TMS320C6416 Seamless CVE Model

User's Guide

Literature Number: SPRU548B
May 2002

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 contains the following chapters:

Chapter one describes some of the features of the TMS320C6416 Seamless CVE model, including features supported, device pins, accuracy and known limitations.

The second chapter takes you through the tutorials provided with the Seamless CVE model.

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:

  .asect "*section name***",** *address*

.asect is the directive. This directive has two parameters, indicated by *section name* and *address*. When you use .asect, the first parameter must be an actual section name, enclosed in double quotes; the second parameter must be an address.

- 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:

  LALK *16–bit constant [, shift]*

  The LALK instruction has two parameters. The first parameter, *16-bit constant*, is required. The second parameter, *shift*, is optional. As this syntax shows, if you use the optional second parameter, you must precede it with a comma.

  Square brackets are also used as part of the pathname specification for VMS pathnames; in this case, the brackets are actually part of the pathname (they are not optional).

- 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.

Information About Cautions and Warnings

This book may contain cautions and warnings.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Trademarks

TI is a trademark of Texas Instruments Incorporated.

Microsoft is a registered trademark of Microsoft Corporation.

Windows and Windows NT are registered trademarks of Microsoft Corporation.

Seamless CVE is a trademark of Mentor Graphics Corporation.

Contents

Chapter 1

Features of the TMS320C6416 Seamless CVE

This chapter describes the processor support package, methods for invoking, supported devices, simulation features, simulation accuracy, and known limitations of the TMS320C6416™ Seamless CVE model.

Topic **Page**

1.1 Processor Support Package

The TMS320C6416™ Seamless model simulates most of the pins of the TMS320C6416 device accurately. The TMS320C6416 Processor Support Package contains the Seamless model and all associated files required to run in a Mentor Seamless CVE environment.

Code Composer Studio™ IDE, the debugger interface for the TMS320C6416 ISS, is not shipped as part of this package. Please contact Texas Instruments to obtain a copy.

1.1.1 Contents

The package contains the following

- ❏ TMS320C6416 Seamless Model
- ❏ TMS320C6416 Seamless CVE Model User's Guide
- ❏ Code Composer Studio™ GEL files that implement some of the standard ISS commands to Seamless
- ❏ Binaries and Library files needed by the TMS320C6416 Seamless model for execution
- ❏ Examples that help in using the various features of the TMS320C6416 Seamless model

1.1.2 Licensing Information

The TMS320C6416 Seamless model is a licensed product. Licenses need to be obtained for two different entities of the product

- ❏ TMS320C6416 Seamless CVE model
- ❏ Code Composer Studio IDE on Solaris

Licenses for the model can be obtained by contacting Mentor Graphics Support. Please contact Texas Instruments to obtain licenses for Code Composer Studio IDE.

1.1.3 Supported Platforms

The model is supported only on the UNIX–Solaris platform.

1.1.4 Versions

The following Solaris versions are supported:

- SUNOS 5.5.1 (SOLARIS 2.5.1)
- SUNOS 5.6 (SOLARIS 2.6)

The model supports Seamless CVE version 4.1.

The model supports ModelSim version 5.4c.

The model works with Code Composer Studio IDE V1.1 and V2.0 on the UNIX platform.

1.2 How to Invoke

This section details the steps to be performed to invoke the TMS320C6416 Seamless model. Please invoke the Seamless CVE executable only after the ISS and BIM have been successfully set up, as seen in sections 1.2.2 and NO TAG of this document.

1.2.1 Setting Up the Seamless CVE

Please refer the Mentor Graphics Seamless CVE User Guide for details as to how to setup the Seamless CVE application.

The TMS320C6416 Seamless model is released as part of the Mentor Graphics Seamless release tree.

1.2.2 Setting Up the ISS

1) Set the environment variable TI_PSP_DIR to the installation directory

2) Ensure that the location of cc_app (Code Composer Studio IDE) is in your path.

3) Invoke cc_setup, select the tisimC64xx board and add to the system. For more help, refer to the Code Composer Studio help.

4) Edit the board properties to select the simulator config file simc6416seamless.cfg from the directory: TI_PSP_DIR/isms/TMS320C6416/.

1.2.3 Setting up the BIM

1) Set the environment variable TI_PSP_DIR to the installation directory.

2) Set the environment variables CVE_HOME and MODELSIM.

3) Go to TI_PSP_DIR/isms/TMS320C6416/ and source the script setup_tms320c6416

Sourcing this script will override the TI_PSP_DIR settings and point it to CVE_HOME. If you do not source the script, you need to do the following settings yourself:

❑ LD_LIBRARY_PATH settings

You need to set the LD_LIBRARY_PATH environment variable to include

- TI_PSP_DIR/isms/TMS320C6416,

- TI_PSP_DIR/lib/TMS320C6416, and
- TI_PSP_DIR/lib/TMS320C6416/VIP_MODEL/VIP_END_USER_40/SunOS5/lib.

❑ VIPENV_VHD settings

You need to set the VIPENV_VHD environment variable to

- TI_PSP_DIR/lib/TMS320C6416/VIP_MODEL

❑ PATH settings.

You need to set the path to include the directories

- $(CVE_HOME}/bin, ${VIPENV_VHD}/SunOS5/bin
- ${VIPENV_VHD}/jre/SunOS5/bin

❑ VHDL GENERIC settings.

