C55x
Instruction Set Simulator

User's Guide

Literature Number: SPRU517
June 2001

PRINTED WITH SOY INK

TEXAS INSTRUMENTS

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's 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, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright © 2001, Texas Instruments Incorporated

Preface

Read This First

About This Manual

This document is the user's guide for the TMS320C55x™ instruction set simulator, available within Code Composer Studio. 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 1 discusses two simulation drivers that are available to simulate C55x DSP CPU and subsystems. This chapter also describes the capabilities and limitations of each driver.

Chapter 2 contains information on how to change stack configuration and also the differences related to C54x-compatible mode.

Chapter 3 discusses the importing and custom memory configuration setups.

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 TMS320C55x devices and related support tools.

TMS320C55x Optimizing C/C++ Compiler User's Guide (literature number SPRU281) describes the C55x C/C++ compiler. This compiler accepts ANSI standard C/C++ source code and produces assembly language source code for the TMS320C55x device.

TMS320C55x Assembly Language Tools User's Guide (literature number SPRU280) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for TMS320C55x™ devices.

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

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

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

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

TMS320C55x and Code Composer Studio are trademarks of Texas Instruments.

If You Need Assistance. . .

If you want to. . .	Do this. . .
Request more information about Texas Instruments Digital Signal Processing (DSP) products	Call the CRC† hotline: **(800) 336–5236** Or write to: Texas Instruments Incorporated Market Communications Manager, MS 736 P.O. Box 1443 Houston, Texas 77251–1443
Order Texas Instruments documentation	Call the CRC† hotline: **(800) 336–5236**
Ask questions about product operation or report suspected problems	Call the DSP hotline: **(713) 274–2320**
Report mistakes in this document or any other TI documentation	Fill out and return the reader response card at the end of this book, or send your comments to: Texas Instruments Incorporated Technical Publications Manager, MS 702 P.O. Box 1443 Houston, Texas 77251–1443

† Texas Instruments Customer Response Center

Contents

Figures

Chapter 1

Simulator Features and Limitations

This chapter discusses two simulation drivers that are available to simulate the C55x CPU and its subsystems. This chapter also describes the capabilities and limitations of each driver.

Topic **Page**

1.1 C55x CPU and Memory System Simulation Capabilities

1.1.1 Functional Capabilities

The simulator's functional capabilities are listed below.

- C55x CPU full instruction set architecture execution (except emulation instructions and IDLE instruction). For more information, see the *TMS320C55x DSP CPU Reference Guide* and the *TMS320C55x DSP Instruction Set Reference Guides*.
- Parallel instruction execution
- Configurable memory system simulation
  - If the memory configuration is not provided, a flat memory system (memory with no latency, no DARAM/SARAM) is used as a default.
  - Program/data memory with latency is supported.
  - If the memory map is provided in a configuration file, the driver uses the cycle accurate memory system (SARAM/DARAM). Support of SARAM and DARAM memory models follow the C55x memory protocol and access priorities. To use the memory system, you must set up the configuration file accordingly (see Chapter 3).
  - If a hole exists in the memory map, access to an unmapped location generates a bus error, and it is flagged in 8th bit of IFR1 register (INT24).
- The estop_1 instruction can be used in your code as a software breakpoint in addition to simulator breakpoints.
- The Port Connect tool supports external peripheral simulation (in I/O memory). For more information on Port Connect, select Help–>Contents in Code Composer Studio. Port Connect online help is listed in the Contents pane.

  Port Connect is supported only for I/O accesses from 0x0 to 0xFFFF.

  However, two functional timer modules are simulated at I/O memory ranges 0x1000 to 0x13FF and 0x2400 to 0x27FF. Port connect is not supported within these ranges.
- The Pin Connect tool supports external interrupt simulation. The following interrupts pins are supported: NMI, SINT2, SINT3 … SINT24. For more information on Pin Connect, select Help–>Contents in Code Composer Studio. Pin Connect online help is listed in the Contents pane.

