chapter 1

SPRU598B−July 2002−Revised October 2003

Introduction to the TMS320C54x Simulator

The TMS320C54x™ Instruction Set Simulator, available within the Code Composer Studio™ Integrated Development Environment (IDE) for TMS320C5000™, simulates the instruction set of many members of the C54x™ family of devices.

Topic **Page**

1.1 Features 8

1.2 Processors Supported by the C54x Simulator 8

1.3 Considerations for Choosing a Simulator 9

1.4 Supported Hardware Resources 9

1.1 Features

The TMS320C54x simulators support the following features:

- Included in Code Composer Studio IDE for TMS320C5000™
- TMS320C54x CPU full instruction set architecture execution
  - Support of all instructions for devices with extended program pages
  - Parallel instruction execution
  - Provision for warning in case of possible inconsistency due to pipeline latency of some combinations of instructions
- Configurable memory simulation
- Accurate cycle simulation
  - On-chip memory blocks
  - External memory blocks
  - Support for the enhanced external parallel interface (XIO2) on C5410, C5403, C5404, C5406 and C5407
- Port Connect
  - Supports external peripherals in I/O page
  - Attachment of files to MMRs, On-chip and external memory locations to read and write when those locations are accessed
- Pin Connect
  - Supports external event simulation
- Supports simulator analysis events
- RTDX™ support
  - Host-target and target-host communication
- DSP/BIOS™ real-time analysis support

1.2 Processors Supported by the C54x Simulator

Table 1-1 lists the processors supported, with the corresponding configuration to be selected under the Import Configuration menu of Code Composer Studio Setup. The default configuration at installation is the C55x Functional Simulator. Subsequently, as Code Composer Studio IDE starts up it will load the most recent configuration selected in Code Composer Studio Setup.

Table 1-1. Processors Supported by the C54x Simulator

PROCESSOR	CODE COMPOSER STUDIO IDE IMPORT CONFIGURATION
TMS320C541	C541 Device Simulator
TMS320C542	C542 Device Simulator
TMS320C543	C543 Device Simulator
TMS320C545	C545 Device Simulator
TMS320C545LP	C545lp Device Simulator(1)
TMS320C546	C546 Device Simulator
TMS320C548	C548 Device Simulator
TMS320C549	C549 Device Simulator
TMS320C5401	C5401 Device Simulator
TMS320C5402	C5402 Device Simulator
TMS320C5403	C5403 Device Simulator
TMS320C5404	C5404 Device Simulator
TMS320C5406	C5406 Device Simulator

(1) Functionally equivalent to the C545 Simulator.

Table 1-1. Processors Supported by the C54x Simulator (continued)

PROCESSOR	CODE COMPOSER STUDIO IDE IMPORT CONFIGURATION
TMS320C5407	C5407 Device Simulator
TMS320C5409	C5409 Device Simulator
TMS320C5410	C5410 Device Simulator
TMS320C5416	C5416 Device Simulator
TMS320C5420	C5420 Device Simulator

1.3 Considerations for Choosing a Simulator

Although simulators for other DSP platforms and families (e.g. C6000, C55x) have different types of configurations (functional, cycle-accurate, etc.) that provide tradeoffs between the functionality modeled, cycle accuracy of the simulation, and performance, simulators for the C54x family are of only one type. All C54x simulator configurations are cycle-accurate. The differences among the configurations are due to the differences in the actual hardware of the devices (DSPs) in TMS320C54x family. For example, the C5402 Device Simulator configuration is used to simulate the TMS320C5402 processor. The device simulator configurations model most of the peripherals of the devices. The peripherals modeled are cycle-accurate, as in the silicon. These simulators can be used to get an indication of the cycle behavior of the application. The details of the memory and peripherals modeled in different configurations are discussed in Section 1.4.2 and Section 1.4.3, respectively.

Although there are many DSP processors in the C54x family, only those listed in Table 1-1 are currently supported in the simulator. But, because of similarities among the processors in the C54x family, users developing code for an unsupported processor can select the configuration most similar in terms of the CPU and on-chip peripherals. Table 1-2 shows the configuration that should be selected for use with some of the unsupported devices.

Table 1-2. Configurations for Use With Unsupported C54x Devices

UNSUPPORTED HARDWARE DEVICE (PROCESSOR)	NEAREST SIMULATOR CONFIGURATION
TMS320C5409A	C5409 Device Simulator
TMS320C5410A	C5410 Device Simulator
TMS320C5421	C5420 Device Simulator
TMS320C5440	C5420 Device Simulator
TMS320C5441	C5420 Device Simulator

1.4 Supported Hardware Resources

The following sections provide a concise overview of the supported hardware resources for each of the simulator configurations. For more detailed information, see Chapter 2.