You need to set the VHDL generic ENDIAN_MODE. By default it is set to 1.

- 1 – little endian
- 0 – big endian

1.3 Supported TMS320C6416 Device Pins

The following table summarizes the pins simulated by the TMS320C6416 Seamless Model.

Table 1–1. Pins Simulated by the TMS320C6416 Seamless Model

Pin Name	Direction	Width	Supported	Peripherals
CLKIN	IN	1	X	CLOCKING/PLL
CLKMODE	IN	2–0	–	CLOCKING/PLL
PLLV	OUT	1	–	CLOCKING/PLL
RESET	IN	1	X	SYSTEM CONTROL
PCI_EN	IN	1	X	SYSTEM CONTROL
MCBSP2_EN	IN	1	X	SYSTEM CONTROL
TINP	IN	2–0	–	TIMER
BARDY	IN	1	X	EMIFB (16 BIT EXTERNAL MEMORY INT)
BHOLD	IN	1	–	EMIFB (16 BIT EXTERNAL MEMORY INT)
BECLKIN	IN	1	X	EMIFB (16 BIT EXTERNAL MEMORY INT)
AARDY	IN	1	X	EMIFA (64 BIT EXTERNAL MEMORY INT)
AHOLD	IN	1	–	EMIFA (64 BIT EXTERNAL MEMORY INT)
AECLKIN	IN	1	–	EMIFA (64 BIT EXTERNAL MEMORY INT)
TCLK	IN	1	–	EMULATION CONTROL
TDI	IN	1	–	EMULATION CONTROL
TMS	IN	1	–	EMULATION CONTROL
TRST	IN	1	–	EMULATION CONTROL
UXCLK	IN	1	X	UTOPIA
URSOC	IN	1	X	UTOPIA
URENB	IN	1	X	UTOPIA
URDATA	IN	7–0	X	UTOPIA
CLKS0	IN	1	–	MCBSP0
CLKS2_GP8	INOUT	1	X	MCBSP2
DR0	IN	1	X	MCBSP2
GP0	INOUT	1	X	GPIO
GP1_CLKOUT4	INOUT	1	X	SYSTEM CONTROL
GP2_CLKOUT6	INOUT	1	X	SYSTEM CONTROL
NMI	IN	1	X	SYSTEM CONTROL
GP3	INOUT	1	X	SYSTEM CONTROL
GP4_EINT4	INOUT	1	X	SYSTEM CONTROL
GP5_EINT5	INOUT	1	X	SYSTEM CONTROL
GP6_EINT6	INOUT	1	X	SYSTEM CONTROL
GP7_EINT7	INOUT	1	X	SYSTEM CONTROL
XSP_CS	OUT	1	–	PCI SERIAL EEPROM INTERFACE
TOUT	OUT	2–0	–	TIMER
BED	INOUT	15–0	X	EMIFB 16 BIT MEMORY INTERFACE
BEA	INOUT	8–1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA09	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA10	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE

Table 1–1. Pins Simulated by the TMS320C6416 Seamless Model (Continued)

Pin Name	Direction	Width	Supported	Peripherals
BEA11_UTOPIAEN	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA20_LENDIAN	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA13_PCIEEAI	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA14_BECLKINSEL0	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA15_BECLKINSEL1	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA12	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA17_AECLKINSEL1	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA18_BOOTMODE0	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA19_BOOTMODE1	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BEA16_AECLKINSEL0	INOUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
AEDH	INOUT	31–0	X	EMIFA 64 BIT MEMORY INTERFACE
AEDL	INOUT	31–0	X	EMIFA 64 BIT MEMORY INTERFACE
EMU	INOUT	9–0	–	EMULATION CONTROL
EMUCLK	INOUT	1–0	–	EMULATION CONTROL
UXCLAV	OUT	1	X	UTOPIA
UXENB	IN	1	X	UTOPIA
UXADDR0	IN	1	X	UTOPIA
URCLAV	OUT	1	X	UTOPIA
URCLK	IN	1	X	UTOPIA
DX1_UXADDR4	INOUT	1	X	MCBSP1/UTOPIA
FSX1_UXADDR3	INOUT	1	X	MCBSP1/UTOPIA
FSR1_UXADDR2	INOUT	1	X	MCBSP1/UTOPIA
DR1_UXADDR1	IN	1	X	MCBSP1/UTOPIA
CLKX1_UXADDR4	INOUT	1	X	MCBSP1/UTOPIA
CLKS1_UXADDR3	IN	1	X	MCBSP1/UTOPIA
CLKR1_UXADDR2	INOUT	1	X	MCBSP1/UTOPIA
URADDR	IN	1–0	X	UTOPIA
CLKX0	INOUT	1	X	MCBSP0
CLKX2_XSPCLK	INOUT	1	X	MCBSP2/PCI
CLKR0	INOUT	1	X	MCBSP0
CLKR2	INOUT	1	X	MCBSP2
FSX0	INOUT	1	X	MCBSP0
FSX2	INOUT	1	X	MCBSP2
FSR0	INOUT	1	X	MCBSP0
DR2_XSPDI	IN	1	X	MCBSP2/PCI
FSR2	INOUT	1	X	MCBSP2
PCBE0	INOUT	1	–	PCI
HDS2_PCBE1	INOUT	1	X	HPI/PCI
HRW_PCBE2	INOUT	1	X	HPI/PCI
GP9_PIDSEL	INOUT	1	X	GPIO/PCI
GP10_PCBE3	INOUT	1	X	GPIO/PCI
GP11_PREQ	INOUT	1	X	GPIO/PCI
GP12_PGNT	INOUT	1	X	GPIO/PCI
GP13_PINTA	INOUT	1	X	GPIO/PCI