  Note that SIN24 and SINT22 are not supported because internal timers use them.

- The simulator driver includes I/O memory (a placeholder for peripherals) that supports word reads/writes. This functionality can be used for general access and storage.

  I/O memory is supported for I/O accesses from 0x0 to 0xFFFF.

  However, two functional timer modules are simulated at I/O memory ranges 0x1000 to 0x13FF and 0x2400 to 0x27FF. I/O memory accesses not supported within these ranges.

- When the CPU is writing to an I/O address, the simulator first checks if there is a file connected to that address. The write happens to the file and also in the I/O memory. During the reading of I/O space, the simulator first checks if there is a file connected to that address and reads from there. If there is no file, then the simulator reads from the I/O memory.

- CPU internal registers are visible in the Code Composer Studio Register window. For more information, see the Code Composer Studio online help.

1.1.2 Functional Timer Support

Timer support includes the following:

- Setting up timer registers
- Count down and generation of interrupts
- Timer 0 generates SINT4 and Timer 1 generates SINT22

For more information on the timers, see the *TMS320C55x DSP Peripherals Reference Guide*.

1.1.3 RTDX Support

RTDX™ support includes the following:

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

1.1.4 Limitations

The simulator has the following limitations:

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

1.2 C5510 Device Simulation Capabilities

The simulator supports several components as they are defined in the C5510 device specification. Capabilities and limitations of each module are listed in the following sections.

1.2.1 C55x CPU

For detailed information on the supported C55x CPU features, see Section 1.1.1, *Functional Capabilities*, on page 1-2.

1.2.2 Internal Memory Subsystem

The features and limitations are listed below:

- ❏ Internal memory interface supports interfacing with SARAM and DARAM models. Note that PDROM is not supported. By default, an SARAM bank of the same size is mapped to the ROM space.
- ❏ SARAM and DARAM memory models are supported according to the C55x memory protocol and access priorities. To use the memory system, you must set up the configuration file accordingly (see Chapter 3).
- ❏ Support of memory stalls due to slow program memory (memory with latency) and access conflicts in SARAM and DARAM. Refer to the C5510 configuration file setting to learn more about the C5510 memory map.
- ❏ If a hole exists in the memory map, access to an unmapped location generates a bus error and it is flagged in the 8^{th} bit of IFR1 register (INT24).
- ❏ Controlling of the memory map using the MPNMC bit is not supported.

1.2.3 External Memory Interface and Subsystem

For details on the EMIF features, see the *TMS320C55x DSP Peripherals Reference Guide*.

For simulation:

- ❏ Three types of memory are available that can be configured as external memory: asynchronous 32 bit, asynchronous 16 bit, and 32 bit SBSRAM.
- ❏ The type of memory attached is determined by the programming of CE (Chip Enable) space register in the EMIF. Keep the memory type as EXTERNAL in the configuration file (see Chapter 3).
- ❏ If a hole exists in the memory map, then the CPU access generates a bus error and it is flagged in 8^{th} bit of IFR1 register.

The limitations are listed below:

- ❏ Asynchronous 8 bit memory is not supported.
- ❏ SDRAM interface in EMIF is not supported.
- ❏ Posted write in EMIF is not supported.
- ❏ Minimum 2-cycle strobe period is needed for the asynchronous memory interface.

1.2.4 I-Cache (Instruction Cache)

For details on the I-cache features, see the *TMS320C55x DSP Peripherals Reference Guide*.

For simulation:

- ❏ To enable the I-cache, set bit 14 of ST3 to 1. Reset it to zero to disable the I-cache.
- ❏ By default, none of the I-cache banks are enabled.
- ❏ Program the I-CACHE registers to configure the I-cache and enable respective banks of the I-cache.

The limitations are listed below:

- ❏ I-cache flushing is not supported.
- ❏ The two-way I-cache bank is not supported.

1.2.5 DMA Controller

For details on the DMA Controller features, see the *TMS320C55x DSP Peripherals Reference Guide*.