1.4.1 CPU

The C54x CPU core simulated is the same for all the supported devices. It includes simulation of the full instruction set architecture, including FAR and IDLE instructions. Pipeline latency warning messages are displayed if the pipeline latency feature is enabled through the Board Properties section of Code Composer Studio Setup. For more information on pipeline latencies, see the *TMS320C54x DSP CPU and Peripherals Reference Guide, Volume 1* (literature number SPRU131).

1.4.2 Memory

The simulator provides configurable memory simulation. For each device, the simulator provides all on-chip memory blocks of that device. It also, by default, provides external memory blocks in data space, program space for devices without extended program pages, and up to the first program page that does not have any on-chip program memory for devices with extended program pages. By default, the simulator does not provide external memory in I/O space, however, the rest of the external memory blocks in program and I/O space can be simulated by adding memory blocks using the Memory Map Add feature or by changing the appropriate configuration (.cfg) file.

The simulator provides cycle-accurate simulation for both on-chip and external memory blocks. For example, if there are two simultaneous accesses (PB access and DB access) to the same on-chip memory block, the simulator will take no extra cycle if the memory block is dual-access memory, but will take one extra cycle if the memory is single-access memory. For external memory, the simulator provides wait-state support (required for the slow external memory interface) according to the number of wait-states being set by the SWWSR, SWCR, and BSCR registers. The simulator also supports XIO2 as applicable to some of the devices.

1.4.3 Peripherals

Table 1-3 shows the on-chip peripherals supported under each simulator configuration. For more information on these peripherals, see the *TMS320C54x DSP CPU and Peripherals Reference Guide, Volume 1* (literature number SPRU131) and the *TMS320C54x DSP Enhanced Peripherals Reference Guide, Volume 5* (literature number SPRU302). Please refer to Chapter 3 for more details on the peripherals under different configurations.

Table 1-3. On-Chip Peripherals Supported by the C54x Simulator

SIMULATOR CONFIGURATION	PERIPHERALS SUPPORTED							
	TIMERS		SERIAL PORTS			DMA CONTROLLER	HPI	FIFO
C541	Timer0		SP0	SP1	–	–	–	–
C542	Timer0		BSP0	TDM	–	–	HPI	–
C543	Timer0		BSP0	TDM	–	–	–	–
C545	Timer0		BSP0	SP1	–	–	HPI	–
C546	Timer0		BSP0	SP1	–	–	–	–
C548	Timer0		BSP0	TDM	BSP1	–	HPI	–
C549	Timer0		BSP0	TDM	BSP1	–	–	–
C5401	Timer0		McBSP0	McBSP1	–	XDMA	HPI8	–
C5402	Timer0	Timer1	McBSP0	McBSP1	–	XDMA	HPI8	–
C5403	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI8	–
C5404	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI8	–
C5406	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI8	–
C5407	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI8	–
C5409	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI8	–
C5410	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI8	–
C5416	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI16	–
C5420	Timer0		McBSP0	McBSP1	McBSP2	XDMA	HPI16	FIFO

Supported Simulation Features

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

Topic **Page**

2.1 External Event & Data Simulation 12
2.2 Pipeline Latency Warning 17
2.3 Simulation Pipeline Modes 19
2.4 Simulator Analysis 21
2.5 RTDX 21
2.6 DSP/BIOS 21

2.1 External Event & Data Simulation

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

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

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

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

2.1.1 Pin Connect

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

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

The C54x simulator provides the Pin Connect feature for all processor configurations. The pins supported for different processor configurations are shown in Table 2-1.

Table 2-1. Pins Supported by the Pin Connect Feature of the C54x Simulator

CONFIG	PINS SUPPORTED														
	INTERRUPTS (PULSE TYPE)				BIO[1]	SERIAL PORT RELATED PINS (PULSE W/ CLOCK RATIO SPECIFIED)[2]									HPI PIN
C541	INT 0	INT 1	INT 2	INT 3	BIO	FSR0		FSX0		FSR1		FSX1			-
C542						BFSR		BFSX		TFSR		TFSX			HPI
C543						BFSR		BFSX		TFSR		TFSX			-
C545						BFSR		BFSX		FSR		FSX			HPI
C546						BFSR		BFSX		FSR		FSX			-
C548						BFSR0	BFSX0	-	TFSR	TFSX	-	BFSR1	BFSX1	-	HPI
C549						BFSR0	BFSX0	-	TFSR	TFSX	-	BFSR1	BFSX1	-	-
C5401						McBFSR0	McBFSX0	-	McBFSR1	McBFSX1	-	McBFSR2	McBFSX2	-	HPI
C5402						McBFSR0	McBFSX0	-	McBFSR1	McBFSX1	-	McBFSR2	McBFSX2	-	HPI
C5403						McBFSR0	McBFSX0	CLKS0	McBFSR1	McBFSX1	CLKS1	McBFSR2	McBFSX2	CLKS2	HPI
C5404						McBFSR0	McBFSX0	CLKS0	McBFSR1	McBFSX1	CLKS1	McBFSR2	McBFSX2	CLKS2	HPI
C5406						McBFSR0	McBFSX0	CLKS0	McBFSR1	McBFSX1	CLKS1	McBFSR2	McBFSX2	CLKS2	HPI
C5407						McBFSR0	McBFSX0	CLKS0	McBFSR1	McBFSX1	CLKS1	McBFSR2	McBFSX2	CLKS2	HPI
C5409						McBFSR0	McBFSX0	CLKS0	McBFSR1	McBFSX1	CLKS1	McBFSR2	McBFSX2	CLKS2	HPI
C5410						McBFSR0	McBFSX0	CLKS0	McBFSR1	McBFSX1	CLKS1	McBFSR2	McBFSX2	CLKS2	HPI
C5416						McBFSR0	McBFSX0	-	McBFSR1	McBFSX1	-	McBFSR2	McBFSX2	-	HPI
C5420						McBFSR0	McBFSX0	CLKS0	McBFSR1	McBFSX1	CLKS1	McBFSR2	McBFSX2	CLKS2	HPI