Table 1–1. Pins Simulated by the TMS320C6416 Seamless Model (Continued)

Pin Name	Direction	Width	Supported	Peripherals
GP14_PCLK	INOUT	1	X	GPIO/PCI
GP15_PRST	INOUT	1	X	GPIO/PCI
HINT_PFRAME	INOUT	1	X	HPI/PCI
HCNTRL0_PSTOP	INOUT	1	X	HPI/PCI
HCNTRL1_PDEVSEL	INOUT	1	X	HPI/PCI
HHWIL_PTRDY	INOUT	1	X	HPI/PCI
HAS_PPAR	INOUT	1	X	HPI/PCI
HCS_PPERR	INOUT	1	X	HPI/PCI
HDS1_PSERR	INOUT	1	X	HPI/PCI
HRDY_PIRDY	INOUT	1	X	HPI/PCI
HD_AD	INOUT	31–0	X	HPI/PCI
BCE	OUT	3–0	X	EMIFB 16 BIT MEMORY INTERFACE
BBE	OUT	1–0	X	EMIFB 16 BIT MEMORY INTERFACE
BSDCAS	OUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BSDRAS	OUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BSOE3	OUT	1	–	EMIFB 16 BIT MEMORY INTERFACE
BSDWE	OUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BHOLDA	OUT	1	–	EMIFB 16 BIT MEMORY INTERFACE
BBUSREQ	OUT	1	–	EMIFB 16 BIT MEMORY INTERFACE
BPDT	OUT	1	–	EMIFB 16 BIT MEMORY INTERFACE
BECLKOUT1	OUT	1	X	EMIFB 16 BIT MEMORY INTERFACE
BECLKOUT2	OUT	1	–	EMIFB 16 BIT MEMORY INTERFACE
AEA	OUT	19–0	X	EMIFA 64 BIT MEMORY INTERFACE
ACE	OUT	3–0	X	EMIFA 64 BIT MEMORY INTERFACE
ABE	OUT	7–0	X	EMIFA 64 BIT MEMORY INTERFACE
ASDCAS	OUT	1	X	EMIFA 64 BIT MEMORY INTERFACE
ASDRAS	OUT	1	X	EMIFA 64 BIT MEMORY INTERFACE
ASOE3	OUT	1	–	EMIFA 64 BIT MEMORY INTERFACE
ASDWE	OUT	1	X	EMIFA 64 BIT MEMORY INTERFACE
ASCKE	OUT	1	–	EMIFA 64 BIT MEMORY INTERFACE
AHOLDA	OUT	1	–	EMIFA 64 BIT MEMORY INTERFACE
ABUSREQ	OUT	1	–	EMIFA 64 BIT MEMORY INTERFACE
APDT	OUT	1	–	EMIFA 64 BIT MEMORY INTERFACE
AECLKOUT1	OUT	1	X	EMIFA 64 BIT MEMORY INTERFACE
AECLKOUT2	OUT	1	–	EMIFA 64 BIT MEMORY INTERFACE
TDO	OUT	1	–	EMULATION CONTROL
UXSOC	OUT	1	X	UTOPIA
UXDATA	OUT	7–0	X	UTOPIA
DX0	OUT	1	X	MCBSP0
DX2_XSPDO	OUT	1	X	MCBSP2/PCI

1.4 Simulation Features

1.4.1 BootLoad

The TMS320C6416 Seamless model supports device BOOTLOAD through the EMIF pins.

At reset the model samples the pins BEA(19:18). The bootload operation happens depending on these pin values:

0 No Boot load. The model begins execution from the memory located at address 0.

1 Boot load through HPI/PCI. The boot load is done through the Host Port Interface (HPI) or the Peripheral Component Interconnect (PCI) port. The CPU will wait till an external host initializes the CPU's memory space as necessary through the parallel port. Once this is done, the CPU begins execution from address 0. Boot load happens through PCI if PCI is enabled, otherwise Bootload happens through HPI.

2 Boot load from EMIF B CE1 space start address (0x64000000). (ROM boot process) The EDMA copies 1K bytes from the beginning of CE1 to address 0, using default ROM timings. After the transfer the CPU begins executing from address 0.

3 Quick Boot Load. This feature is supported by the model and not the actual device. In this mode, the bootload will be performed through debug reads of 1K bytes from memory address 0x90000000. Once this is complete, the CPU begins execution from address 0.

1.4.2 Reset Sequence

The TMS320C6416 Seamless model performs reset by sampling the RESET pin. The reset pulse width should be a minimum of 10 CPU clock cycles. The RESET pin is not sampled after the first reset.

The device reset uses active low signal RESET. While the RESET is low the model is held in its reset state. The control will be with the Hardware simulator. Most output pins go to their default state. The rising edge on the RESET starts the model run with the prescribed boot configuration. The following pins are sampled at reset:

- BEA20_LENDIAN : The model will sample this pin to ensure that the Instruction Set Simulator (ISS) and the Bus Interface Model (BIM) are operating in the same endian. A '1' on this pin will mean that the model should

be operating in the little endian mode, whereas a ’0’ on this will mean that the model should be operating in the big endian. The model will check to see if the ISS and BIM are operating in the same endian, and if they are not it will give an error message and exit.