The limitations are listed below:

- ❏ Burst transfer of 8 elements is not supported.

1.2.6 Peripheral Bus Controller

For details on the Peripheral Bus Controller features, see the *TMS320C55x DSP Peripherals Reference Guide.*

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

Peripheral bus controller	0x0000
External memory interface	0x0800
DMA configuration register	0x0C00
Timer registers	0x1000 0x2400
McBSP	0x2800 0x2C00 0x3000

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

The limitations are listed below:

- ❏ IDLE/Wakeup functionality is not supported.

1.2.7 Timer

For details on the timer features, see the *TMS320C55x DSP Peripherals Reference Guide.*

The limitations are listed below:

- ❏ Timer input/output pins are not supported.

1.2.8 Multi-channel Buffered Serial Ports

For details on the McBSP features, see the *TMS320C55x DSP Peripherals Reference Guide.*

For simulation:

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

The limitations are listed below:

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

1.2.9 Simulating Enhanced Host Port Interface

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

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

1.2.9.1 Setting Up the Input Command File

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

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

Commands use the following format:

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

- Each command must be on a new line and the line must not contain anything else.
- To specify comments, use a hash (#) as the first character in the line
- The *clock_cycle* parameter specifies the DSP clock cycle in which the host applies the request to EHPI.
- The *address* parameter represents a 16-bit address field for multiplexed mode and 20-bit address field for non-multiplexed mode.
- The *data* parameter represents a 16-bit data field.
- The *command* specifies the type of operation requested by the host, and can be any of the following commands shown in Table 1–1.

Table 1–1. Operation Commands

Commands for Non-Multiplexed Mode	**Syntax**
WRITEMEM (data write) is a command for the non-multiplexed mode of operation. This command writes the specified data word at the specified address.	WRITEMEM clock_cycle address data
READMEM (data read) is a command for the non-multiplexed mode of operation. This command reads the data word at the specified address.	READMEM clock_cycle address
Commands for Multiplexed Mode	**Syntax**
WRITEHPIA (hpia write) is a command for the multiplexed mode of operation. This command writes the specified address to the HPIA register of EHPI.	WRITEHPIA clock_cycle address
WRITEHPID (hpid write) is a command for the multiplexed mode of operation. This command writes specified data at the address specified in the HPIA register.	WRITEHPID clock_cycle data
WRITEHPIDAUTOINC (hpid write autoinc) is a command for the multiplexed mode of operation. This command writes specified data at the address specified in the HPIA register. HPIA is then post incremented.	WRITEHPIDAUTOINC clock_cycle data
READHPID (hpid read) is a command for the multiplexed mode of operation. This command reads data at the address specified in the HPIA register.	READHPID clock_cycle
READHPIDAUTOINC (hpid read autoinc) is a command for the multiplexed mode of operation. This command reads data at the address specified in the HPIA register. HPIA is post incremented.	READHPIDAUTOINC clock_cycle
READHPIC (hpic read) is a command for the multiplexed mode of operation. This command reads value from the HPIC register.	READHPIC clock_cycle
WRITEHPIC (hpic write) is a command for the multiplexed mode of operation. This command writes specified data value to the HPIC register.	WRITEHPIC clock_cycle data

Figure 1–1 and Figure 1–2 show sample input file examples.

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

```
{
  WriteMem 10 0x11020 0x1234;
  ReadMem 25 0x11020;
  WriteMem 40 0x11021 0x4321;
  ReadMem 55 0x11021;
  WriteMem 70 0x11022 0xabcd;
  ReadMem 85 0x11022;
  WriteMem 100 0x11023 0xbcda;
  ReadMem 125 0x11023;
  WriteMem 140 0x11024 0x5612;
  WriteMem 160 0x11025 0x64cd;
  WriteMem 180 0x11026 0xac72;
  ReadMem 215 0x11024;
  ReadMem 230 0x11025;
  ReadMem 245 0x11026;
}
```