(1) BIO (waveform type)
(2) In the case of serial ports, the CLKX pin functionality is combined with the functionality of the corresponding FSX pin, and only one logical pin (FSX) is provided. The same is true of the CLKR pin. In the pin-connect file of these FSX and FSR pins, information regarding CLKX (or CLKR), such as the clock ratio with respect to CLOCKOUT, is provided in addition to the information regarding external frame synchronization events.

The Pin Connect file can have one or more statements of the following formats for the different types of pins supported:

- Pulse type
- Waveform type
- Pulse with clock ratio specified type
- HPI type

2.1.1.1 Pulse Type

clock-cycle [**rpt** { *n* | **EOS** }]

The *clock-cycle* parameter represents the CPU clock cycle in which the interrupt should occur. The clock value can be specified in decimal or hexadecimal format (using the 0x or 0X prefix format or the h or H suffix format).

There can be two types of CPU clock cycles:

- **Absolute**. The *clock-cycle* value must represent the actual CPU clock cycle in which the interrupt should occur.

  For example:

  12 34 56

  Interrupts are simulated at the 12th, 34th, and 56th CPU clock Cycles.
- **Relative**. The *clock-cycle* value is relative to the time at which the last event occurred.

  For example:

  12 +34 55

  Three interrupts are simulated: at the 12th, 46th (12 + 34), and 55th CPU clock cycles. A plus sign (+) before a clock cycle adds that value to the total clock cycles preceding it. Both relative and absolute values can be mixed in input file as shown.

The **rpt** { *n* **| EOS** } parameter is optional and represents a repetition value.

Two forms of repetition in simulating interrupts can be used:

- **Repeat a fixed number of times**. Repeat a particular pattern a fixed number (n) 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, 15th (5 + 10), 35th (15 + 20), 45th (35 + 10), and 65th (45 + 20) CPU clock cycles. The parameter *n* is a positive integer value.
- **Repeat to the end of simulation**. To repeat the same pattern throughout the simulation, the string EOS should be added to the line.

  10 (+5 +20) rpt EOS

  Interrupts are simulated at the 10th, 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.

2.1.1.2 Waveform Type

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

The square brackets ([ ]) are required to encapsulate *clock-cycle* and its corresponding *logic-value*.

The *logic-value* is valid only for the BIO pin and this is the only difference between pulse type and waveform type. The signal can be forced to go high or low at specified clock cycles. A value of 1 forces the signal to go high, and a value of 0 forces the signal to go low. For example:

[12,1] [23,0] [45,1]

See Section 2.1.1.1 for details on the *clock-cycle* and **rpt** parameters.

2.1.1.3 Pulse with Clock Ratio Specified Type

DIVIDE *r*

clock-cycle [**rpt** { *n* **| EOS** }]

The difference in this format and the pulse type format (see Section 2.1.1.1) is the addition of the DIVIDE command to specify the divide-down ratio for the device clock. The parameter *r* is a real number or integer specifying the ratio of the CPU clock rate to the serial port clock rate. The divide ratio is used when the serial port is configured to use the external clock.

When the DIVIDE command is used, it must be the first command in the file. The following example specifies the clock ratio of the transmit clock and the clock cycles for the occurrence of TFSX pulses (if this file is connected to the TFSX pin):

```
DIVIDE 5
100 +200 +100
```

The DIVIDE command specifies the divide-down ratio of the clock against the CPU clock. That is, the CLKX frequency is 1/5 of the CPU clock. The second line indicates that the TFSX should go high at the 100th, 300th (100 + 200) and 400th (300 + 100) CPU cycles. The TFSX pin goes high in the 500th, 1500th, and 2000th cycles of the serial port clock.

See Section 2.1.1.1 for details on the *clock-cycle* and **rpt** parameters.