- Boot load config pins BE18_BOOTMODE0, BE19_BOOTMODE1. The values on these pins determine the boot load mode as explained in section 1.4.1, on page 1-9.

1.4.3 Clock Behavior

The model operates from the clock supplied by the CLKIN pin. The EMIF operates at 1/4th this clock frequency, whereas the HPI/PCI and Utopia operate at 1/2 this clock frequency. For the model to work properly, the rising edge of all the clocks should be in sync when the reset is applied.

The EMIF doesn’t support other clock frequencies or external clock inputs.

1.5 Seamless Features Supported

1.5.1 Seamless Optimization

1.5.1.1 Memory Optimization

To learn more about Memory Optimization, please read the Mentor Graphics Seamless CVE User's Guide.

TMS320C6416 PSP supports Memory Optimization. This feature has been tested with Denali SRAM models. While using the Memory optimization feature, please register the memory ranges that are to be optimized, with the Seamless CVE.

1.5.1.2 Time Optimization

To learn more about Time Optimization refer to the Mentor Graphics Seamless CVE manuals.

The TMS320C6416 Seamless model supports the Time Optimization feature. This optimization can be enabled at any time of execution of the model. The model takes care of any unfinished pipelined memory requests that may be pending at the time of turning on the Time Optimization feature, before switching into optimized mode of execution.

1.5.1.3 Ratio Time Optimization

To learn more about Ratio Time Optimization please refer to the Mentor Graphics Seamless CVE User's Guide.

On the same lines of support for Time Optimization, the TMS320C6416 Seamless model supports Ratio Time Optimization also.

1.5.2 HW Break

This command is implemented as a Code Composer Studio GEL function. Invoking this function from Code Composer Studio IDE causes the hardware simulator to go into an interactive mode. It is possible to simultaneously use both the hardware simulator and Code Composer Studio IDE interactively.

1.5.3 Loading COFF into External Memory

The TMS320C6416 Seamless model allows loading of a COFF file into external memories. The hardware simulator is not advanced while the load is in progress.

This feature is supported only when the external memories are Seamless memories such as Denali memory models.

The COFF load is performed by loading the file through Code Composer Studio.

1.6 Simulation Accuracy

The TMS320C6416 Seamless Model simulates the following blocks of the device:

- CPU
- L1, L2 Cache System
- Enhanced DMA sub–system
- Interrupt Selector
- Timers
- Serial Ports (McBSP)
- TCP, VCP co–processors
- EMIF interface
- Utopia–II
- HPI/PCI
- GPIO

CPU – The CPU core of the device has been modeled accurately both in terms of functionality and cycles.

L1, L2 Cache System – All of the functionality in the L1, L2 Cache system has been modeled in the TMS320C6416 Seamless Model. The simulation of the Cache System is still not 100% cycle accurate though. L1P is not accurate on the Instruction Read Misses. L1D is very accurate on Data Read misses and Victim requests to L2, but is about 95% accurate on L1D write miss cycles.

EDMA – The EDMA subsystem is modeled functionally to support complete programmability of the EDMA configuration registers. The number of cycles spent in the subsystem is however inaccurate.

Peripherals such as the EMIF, Timer, Serial Ports, Utopia, GPIO, HPI, and PCI are modeled accurately for functionality and cycles. Coprocessors TCP/VCP are modeled accurately for functionality and cycles.

Please refer to Section 1.7 on page 1-14 for known limitations in the TMS320C6416 Seamless model.

1.7 Known Limitations

- Only Asynchronous and SDRAM interfaces of the EMIF are supported. If EMIF spaces are configured to other interface types the behavior of the model is undefined.
- The EMIF always runs at 1/4th the CPU clock frequency.
- External EMIF clock inputs are not supported.
- The model samples RESET only once during simulation. Subsequent reset events are not communicated to the ISS.
- Debug reads/writes to HPI registers are not supported.
- McBSP CLKS is not supported.
- McBSP external frame sync is supported from Release 1.2.2 onward.
- Debug reads/writes to UTOPIA registers are not supported.
- Debug writes to GPIO registers are not supported.
- Debug reads/writes to PCI are not supported.
- SDRAM refresh mode is not supported.
- SRDAM burst access is not supported.
- Generation of frame sync in McBSP receiver mode is not supported.
- EEPROM and Power Management for PCI is not supported.

Chapter 2

TMS320C6416 Seamless CVE Model Tutorial

This chapter contains tutorials that allow you to explore some of the capabilities of the TMS320C6416™ Seamless CVE model.

Topic **Page**

2.1 Hooking up the Denali SRAM

To learn about the Denali memory models and their usage, please read the Mentor Graphics Seamless CVE User's Guide.

2.1.1 Connecting the TMS320C6416 Device and a Denali SRAM Model

In order to connect the C6416 device to the Denali SRAM module, instantiate the memory models in the VHDL testbench and connect the address bus, data bus chip select, etc. When the testbench is loaded in the vsim, these memory models will be registered with the CVE. Map the memory ranges and their usage modes. For more information on Memory configuration and usage refer to the Mentor Graphics Seamless CVE User's Guide.