Figure 1–2. Example Input File for Multiplex Mode

```
{
  WriteHpia 10 0x11020;
  WriteHpidAutoinc 25 0xabcd;
  WriteHpidAutoinc 40 0x1342;
  WriteHpidAutoinc 60 0x6ca3;
  WriteHpid 80 0x15b2;
  WriteHpia 100 0x11020;
  ReadHpidAutoinc 120;
  ReadHpidAutoinc 160;
  ReadHpidAutoinc 200;
}
```

1.2.9.2 Connecting the Input Command File to the Interrupt Pin

To connect your input file to the interrupt pin, you can either use the Pin Connect tool or the Command Window tool in Code Composer Studio.

To use the Pin Connect tool:

1) From the Tools menu, select Pin Connect.

2) From the list of available pins, select HPI.

3) Connect the file.

To use the Command Window tool:

1) From the Tools menu, select Command Window.

2) In the Command Window, enter the following command:

```
pinc HPI, filename
```

1.2.9.3 Limitations

The limitations are listed below:

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

1.2.10 Modules Not Supported

The following modules are not supported:

- ❏ ROM model
- ❏ GP I/Os
- ❏ Clock PLL
- ❏ Hardware accelerator modules

1.2.11 Other Limitations

The simulator has the following limitations:

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

1.2.12 RTDX Support

RTDX support includes the following:

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

1.3 Pipeline Effect on Blue-Bar Movement

The C55x DSP has the following pipeline stages:

- Decode
- Address
- Access1
- Access2
- Read
- Execute
- Memory Write

The memory write phase is activated only for those instructions where memory writes or memory-mapped register writes occur. In Code Composer Studio's Disassembly window, the PC (blue-bar/arrow) indicates the instruction about to be executed. This is the instruction at the end of the Read phase of the pipeline. A step command at this point would execute the instruction at the blue bar and continue until the end of the Read phase of the next instruction in the pipeline.

Some C55x instructions complete their operation in the Address phase of the pipeline. These instructions include those that modify address registers, load repeat counters, etc. When the PC indicator reaches one of these instructions, the results are already available. Thus, the new value of the modified registers can be seen in Code Composer Studio's CPU Register window.

In most cases, the PC indicator moves one instruction at a time with every step command. However, the following cases are exceptions:

- A memory write instruction followed by a non-memory instruction

  In this case, the first instruction (the memory write) finishes operation in the Memory Write phase while the second instruction (a non-memory instruction) finishes operation in the Execute phase. Since they are pipelined, both instructions effectively finish operation at the same clock cycle. If the PC indicator is at the first instruction, one step command will finish operation of both instructions. The PC indicator will then jump to the third instruction in the Disassembly window. This may be a little confusing. However, if necessary, you can set a breakpoint in the second instruction and stop just before the execution.

- PC discontinuity instruction

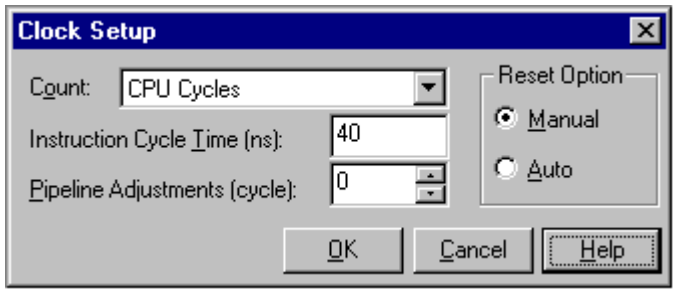
  Since PC discontinuity instructions have different operation latencies, the PC indicator might skip one instruction and jump to another.