2.1.1.4 HPI Type

clock-cycle hpi-function-specification [**rpt** { *n* | **EOS** }]**:**

A colon is required as a delimiter after every statement, as shown in the syntax.

The *hpi-function-specification* parameter specifies the type of Host Port Interface (HPI) operation to be performed. Table 2-2 describes the different formats of the *hpi-function-specification* depending upon the type of HPI.

Table 2-2. *hpi-function-specification* Parameter Values

FORMAT	HPI TYPE	DESCRIPTION
CTL_READ LSB **CTL_READ MSB**	HPI, HPI8	Control register read, byte is determined by **MSB** or **LSB** as HPI/HPI8 can handle only one byte at a time
CTL_READ	HPI16	Control register read, HPI16 can handle one word at a time
CTL_WRITE*data***LSB** **CTL_WRITE** data **MSB**	HPI, HPI8	Control register write; *data* specifies the write value
CTL_WRITE *data*	HPI16	Control register write; *data* specifies the write value
DATA_READ LSB ++ **DATA_READ MSB ++**	HPI, HPI8	Data read with HPIA increment
DATA_READ LSB **DATA_READ MSB**	HPI, HPI8	Data read without HPIA increment
DATA_READ ++	HPI16	Data read with HPIA increment
DATA_READ	HPI16	Data read without HPIA increment
DATA_WRITE *data* **++ LSB** **DATA_WRITE** *data* **++ MSB**	HPI, HPI8	Data write with HPIA increment; *data* specifies the write value
DATA_WRITE *data* **LSB** **DATA_WRITE** *data* **MSB**	HPI, HPI8	Data write without HPIA increment; *data* specifies the write value
DATA_WRITE *data* **++**	HPI16	Data write with HPIA increment; *data* specifies the write value
DATA_WRITE *data*	HPI16	Data write without HPIA increment; *data* specifies the write value
LOAD *file-name*	HPI, HPI8, HPI16	Initialize HPI RAM with data from *file-name*
HPIA_WRITE *address-as-data* **LSB** **HPIA_WRITE** *address-as-data* **MSB**	HPI, HPI8	HPIA register write, byte is determined by **MSB** or **LSB** as HPI/HPI8 can handle only one byte at a time; *address-as-data* specifies the write value
HPIA_WRITE *address-as-data* **EXB**	HPI8	HPIA register write, EXB specifies that this is the extended seven (16..22) bits of address
HPIA_WRITE *address-as-data*	HPI16	HPIA register write, HPI16 can handle one word at a time, XHPIA bit in HPIC determines whether it is lower or extended bits

See Section 2.1.1.1 for details on the *clock-cycle* and **rpt** parameters.

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

2.1.2 Port Connect

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

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

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

To simulate a data transfer from an external entity to the DSP, first all data values which will be transferred from the external entity to the DSP are put into a file (the details of the file format are in Section 2.1.2.1). Then an association is made in the simulator between this file and the address at which the external entity sits. This association is called read-mode port-connect. Whenever the simulator must read from the external entity through the associated address, it reads from the file one word at a time. To simulate the data transfer from the DSP to an external entity, a file is port-connected in write mode against the address of the external entity. Whenever the simulator has to write data to that address, it writes in the file one word at a time.

Although only one bit at a time of a sample transfers through the serial line when HPI or HPI8 is specified (see Section 2.1.1.4), samples can be encapsulated in words as serial port reads from the Serial Port Receive Register (DRR) or writes in to the Serial Port Transmit Register (DXR). All the samples that are to be sent to the simulator are written in a file and this file is port-connected in read mode against the memory address corresponding to the DRR. Whenever the serial port of the simulator has to read one word from the DRR as part of the receive operation, it reads the word from the file. Similarly, to get the data serial port transmits through the serial line, a file is port-connected in write mode against the memory address corresponding to the DXR. Whenever the serial port of the simulator writes one word to the DXR as a part of the transmit operation, it also writes the word in to the file.

The C54x simulator provides the Port Connect feature for all processor configurations. Although Port Connect is primarily meant for the I/O memory and serial port registers, the same concept can be applied to program and data memory as well. The C54x simulators support Port Connect to the following memory locations:

- All I/O memory locations
- All data and program memories, except the reserved and memory mapped register (MMR) locations
- Only the serial-port receive and transmit register locations among the MMR locations. (See the data sheet corresponding to the particular configuration to get the address of the serial port receive and transport registers.)

Before using this feature, make sure the address being connected to is already mapped. This is important in case the address used is in I/O or higher extended external pages, as the simulator by default does not include those addresses in its memory map.