Example 1 demonstrates the usage of Denali SRAM memory models. The VHDL testbench contains an instance of TMS320C6416 processor and an instance each of 4 Denali SRAMs of different memory widths. An SRAM is connected to each of the processor's EMIF–A CE spaces. An 8 bit SRAM is connected to the EMIF–A CE0 space, a 16 bit SRAM to the EMIF–A CE1 space, a 32 bit SRAM to the EMIF–A CE2 space and a 64 bit SRAM to the EMIF–A CE3 space.

As shown in the CVE configuration file, these instances of the memories are mapped to the appropriate address ranges with the appropriate data bus size by overriding the default data bus widths. The example configures the EMIF–A CE0, CE1, CE2 and CE3 spaces to 8, 16, 32 and 64 bit Asynchronous interfaces by writing appropriate control words to the CE space control registers. Once the configuration of the EMIF is done, the example performs reads & writes to these memories. You can turn on/off Seamless Optimizations on these memories and observe the effect.

2.1.2 Performing Seamless Optimizations

The TMS320C6416 Seamless CVE model supports all types of Seamless CVE optimizations. For information on optimizations and their usage refer to the Mentor Graphics Seamless CVE User's Guide.

Example 2 connects four instances of Denali SRAMs of different memory widths to the TMS320C6416 Seamless model. The example performs read and write accesses to the external memory ranges. The Modelsim '.do' files contain commands to toggle Seamless Optimizations at different times during the simulation.

2.1.3 Performing COFF Load into External Memory

A COFF file can be loaded into external memory by loading the file in the Code Composer Studio™ IDE. Simulation time is not advanced while performing the COFF load.

Another way to perform the same activity is by loading a memory image into the external memory.

Example 8 illustrates this with a COFF file that loads into and executes from external memory. A load command from Code Composer Studio IDE results in the Denali memories attached to the TMS320C6416 model being written with values without advancing the simulation time.

2.2 Using the Serial Port

2.2.1 Using the Serial Port for Transmission and Reception

Each of the three serial ports of TMS320C6416 can be programmed to transmit and receive the data in serial fashion.

Example 3 demonstrates this capability. It performs transfers from McBSP0 (through the DX0 pin) and the transfer is received in McBSP1 (through the DR1 pin). Similarly, it transmits from McBSP1 (through the DX1 pin) and receives in McBSP2 (through the DR2 pin). Likewise, it transmits from McBSP2 (through the DX2 pin) and receives it in McBSP0 (through the DR0 pin). The sample testbench instantiates the TMS320C6416 Seamless model and connects the signals appropriately. It generates the system clock and the reset also.

To run this example:

1) Go to the example3 directory and run the scriptcompile_example.sh. This script compiles the TMS320C6416 seamless model and the sample testbench into the work directory.

2) Run the script runCve.sh, present in the directory. This will bring up the Seamless CVE and Modelsim. In the Modelsim command window, give the command ”run –all”. This will bring up Code Composer Studio™ IDE.

3) For CCS IDE V1.1: Build the project in the CCS IDE, then load the COFF “example3.out”. Set breakpoints at “passed” and “failed”. Issue a RUN command in CCS IDE. When the model stops at the breakpoint, click on GEL —> Seamless —> HW_BREAK. Now you can observe the waveform in the Modelsim window.

4) For CCS IDE V2.0: In the UNIX command prompt, assemble the file “example3.asm” and then link “example3.obj” to create the COFF “example3.out”. Load this COFF in the CCS IDE. Set breakpoints at “passed” and “failed”. Issue a RUN command in CCS IDE. When the model stops at the breakpoint, click on GEL —> Seamless —> HW_BREAK. Now you can observe the waveform in the Modelsim window.

2.2.2 Generating Serial Port Events

Each of the three Serial Ports of TMS320C6416 device can be programmed to send events to XDMA when the serial port is ready for transmission or when it completes the reception. Upon receiving the events, XDMA can either send the data for transmission to the serial port or read the received data from the

serial port if the XDMA was programmed properly. For details about programming XDMA you can refer to the TMS320C6000 Peripherals Guide (SPRU190).

Example 4 demonstrates this capability. It programs XDMA to transfer data from memory to the MCBSP0 register DXR upon receiving the event XEVT0 on one channel. It also programs XDMA to transfer data from the MCBSP1 register DRR to memory upon receiving the event REVT1 in some other channel.

Example4 then takes out the MCBSP0 (transmitter) and MCBSP1 (Receiver) from reset. MCBSP0 sends XEVT0 to XDMA, upon which XDMA transfers the data from the memory to the MCBSP0 register DXR. Then MCBSP0 starts transmitting the data in serial fashion through the pin DX0. MCBSP1 receives the data through the pin DR1. At the end of reception, MCBSP1 sends the event REVT1 to XDMA. Upon receiving the event XDMA transfers the data from MCBSP1 register DRR to memory.

Example 4 instantiates the TMS320C6416 Seamless model and connects the signals appropriately. It generates the system clock and the reset as well.

To run this example:

1) Go to the example4 directory and run the script compile_example.sh. This script compiles the TMS320C6416 seamless model and the sample testbench into the work directory.

2) Run the script runCve.sh, present in the directory. This will bring up the Seamless CVE and Modelsim. In the Modelsim command window, give the command ”run –all”. This will bring up the Code Composer Studio (CCS) IDE.

3) For CCS IDE V1.1: In the CCS IDE load the COFF example4.out, and set break points at ”passed” and ”failed”. Issue a RUN command in the CCS IDE. When the model stops at the breakpoint, click on GEL —> Seamless —> HW_BREAK . Now you can observe the waveform in the Modelsim window.