1.4 Using the Profiler to Measure Stall Cycles

The Code Composer Studio profiler can be used to profile the following events in the simulator:

- Number of clock cycles executed in a range or a function (as selected)
- Number of pipeline stall cycles due to register or memory conflicts occurring in a range or a function (as selected)
- Number of pipeline stall cycles due to slow data memory operations occurring in a range or a function (as selected)
- Number of pipeline stall cycles due to a pre-fetch operation in a range or a function (as selected)

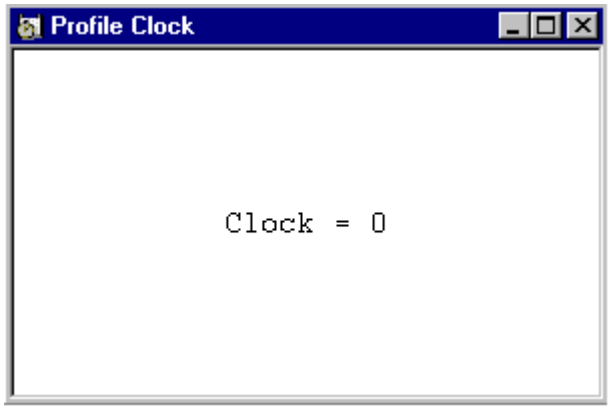
To profile events, open the Clock Setup window. From the Profiler menu, select Clock Setup.



In the Clock Setup window, in the Count field, select one of the four options:

- CPU cycles
- Pipeline Protection stalls
- Memory stalls
- Pre-fetch stalls

After completing the clock setup, you must enable and view the clock. From the Profiler menu, select View Clock. The clock window appears.



Now when you step or run through the program code, the clock count will indicate the number of cycles selected in the clock setup process.

1.5 Pipeline Stall Summary Report

The C55x simulator supports a pipeline stall summary report, which indicates number of pipeline stalls in the whole application or in the code that has been executed. This report is a text file with the name C55x_stall.report. It is created in the driver directory. An example of the stall report is shown in Figure 1–3.

Figure 1–3. Pipeline Stall Summary Report Example

```
################################################################################
# Pipeline stalls report
################################################################################
# Address  Instruction         PPU      Memory      Memory        PF/PPU   Total
                               Stalls  Read stalls  Write stalls  Stalls  Stalls
################################################################################
 0x23456  AMOV #234,AR1          3         0           0             0        3
 0xa152f OR #16640,mmap(ST1_55)  2         0           0             0        2
 0x76878  MOV *AR2+,AR0          4         2           0             0        6
 0x98764  MOV AR0,*AR2           0         0           2             3        5
################################################################################
# Total stall cycles             9         2           2             3        16
```

In the example above, instruction MOV *AR2+,AR0 stalls for 4 cycles fo register conflicts and then stalls for 2 cycles for reading the memory location pointed to by *AR2. Instruction MOV AR0, *AR2 stalls in the decode phase for 3 cycles due to pre-fetch/PPU and then stalls for 2 cycles due to memory write.

Note that the PF/PPU stall is really the sum of the decode stalls due to pipeline protection stalls in the execution pipeline and the decode stalls due to pre-fetch. So, in some cases, an instruction can be detected and traced. However, in other cases, if the instruction has not been pre-decoded, it will not be possible to show the name of the instruction while tracing a decode stall. The total indicates the total stall (dead) cycles in the execution of the program.

Chapter 2

General Tips for Simulation

This chapter contains information on how to change the stack configuration. It also describes the simulator functionality differences in C54x-compatible mode.

Topic **Page**

2.1 Changing Stack Configuration

C55x supports the following three modes of stack operation:

- ❏ 2x16 bit memory + register stack (fast return through RETA)
- ❏ 2x16 bit memory stack (slow return via memory)
- ❏ 32-bit memory stack (slow return via memory)

To change the stack mode, you must modify the configuration register, which resides at the first 4 bits of the reset vector. At reset, the C55x ignores the first 8 bits of the reset vector, pushes the lower 24 bits to the program counter, and executes. By default, the stack mode is the 32-bit stack with slow return.