2.1.2.1 Port Connect File Format

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

The following example is a sample Port Connect file:

```
6666
9999
aaaa
cccc
7f7f
```

Port Connect File Format Notes:

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

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

2.2 Pipeline Latency Warning

Since the C54x pipeline is unprotected an assembly programmer must be very careful in using multiple instructions that access CPU resources at the same time (C compiler handles the issue for C programmers). In particular, the following sentences in the CPU and peripheral user's guide point to potential problems: *"Some of these pipeline conflicts are resolved automatically by the C54x by delaying accesses. Other conflicts are unprotected and must be resolved by the programmer."*

The simulator provides assistance for the detection of pipeline conflicts on the C54x. It gives warning messages whenever pipeline conflicts occur during the execution of an application's assembly code. Although the C54x assembler provides similar features for the detection of pipeline conflicts, the differences between this feature in the simulator and that provided in the assembler are as follows:

- Since the simulator does run-time analysis, it can tell when pipeline conflicts occur - as opposed to the assembler, which must report potential pipeline conflicts at the time of assembling the code.
- The simulator can only detect pipeline conflicts in code that is actually executed in the simulator. The assembler can report potential pipeline conflicts in all the code it processes, with the exception of code surrounding branches. The assembler detects pipeline conflicts in straight-line code only.
- The potential pipeline conflicts detected by the assembler remains the same as long as the assembly code does not change, but the pipeline conflicts detected by the simulator may change from execution to execution depending on the input data, interrupts, and the dynamics of the real-time execution.

The following pipeline latencies are handled in the simulator:

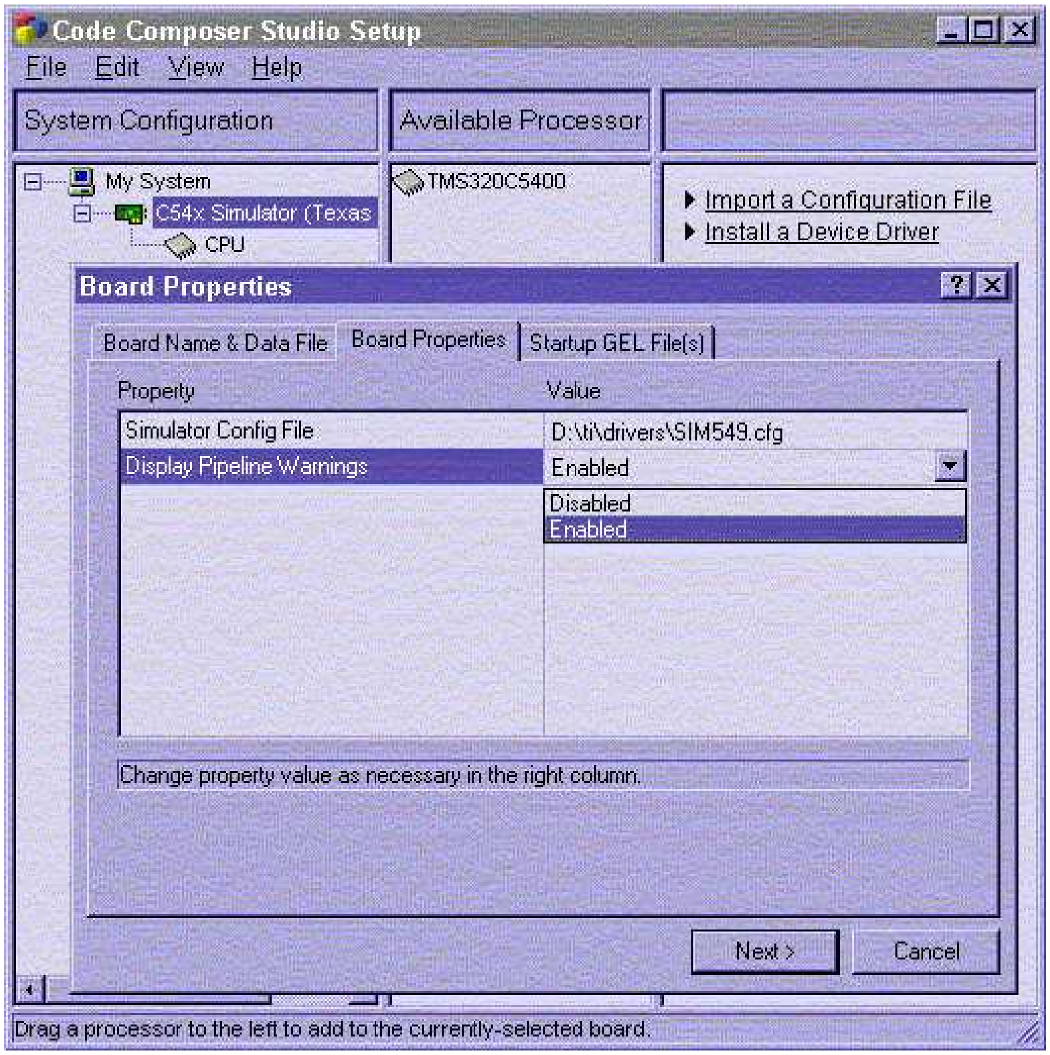
- DAGEN registers latency
  - ARx latency
  - BK latency
- Stack pointer latency
  - As an offset in direct addressing (CPL = 1)
  - Stack operation (CPL = 0)
- TREG latency