4) For CCS IDE V2.0: In the UNIX command prompt, assemble the file “example4.asm” and then link “example4.obj” to create the COFF “example4.out”. Set break points at ”passed” and ”failed”. Issue a RUN command in the CCS IDE. When the model stops at the breakpoint, click on GEL —> Seamless —> HW_BREAK . Now you can observe the waveform in the Modelsim window.

2.3 Simulating External Interrupts

2.3.1 Using External Interrupts

External devices can interrupt the CPU of TMS320C6416 by triggering off the External Interrupt pins EINT4, EINT5, EINT6, EINT7 and NMI.

Example5 demonstrates this capability. The ISR 4 writes 0xABCD in the CPU register B1 to indicate a pass condition. The sample test bench triggers the external interrupt–4 after 700ns.

When the CPU gets an interrupt, it completes the execution of the current instruction and then jumps to the ISR and changes the register B1 from 0 to 0xABCD. Upon return from interrupt, the CPU comes back to the instruction from where it jumped to the ISR.

To run the example:

1) Go to the example5 directory and run the script compile_example.sh. This script compiles the TMS320C6416 seamless model and the sample testbench into the work directory.

2) Run the script runCve.sh present in the directory. This will bring up the Seamless CVE, Modelsim and Code Composer Studio(CCS).The Model sim runs the do file automatically.

3) For CCS IDE V1.1: Build the project and load the COFF file example5.out in CCS.

4) For CCS IDE V2.0: Assemble external_interrupt.asm and vectors.asm. Link external_interrupt.obj and vectors.obj using the linker command file external_interrupt.cmd to create the COFF. Load the COFF in CCS.

5) Set the breakpoints at ”passed” and ”failed”.

6) Now press RUN in the CCS IDE.After running for 700ns, the testbench triggers the pin EINT4.Now CCS jumps to the ISR 4 upon receiving the interrupt. Then it returns from the ISR 4 and halts in either “passed” or “failed”.

2.4 Using the HPI

This section describes examples that use the HPI of the TMS320C6416 device. These examples are packaged as part of the release as Example6 and Example7.

2.4.1 Performing HPI Bootload

The TMS320C6416 device can be instructed to perform Bootload from HPI by setting the BEA18_BOOTMODE0 and BEA19_BOOTMODE1 pins to values 0 and 1, respectively. These pins are sampled at reset. The host performs the bootload operation by writing to specific memory locations in the TMS320C6416 device. The host indicated completion of the bootload operation by generating a host interrupt (HINT).

Example6 demonstrates this capability. In this example, a sample testbench instantiates the TMS320C6416 Seamless model. The testbench generates the clocks and the reset. The example also comes with a set of Modelsim .do files that perform the task of the host by driving the appropriate pins.

To run this example:

1) Go to the Example6 directory and run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Execute the runCve.sh script present in the Example6 directory. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) In the Code Composer Studio IDE, give a Step command through Debug –> Step. This operation will complete after the bootload is completed and the first instruction is executed.

4) The example loads a sample code starting from memory location 0x00000000. You can view the contents of memory locations in the Code Composer Studio IDE after the bootload operation is complete.

5) To execute the sample boot code, continue stepping through the Code Composer Studio debugger.

2.4.2 Performing HOST Reads and Writes

The program sets up data at the address and then transfers control to the Host by setting the HINT control register. The Host performs a write transfer and then interrupts the DSP by setting the DSPINT bit of the HPI control register. The same process can be repeated for read operation.

To run this example:

1) Go to the Example7 directory and run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Execute the runcve.sh script present in the Example6 directory. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) For CCS IDE V1.1: In the Code Composer Studio IDE, build the project and load hpi_rw.out through File–> Load Program.

4) For CCS IDE V2.0: Assemble hpi_rw.asm and link it with the linker command file hpi_rw.cmd and then load the COFF in the CCS IDE.

5) Select Debug–>Breakpoint. Enter Break locations as ”OK” and ”NOT_OK”.

6) Run the example by pressing F5.

7) When the ”OK” breakpoint is hit, select GEL–>SEAMLESS CVE–>HW_BREAK from the CCS menu. View the wave form in Modelsim to see the HPI transaction.

2.5 Using the Utopia

In this section we describe an example that uses the Utopia of the TMS320C6416 device. This example is packaged as part of the release as Example9.

2.5.1 Performing Read and Write of an ATM Cell

The Utopia on TMS320C6416 is used in the slave mode. It communicates with Utopia in master mode. The Utopia in master mode is simulated using .do files.

In this example a sample testbench instantiates the TMS320C6416 Seamless model. The testbench generates the clocks and the reset. The test case programs the DMA to make 14 words (of a single standard cell) read as well as write transfer, and interrupt the cpu to signal the completion of the transfer. In the actual program, the Utopia Configuration Register (UCR) is configured to read a cell. It then waits for a DMA interrupt to come. In the Interrupt Service Routine (ISR) of the DMA interrupt, the configuration of the UCR is changed. The .do file first initiates a read transfer followed by a write transfer using the appropriate events.

To run this example:

1) Go to the Example9 directory and run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Execute the runcve.sh script present in the Example9 directory. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) For CCS IDE V1.1: Build the project in CCS IDE to generate the example9.out file.