To change the stack mode, perform the following steps in the debugger:

1) Load the program with the default stack configuration.
2) Modify the first 4 bits of the reset vector residing at 0xFFFF00 (program space) to the desired value. For example, to change from 32-bit stack mode to 16-bit register and memory stack mode, change the first 4 bits of the reset vector from 0110 (default) to 0000 (fast return with RETA).
3) Perform a reset through the debugger.
4) Step into the program and start executing. The new stack configuration will be in effect from this point on.

   You must step into the program after reset. If you perform a restart immediately after reset instead of stepping into the program, the new stack configuration will not be in effect. The restart forces the PC to the start address without decoding the delay slot instruction.

Alternatively, you can use a memory store instruction to modify the first 4 bits of the reset vector and execute a software reset instruction. This will have the same effect as described above.

For more information on the different stack modes, please refer to the *TMS320C55x DSP CPU Reference Guide*.

2.2 C54x-Compatible Mode Operation

The C55x DSP simulator behaves in C54x-compatible mode (the reset value of the C54CM bit is 1). Because the simulator operates in this mode, you should be aware of the following functionality differences:

- Indirect addressing
  - Indirect addressing uses the AR*x* register in place of DR*x* register for *(AR*x* +/– DR*x*) expressions.
  - Circular addressing always uses the BK03 for block-size calculation.
- Repeat loop
  - In C54x-compatible mode, the simulator supports only 1 level of hardware repeat loops. In the case of nested repeats, BRC0/RSA0/REA0 registers will be used even for inner loops. Therefore, you must save and restore these registers before the start of nested loops.
  - You can terminate or activate blockrepeat (and localrepeat) by setting the BRAF bit (ST1_55 register, bit #15) to 0 or 1 via a bit clear or set instruction. The BRAF bit is only visible in C54x-compatible mode.
- ASM compatibility
  - The lower 5 bits of the DR2 register are mirrored in the lower 5 bits of the ST1_55 register.
- far() qualifier
  - The use of the far() qualifier with call ACx and goto ACx instructions enables the use of only the 16-bit user stack (similar to C54x). This qualifier is activated only in C54x-compatible mode and will be available for the ported code.

Chapter 3

Simulator Configuration File Setup

This chapter discusses the syntax of the C55x simulator configuration file and how it can be used to configure memory subsystems.

Topic **Page**

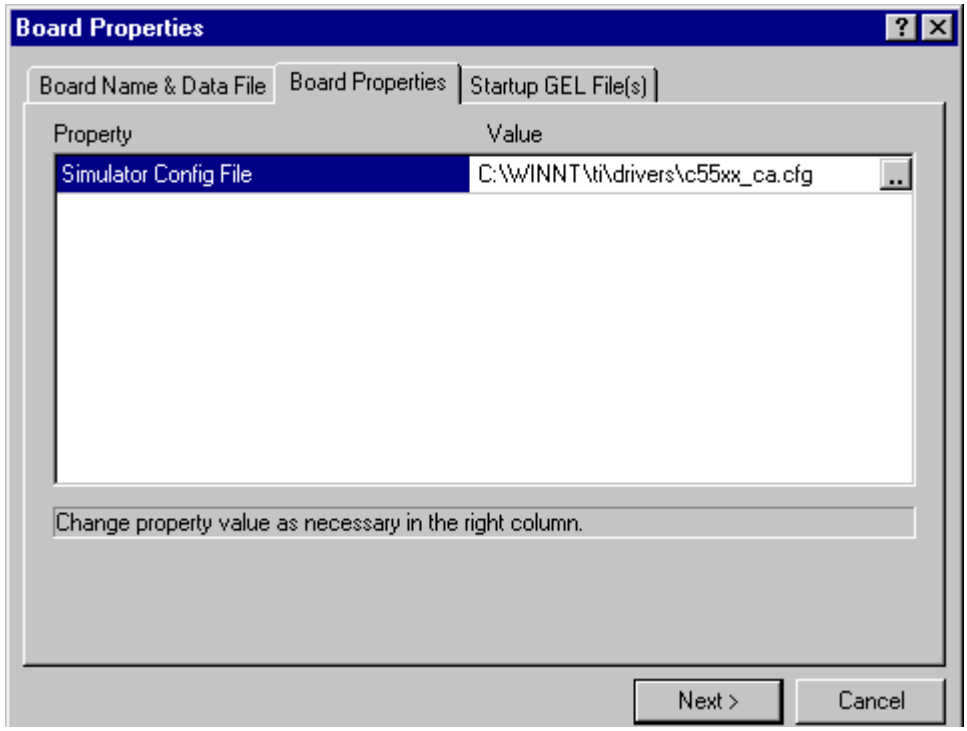
3.1 Specifying a Simulator Configuration