- PMST latency
  - OVLY/MPMC/DROM/IPTR latency
- Status registers (ST0/ST1) latency
  - ARP/CMPT/CPL/DP/SXM/ASM/BRAF latency
- BRC register latency
- MMR access of accumulators latency

To use the pipeline latency warning feature in the simulator, the pipeline latency warning option must be enabled during setup using the following procedure:

1. Open Code Composer Studio Setup. Import the configuration to be used.
2. Select **C54x Simulator** under My System in the **System Configuration** pane.
3. Right-click and open the properties window.
4. Go to the Board Properties tab.
5. Select **Enabled** from the list box corresponding to **Display Pipeline Warnings**. (See Figure 2-1)
6. Save this setup and start Code Composer Studio IDE.

Figure 2-1. Setup for Pipeline Latency Warning



Property	Value
Simulator Config File	D:\ti\drivers\SIM549.cfg
Display Pipeline Warnings	Enabled

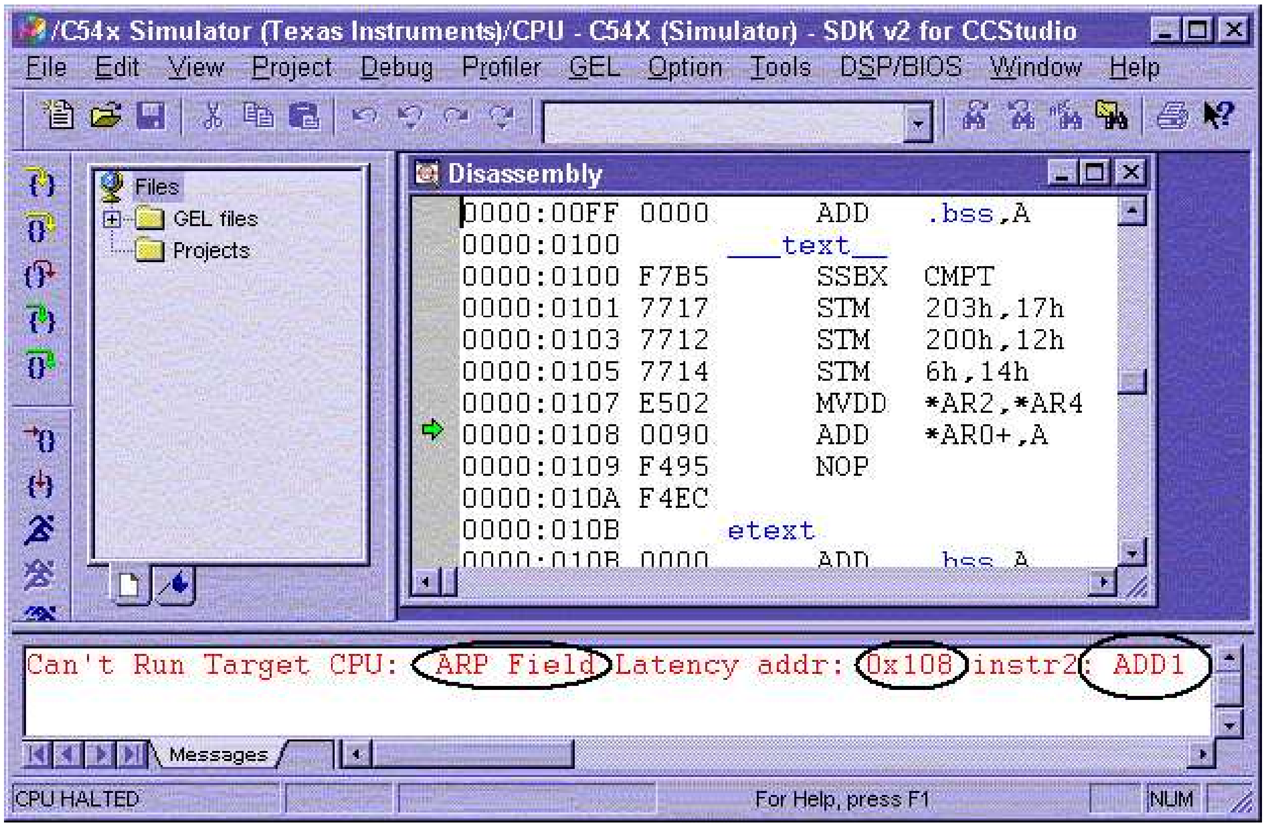
Disabled
Enabled

Change property value as necessary in the right column.

Drag a processor to the left to add to the currently-selected board.