4) For CCS IDE V2.0: Assemble the ut_rtx_sp.asm and vectors.asm and link them to get the COFF example9.out.

5) In the CCS Command Window, take the “init.cmd” file. This will run the testcase and log the output in “utopia.log”

6) Once the testcase has run to completion, the control will be with Modelsim.

7) The test case output will now be available in “utopia.log”. The testcase reads a cell through the UTOPIA and writes it back. The log thus contains the contents of memory locations 0x3000–0x3037, where the cell has been read in, and the output of pin “uxdata”, which shows the cell which has been transferred out.

8) The log generated should be compared with result.log to verify that the testcase has passed.

2.6 Using the Hardware/Software Synchronization Feature

This section describes the Hardware/Software Synchronization feature in the TMS320C6416 Seamless model that allows the user to control synchronization between the ISS and the BIM. This feature would allow the Hardware to run free without synchronizing with the the Software on every clock cycle. Synchronization on every clock cycle is necessary only when EMIF/HPI/UTOPIA/PCI/McBSP/GPIO transactions are happening or when there is an external interrupt to the TMS320C6416 device.

2.6.1 How to Use the Synchronization Feature

Two GEL functions – ”SYNCHRONIZATION_ON” and SYNCHRONIZATION_OFF” – are provided in the TMS320C6416_seamless.gel file.

To use this function, select either

GEL–>SEAMLESS CVE–>SYNCHRONIZATION_ON

or GEL–>SEAMLESS CVE–>SYNCHRONIZATION_OFF

from the Code Composer Studio menu.

2.6.2 Performing SYNCHRONIZATION_ON/_OFF During EMIF Transactions

This example performs the Synchronization On before the EMIF transaction and Synchronization Off after the EMIF transaction.

To run this example:

1) Go to the Example10 directory and run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Execute the runSynchCve.sh script present in the Example6 directory. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) Once the Code Composer Studio IDE comes up, load the test case Synchtest.out

4) Select Debug–>Probe Points from the menu. This causes the Break/Probe/Profile Points dialog box to appear.

5) Select the Probe Points tab.

6) Under Probe type, select Probe at Location if Expression TRUE. Enter the Location as WRITE_LOOP. This is the point before the EMIF Transaction starts. In expression enter SYNCHRONIZATION_ON(). Click ADD and OK.

7) Again Under Probe type, select Probe at Location if Expression TRUE. Enter the Location as IDLE_LOOP. This is the point before the EMIF transaction starts. In the expression put SYNCHRONIZATION_OFF(). Click ADD and OK.

8) Select Debug–>Break Points from the Menu. This causes the Break/Probe/Profile Points dialog box to appear. Make sure the Breakpoint tab is selected. Select Breakpoint type as Break at Location, and enter the location as END (end of example).

9) Run the test case by pressing F5. When the break point is hit, exit the Seamless CVE.

2.6.3 Synchronization Feature on External Interrupt

This example describes the SYNCHRONIZATION feature on external interrupt. When an external interrupt occurs, ”SYNCHRONIZATION_ON” is called in order to synchronize Hardware and Software

To run the example:

1) Run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Run the runSynchCveInt.sh script to run the example. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) Once the Code Composer Studio IDE comes up, load the test case external_interrupt.out

4) Select Debug–>Probe Points from the menu. This causes the Break/Probe/Profile Points dialog box to appear.

5) Select the Probe Points tab. Under Probe type, select Probe at Location if Expression TRUE. Enter the Location as Before_int. This is the point before the Interrupt occurs. In expression put SYNCHRONIZATION_OFF(). Click ADD and OK.

6) Select STEP in the Code Composer Studio IDE. When Before_loop is hit SYNCHRONIZATION_OFF is called. When Interrupt occurs, SYNCHRONIZATION_ON is called.

2.7 Using Host Port Interface and Utopia On/Off Switch

The TMS320C6416 Seamless model supports a feature to selectively disable simulation of the HPI/PCI and UTOPIA peripherals during simulation. This feature can be utilized to enhance model simulation speed by disabling these peripherals when they are not needed. This can significantly increase the speed of simulation.

By default, both HPI/PCI and UTOPIA are enabled.

2.7.1 How to Use the Switching Feature

Four GEL functions – UTOPIA_ON ,UTOPIA_OFF, HPI_ON and HPI_OFF – are provided.

To use this function, select

GEL–>SEAMLESS CVE–>UTOPIA_ON ,

GEL–>SEAMLESS CVE–>UTOPIA_OFF,

GEL–>SEAMLESS CVE–>HPI_ON, or

GEL–>SEAMLESS CVE–>HPI_OFF.

from the Code Composer Studio menu.

2.8 Using the GPIO

The GPIO in TMS320C6416 is a set of 15 pins, in which each pin can be programmed as an INPUT or OUTPUT pin. They can be programmed to generate EDMA events or CPU interrupts.

2.8.1 Generating EDMA Events and CPU Interrupts Using GPIO

Example11 demonstrates the capability of GPIO to generate EDMA events or CPU interrupts. It programs the EDMA channel 8 (GPIO EVT 0) to transfer one word of data from 0x600 to 0x604 upon receiving the event GPIO EVT0. It programs the GPIO registers to configure the GPIO0 pin as INPUT and the GPIO1 pin as OUTPUT. The program waits for the GPINT0 interrupt. After 5μs, the GPIO pin is triggered to 1 using the do file. Upon this, the GPIO module generates GPINT0 to the CPU and GPIO EVT0 to the EDMA. The EDMA transfers the data from 0x600 to 0x604. At this point the program compares the data. If the data matches, it writes a 1 to the GPIO1 pin. In the testbench, the GPIO1 pin is connected to external interrupt EXTINT4. This triggers External Interrupt 4 to the CPU. The test case branches to the label “passed” if the data matches, or to “failed” if it does not.