Code Composer Studio's default simulator configuration is the C55x CPU-only simulator. It is not necessary to run CCS Setup to use the default simulator configuration.

To use the C5510 simulator configuration, open the CCS Setup tool. In the Import Configuration dialog, in the Available Configurations field, select C5510 Simulator. Click the Import button to add this simulator to the System Configuration.

To use a modified simulator configuration file, specify the file in the Board Properties dialog of CCS Setup:

1) In the Available Board/Simulator Types pane, select the simulator that represents your system.
2) Double-click on the simulator device driver in the Available Board/ Simulator Types pane.
3) Click the Board Properties tab. Use the Browse button to locate and specify the simulator configuration file.



4) After specifying the simulator configuration file, click Next. On the Startup Gel File(s) page, click Finish. Save your setup. This file will be used for simulation. Note that this simulator configuration file is loaded every time you perform a reset operation.

3.2 Creating a Memory Map

This section describes the syntax of the C55x simulator configuration file and how it can be used to configure memory subsystems.

The memory map can be specified in the simulator configuration file. The following types of memory are supported:

- ❏ SARAM (Single Access RAM). Only one read/write can be done per cycle.
- ❏ DARAM (Dual Access RAM). Two reads/writes can be done per cycle.
- ❏ EXTERNAL. External memory is handled through EMIF. (Only available in the C5510 simulator.)

The memory map syntax is described below:

```
MEMORY_MAP
{
  #Bank Type  BankName  Start Address  Bank Size  Page  Latency
  ( type        name        addr         size      page   latency )
  (    ...    )
}
```

type names the type of memory bank. It can be one of three: SARAM, DARAM, or EXTERNAL.

name is a user-defined tag used to distinguish different memory banks of the same type. For example, SARAM4, ASYNC32, etc.

addr specifies the starting address (in hex) of the memory bank in the C55x memory address range (0x000000 to 0xFFFFFF).

size specifies the size (in hex) of the memory bank. The size can be from 0x0 to 0xFFFFFF, depending on the start address.

page names the memory page. This field is currently unused and should contain a zero (0).

latency specifies the number of wait states. This field can contain 0 or 1. A latency of 1 signifies that the instruction having a read request in cycle *n* will receive its data in cycle *n*+3. A latency of 0 signifies that the instruction having a read request in cycle *n* will receive its data in cycle *n*+2.

An example memory map is shown in Figure 3–1.

Figure 3–1. Example Memory Map

```
MEMORY_MAP
{
#Bank Type       BankName  Start Address Bank Size    Page   Latency
(SARAM           SARAM0    0x0             0x4000       0      0)
(SARAM           SARAM1    0x4000          0x4000       0      0)
(DARAM           DARAM1    0x8000          0x4000       0      0)
(DARAM           DARAM2    0xc000          0x4000       0      0)
(SARAM           SARAM2    0x10000         0xfe0000     0      1)
(EXTRENAL        SARAM     0xff0000        0x100ff      0      0)
}
```

The C5510 has a fixed memory map for the internal memory. Only the external memory can be configured.

The internal memory consists of 8 banks of DARAM (8KB each) and 32 banks of SARAM (8KB each)

Note that since no ROM model is available, the configuration uses a SARAM in place of ROM so that the reset vector is mapped into internal memory.