Whenever the simulator detects the latency conflict within the application being run it will give a warning message. This message will contain the register bit/field name associated with the conflict, the instruction causing the latency conflict in an instruction pair and the program address of the causing instruction. (See Figure 2-2.)

Figure 2-2. Pipeline Latency Message


/C54x Simulator (Texas Instruments)/CPU - C54X (Simulator) - SDK v2 for CCStudio
File Edit View Project Debug Profiler GEL Option Tools DSP/BIOS Window Help
Files
GEL files
Projects
Disassembly
Messages
CPU HALTED
For Help, press F1
NUM

```
0000:00FF 0000       ADD    .bss,A
0000:0100        ___text__
0000:0100 F7B5       SSBX   CMPT
0000:0101 7717       STM    203h,17h
0000:0103 7712       STM    200h,12h
0000:0105 7714       STM    6h,14h
0000:0107 E502       MVDD   *AR2,*AR4
0000:0108 0090       ADD    *AR0+,A
0000:0109 F495       NOP
0000:010A F4EC
0000:010B        etext
0000:010B 0000       ADD    .bss,A
```

```
Can't Run Target CPU: ARP Field Latency addr: 0x108 instr2: ADD1
```

2.3 Simulation Pipeline Modes

All C54x simulators support two modes of execution called *Pipeline Flush* mode and *Pipelined* mode.

- In **Pipeline Flush mode**, when the simulation is halted, the execution pipeline is flushed. The watch window shows the correct values of local variables in this mode. Profile data may be skewed because of flushing the pipeline whenever the simulation halts.

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

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

2.3.1 Switching Between the Modes

Pipeline Flush mode is the default simulation mode.

The mode can be changed using either of the following mechanisms:

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

2.3.2 Effects of Mode Switching

From Pipeline Flush mode to Pipelined mode:

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

From Pipelined mode to Pipeline Flush mode:

- Simulation flushes the pipeline.
- After mode switching, the PC points to the instruction in the *decode* phase of the pipeline.
- Breakpoints that might have been set at instructions which are being flushed out of the pipeline are ignored.

2.3.3 Recommendations on Simulation Mode

The table below summarizes the Simulation mode recommendations while using various features:

Pipelined Mode	Use this mode for following features: • Profiler • Simulator analysis
Pipeline Flush Mode	Use this mode for following features: • Watch window • Register window • Memory window

2.3.4 Related Warnings

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

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

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

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

2.4 Simulator Analysis

The C54x analysis module gives a detailed look into events occurring in the hardware, expanding debugging capabilities beyond software breakpoints. The analysis module examines C54x bus cycle information in real time and reacts to this information through actions, such as data breakpoints. The simulator provides monitoring of the following types of accesses during a debugging session on all processor configurations:

- Read access on program or data memory
- Write access on program or data memory
- Read and write access on program or data memory
- Read and/or write access for a particular data value
- Read and/or write access for a particular data pattern
- Instruction fetch on the program bus
- Data breakpoint between two instruction addresses

For more information, see the Code Composer Studio C54x Simulator Analysis plug-in online help.

2.5 RTDX

Real-Time Data Exchange (RTDX) is supported when running inside the simulator. To run RTDX inside the simulator, the user must link applications with the RTDX Simulator Target library. All the simulator device configurations support RTDX. RTDX support includes both host-target and target-host communication.

2.6 DSP/BIOS

All applications using the DSP/BIOS can be run on all the C54x simulators. In order to enable Real Time Analysis (RTA) for these applications, the user needs to ensure that the RTDX Mode in the configuration is set to simulator. Please refer to the Code Composer Studio online documentation on DSP/BIOS for more details on RTA and how to configure the simulator target.

TEXAS INSTRUMENTS

Chapter 3

SPRU598B – July 2002 – Revised October 2003

Configuration Specifics

This chapter describes the capabilities of the simulator configuration as well as how to configure the simulator. Performance numbers for the simulator are also included.

Topic **Page**

3.1 Detailed Capabilities of Individual Configurations 24
3.2 Configuring the Simulator 25
3.3 Performance Numbers 26

3.1 Detailed Capabilities of Individual Configurations

The capabilites and known limitations of the simulator configuration are described below.

3.1.1 Peripherals

Modeled peripherals vary by device. The differences are detailed in the following sections.

3.1.1.1 Timer

(Timer0, Timer1)

- Timer0 is supported on all devices
- Timer1 is supported only on the C5402. It is not supported on the other devices (C5403, C5404, C5406, and C5407), which have Timer1 on real hardware.