The sample testbench instantiates the TMS320C6416 Seamless model and connects the signals appropriately. It also generates the system clock and the reset.

To run this example:

1) Go to the example11 directory and run the script compile_example.sh. This script compiles the TMS320C6416 Seamless model and the sample testbench into the work directory.

2) Run the script runCve.sh, present in the directory. This will bring up the Seamless CVE and Modelsim as well as run the do file gpio.do. This will bring up the Code Composer Studio IDE.

3) For CCS IDE V1.1: In the CCS environment, build the project and load the COFF example11.out. Set breakpoints at “passed” and “failed”. Issue a RUN command in CCS. When the model hits the breakpoint, click on GEL —> Seamless —> HW_BREAK. You can now observe the waveform in the Modelsim window.

4) For CCS IDE V2.0: Assemble and link example11.asm to get the COFF example11.out. Load the COFF in CCS. Set breakpoints at “passed” and “failed”. Issue a RUN command in CCS. When the model hits the breakpoint, click on GEL —> Seamless —> HW_BREAK. You can now observe the waveform in the Modelsim window.

2.9 Using the PCI

This section describes examples that use the PCI of the TMS320C6416 device. These examples are packaged as part of the release as example12, example13, and example14.

2.9.1 Performing PCI Bootload

The TMS320C6416 device can be instructed to perform Bootload from PCI by setting the BEA_BOOTMODE0 and BEA_BOOTMODE1 pins to 0 and 1, respectively. These pins are sampled at reset. The host performs the Bootload operation by writing to specific memory locations in the TMS320C6416 device. The host indicates completion of the Bootload operation by generating a host interrupt to the DSP.

Example12 demonstrates this capability. In this example, a sample testbench instantiates the TMS320C6416 Seamless model. The testbench generates the clocks and the reset. The example also comes with a set of Modelsim .do files that perform the task of the host by driving the appropriate pins.

To run this example:

1) Go to the example12 directory and run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Execute the runCve.sh script present in the example12 directory. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) In the Code Composer Studio IDE, give a Step command through Debug –> Step. This operation will complete after the Bootload is completed and the first instruction is executed.

4) The example bootloads a sample code into memory location starting from 0x00000000. You can view the contents of memory locations in the Code Composer Studio IDE after the Bootload operations is complete.

5) To execute the sample boot code, continue stepping through the Code Composer Studio debugger.

2.9.2 Performing PCI Slave Reads/Writes (DSP PCI Port in Slave Mode)

The PCI port on TMS320C6416 can operate in slave mode. This is demonstrated in example13. The DSP configures its control registers to receive interrupts from the PCI and waits for the interrupt to come. After configuring the PCI registers, the PCI host does a write of 1 and 3 words, respectively, into DSP

memory, and then reads back the same words by making a read of 1 and 3 words, respectively. After this, the PCI host sends an interrupt to the DSP.

To run this example:

1) Go to the example13 directory and run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Execute the runCve.sh script present in the example13 directory. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) For CCS IDE v1.1: In the Code Composer Studio IDE, build the project and run pci_slv_rw.out through File–>Load Program.

4) For CCS IDE v2.0: Assemble pci_slv_rw.asm and vectors.asm and link it with the linker command file pci_slv_rw.cmd. Then load the COFF in the CCS IDE.

5) Select Debug–> Breakpoint. Enter the break location as “END”.

6) Run the example by pressing F5.

7) When the END breakpoint is reached, select GEL–>SEAMLESS CVE –>HW_BREAK from the CCS menu. View the waveform in Modelsim to see the PCI transactions.

2.9.3 Performing PCI Master Reads/Writes (DSP PCI Port in Master Mode)

The PCI port on TMS320C6416 can operate in master mode. This is demonstrated in example14. The DSP configures the PCI control registers appropriately for a single word write. The PCI then fetches the word from the DSP and transfers the word to the PCI host, which is in slave mode. The DSP polls on the status register to keep track of the completion of the transaction. The testcase does three more transactions of three words write, one word read, and three words read in a similar way.

To run this example:

1) Go to the example14 directory and run the compile_example.sh script. This compiles the VHDL testbench and the TMS320C6416 Seamless model.

2) Execute the runCve.sh script present in the example14 directory. This will bring up the Seamless CVE, Code Composer Studio IDE, and Modelsim.

3) For CCS IDE v1.1: In the Code Composer Studio IDE, build the project and run pci_mst_rw.out through File–>Load Program.

4) For CCS IDE v2.0: Assemble pci_mst_rw.asm and vectors.asm and link it with the linker command file pci_mst_rw.cmd. Then load the COFF in the CCS IDE.

5) Select Debug–> Breakpoint. Enter the break location as “END”.

6) Run the example by pressing F5.

7) When the END breakpoint is reached, select GEL–>SEAMLESS CVE –>HW_BREAK from the CCS menu. View the waveform in Modelsim to see the PCI transactions.

Index

T

U