An example of the C5510 configuration file is shown in Figure 3–2.

Figure 3–2. Example C5510 Simulator Configuration File

```
MEMORY_MAP
{
#Bank Type    BankName  Start Address Bank Size     Page   Latency
(DARAM        DARAM0    0x000000         0x2000        0      0)
(DARAM        DARAM1    0x002000         0x2000        0      0)
(DARAM        DARAM2    0x004000         0x2000        0      0)
(DARAM        DARAM3    0x006000         0x2000        0      0)
(DARAM        DARAM4    0x008000         0x2000        0      0)
(DARAM        DARAM5    0x00a000         0x2000        0      0)
(DARAM        DARAM6    0x00c000         0x2000        0      0)
(DARAM        DARAM7    0x00e000         0x2000        0      0)
(SARAM        SARAM0    0x010000         0x2000        0      0)
(SARAM        SARAM1    0x012000         0x2000        0      0)
(SARAM        SARAM2    0x014000         0x2000        0      0)
(SARAM        SARAM3    0x016000         0x2000        0      0)
(SARAM        SARAM4    0x018000         0x2000        0      0)
(SARAM        SARAM5    0x01a000         0x2000        0      0)
(SARAM        SARAM6    0x01c000         0x2000        0      0)
(SARAM        SARAM7    0x01e000         0x2000        0      0)
(SARAM        SARAM8    0x020000         0x2000        0      0)
(SARAM        SARAM9    0x022000         0x2000        0      0)
(SARAM        SARAM10   0x024000         0x2000        0      0)
(SARAM        SARAM11   0x026000         0x2000        0      0)
(SARAM        SARAM12   0x028000         0x2000        0      0)
(SARAM        SARAM13   0x02a000         0x2000        0      0)
(SARAM        SARAM14   0x02c000         0x2000        0      0)
(SARAM        SARAM15   0x02e000         0x2000        0      0)
(SARAM        SARAM16   0x030000         0x2000        0      0)
(SARAM        SARAM17   0x032000         0x2000        0      0)
(SARAM        SARAM18   0x034000         0x2000        0      0)
(SARAM        SARAM19   0x036000         0x2000        0      0)
(SARAM        SARAM20   0x038000         0x2000        0      0)
(SARAM        SARAM21   0x03a000         0x2000        0      0)
(SARAM        SARAM22   0x03c000         0x2000        0      0)
(SARAM        SARAM23   0x03e000         0x2000        0      0)
(SARAM        SARAM24   0x040000         0x2000        0      0)
(SARAM        SARAM25   0x042000         0x2000        0      0)
(SARAM        SARAM26   0x044000         0x2000        0      0)
(SARAM        SARAM27   0x046000         0x2000        0      0)
(SARAM        SARAM28   0x048000         0x2000        0      0)
(SARAM        SARAM29   0x04a000         0x2000        0      0)
(SARAM        SARAM30   0x04c000         0x2000        0      0)
(SARAM        SARAM31   0x04e000         0x2000        0      0)
(EXTERNAL     ASYNC321  0x050000         0x3b0000      0      0)
(EXTERNAL     ASYNC322  0x400000         0x400000      0      0)
(EXTERNAL     ASYNC323  0x800000         0x400000      0      0)
(EXTERNAL     ASYNC324  0xc00000         0x3f8000      0      0)
(SARAM        PDROM     0xff8000         0x8000        0      0)
}
```

3.3 Limitations of Memory System Configuration

- For the C5510, it is recommended that you do not change the memory map.
- For the C5510, the external memory latency is obtained by the setup-strobe-hold register values in EMIF internal registers. The latency field in the configuration file is not used to delay the memory accesses.
- The EXTERNAL keyword is not supported for the C55x DSP CPU-only driver.
- For double access (32 bit long accesses), both words must reside in the same memory bank.
- To introduce latency in external memory in the C55x DSP CPU-only driver, use SARAM memories with latency values.

Index

I

L

M

P

R

S

T

W