3.1.1.2 Serial Ports

(SP0, SP1, BSP0, BSP1, TDM, McBSP0, McBSP1, McBSP2)

- The C54x simulator supports simple serial port transmission and reception by reading data from, and writing data to, the files associated (port-connected) with the Data Transmit Register and Data Receive Register, respectively.
- The simulator provides limited support for the simulation of the serial port control pins (frame synchronization pins) with the help of external event simulation capability. Frame synchronization signals for receive and transmit operations at various instants of time are fed through the files associated with the pins. (See Section 2.1.1, *Pin Connect*, for file format details.)
- There are differences between the C5401 and C5402 McBSPs in real hardware, but in the C54x simulator, the McBSPs of both devices have the functionality of the C5402 McBSP real hardware.
- The McBSPs in the C5403, C5404, C5406, C5407 and C5410 simulators are equivalent to the C5410 McBSPs in real hardware.
- McBSPs in other devices are equivalent to their respective hardware.

3.1.1.3 DMA

- The DMA in the C5401 and C5402 simulators are equivalent to the C5410 DMA in real hardware.
- The DMA in the C5403, C5404, C5406, C5407 and C5410 simulators are equivalent to the C5410 DMA in real hardware.
- In the C5416 simulator the DMA supports extended registers but does not support extended I/O and data pages in DMA memory map.
- DMAs in other devices are equivalent to respective hardware.

3.1.1.4 FIFO

(in C5420)

- In the C5420 simulator, only one CPU subsystem is supported. One end of the FIFO is connected to DMA and the other end is connected to host files Rfifo.dat and Wfifo.dat in the current working directory. Any attempt to write from one DMA to another DMA through FIFO will write in to Wfifo.dat. Conversely, any read from another DMA through FIFO will read from "Rfifo.dat". If the user wants to read through FIFO, he should create the file Rfifo.dat just before setting up the DMA. The names of these two files cannot be changed.

3.1.1.5 Host Port Interface

(HPI, HPI8, HPI16)

This simulation is performed using two files associated with HPI. The first file specifies the value of the control signals and the corresponding address and data values. This file is associated (pin-connected) with the HPI pin. (See Section 2.1.1, *Pin Connect*, for file format details.)

The other file is for storing the output values. The outputs generated by the HPI simulation are stored in the output file named "hpi.out" in the current working directory. The name of this file cannot be changed.

3.1.2 IDLE Behavior in the Simulator

Under all configurations, the C54x simulator provides only one kind of idle functionality for all IDLE instructions. When the simulator is in IDLE, the following occur:

- The CPU in the simulator does not do any activity.
- All peripherals inside the simulator are given clocks and, hence, can continue their activities.

The following occurrence will bring the simulator out of IDLE:

- As a part of the activities under the IDLE mode, any one of the peripherals generates an interrupt.
- Using the Pin Connect feature, the user sends an interrupt.

3.2 Configuring the Simulator

The simulator uses configuration files to provide the user the ability to configure the simulator. There is a default configuration file in the drivers subdirectory in the Code Composer Studio installation directory corresponding to each configuration listed in Table 1-1. The exact path of the configuration file can be obtained from the Board Properties window as shown in Figure 2-1.

A small part of the default configuration file used for C549 configuration is shown below:

```
MODULE C54X;
   CHIP C549;    //Processor Number
   MODULE C549;

     // Template for defining blocks of memory
     // MEMORY BLOCK_NAME;
     //   START < STARTING ADDRESS >;
     //   LENGTH < LENGTH OF BLOCK >;
     //   PAGE < IO = 2, DATA = 1, PROG = 0>;
     //   TYPE < DARAM/SARAM/ROM/WOM/RAM/EXRAM >;
     // END BLOCK_NAME;

     MEMORY MEM0;
        START    0x0000;
        LENGTH   0x0800;
        PAGE     1;
        TYPE     DARAM;
     END MEM0;
     ...
     ...
   END C549;
END C54X;
```

3.2.1 Creating a Memory Map

The default configuration files have memory maps for all internal memories and some of the external memories. Changes to internal memories can cause undefined behavior of the simulator. The user can only configure the external memory of a particular configuration. For example, the current default config file for the C549 configuration has the external memory maps up to extended program page three. The user can add another external program page as follows:

```
MEMORY MEM27;
    START     0x40000;
    LENGTH    0x8000;
    PAGE      0;
    TYPE      EXRAM;
END MEM27;

MEMORY MEM28;
    START     0x48000;
    LENGTH    0x8000;
    PAGE      0;
    TYPE      EXRAM;

 END MEM28;
```

3.3 Performance Numbers

Table 3-1 shows the performance numbers of the simulator for different device configurations. These numbers were gathered on a 1.7GHz Intel™Pentium™ 4 PC with 256MB of RAM. The application used for measurement in all three cases is the Reed-Solomon encoding and decoding application from a standard benchmarking suite.

Table 3-1. Performance Numbers of the C54x Simulator

SIMULATOR CONFIGURATION	SIMULATOR SPEED (IN KILOCYCLES/SECOND)
C541-C546	1216
C548-C549	1012
C5410-C5420	751