EXCALIBUR™

Excalibur ARM-Based

Hardware Design Tutorial

User Guide
January 2001
Version 1.0

ALTERA®

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com

ARM® POWERED

A-UG-EXC_ARM_HARDWARE-1.0

About this User Guide

User Guide

This user guide provides comprehensive information about the Altera® Excalibur™ ARM®-based hardware design tutorial.

Table 1 shows the user guide revision history.

Table 1. Revision History

Revision	Date	Description
1.0	Jan 20th 2001	Initial release

How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click on the binoculars icon in the top toolbar to open the Find dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at http://www.altera.com.

For additional information about Altera products, consult the sources shown in Table 2.

Table 2. How to Contact Altera

Information Type	Access	USA & Canada	All Other Locations
Altera Literature Services	Electronic mail	lit_req@altera.com *(1)*	lit_req@altera.com *(1)*
Non-technical customer service	Telephone hotline	(800) SOS-EPLD	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-7606	(408) 544-7606
Technical support	Telephone hotline	(800) 800-EPLD (6:00 a.m. to 6:00 p.m. Pacific Time)	(408) 544-7000 *(1)* (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-6401	(408) 544-6401 *(1)*
	Electronic mail	support@altera.com	support@altera.com
	FTP site	ftp.altera.com	ftp.altera.com
General product information	Telephone	(408) 544-7104	(408) 544-7104 *(1)*
	World-wide web site	http://www.altera.com	http://www.altera.com

Note:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The *Excalibur ARM-Based Hardware Design Tutorial User Guide* uses the typographic conventions shown in Table 3.

Table 3. Conventions

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\maxplus2** directory, **d:** drive, **chiptrip.gdf** file.
Bold italic type	Book titles are shown in bold italic type with initial capital letters. Example: ***1999 Device Data Book***.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75 (High-Speed Board Design).*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <*file name*>, <*project name*>**.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of Quartus and MAX+PLUS II Help topics are shown in quotation marks. Example: "Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `_n`, e.g., `reset_n`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\max2work\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Notes:

Contents

User Guide

Notes:

Getting Started

January 2001, ver. 1.0 User Guide

The Excalibur™ solutions ARM®-based embedded processor PLDs hardware design tutorial user guide walks you through the creation and simulation of an ARM-based processor PLD system hardware design. The design targets an Altera® EPXA 10 device (for more information see the *Excalibur ARM-Based Hardware Reference Manual*). The design uses the Altera ARM-based processor core and the AMBA high-performance bus (AHB) to write to a memory-mapped slave peripheral in the PLD portion of the device. The slave peripheral is an arithmetic unit that performs a function based on data received from the ARM-based processor core. The Excalibur bus transactor—a bus functional model provided in the Quartus™ II software, is used along with the ModelSim software to simulate the design and verify the operational interface between the Stripe-to-PLD bridge and the slave peripheral.

This walkthrough involves the following steps:

1. Installing the tutorial
2. Creating a design
3. Behavioral simulation
4. Compiling the design in the Quartus software
5. Performing timing simulation

The instructions assume that:

- Your PC has the following software installed:
  - The Quartus™ II software, version 1.0
  - Exemplar Leonardo Spectrum software licensed for Verilog HDL
  - Model Technology ModelSim SE software (version 5.3d or above)
- You have saved the **arm_tutorial.zip** file onto your hard drive
- You are familiar with the Quartus software
- You are familiar with the ARM-based processor PLD and its AHB architecture

Excalibur designs can be implemented in VHDL, Verilog HDL, AHDL, and graphical format. This walkthrough uses graphical entry.

When using a third-party synthesis tool, the ARM-based processor core is black-boxed in the HDL file. See ATLAS solutions at http://www.altera.com for guidelines on black-boxing functions in various EDA tools.

Installing the Tutorial

Unzip the file **arm_tutorial.zip** onto your hard drive. Table 1 shows the folder structure that is created.

Table 1. Folder Structure

Folder	Description
\arm_tutorial	Contains the Verilog HDL files for the tutorial.
pld_slave.v	The slave peripheral design. *(1)*
ahb_slave_sm.v	The AHB interface state machine that contains all of the logic necessary for the ARM-based processor core to interface with a slave in the PLD part of the ARM-based device through the AHB2PLD bridge.
alu.v	The arithmetic unit that performs the calculations on the two operands.
regfile.v	Contains the operand, operator, and data registers.
wait_state_gen.v	Ensures that the current result is output.
ahb_slave_include.v	Logic that ensures the current include file defines the required parameters.
input.dat	Command language file that contains bus transaction commands.
\behavioral_sim	Contains files for behavioral simulation.
alt_exc_upcore_0.v	Embedded processor simulation model.
input.dat	Command language file that contains bus transaction commands.
slavememory.cfg.dat	Memory configuration file required when using the Excalibur bus transactor.
sim_arm_top.do	ModelSim command file that contains stimulus for simulation.
wave.do	ModelSim waveform command file

Note:
(1) The slave peripheral design comprises the following lower-level Verilog files: **ahb_slave_sm.v**, **alu.v**, **regfile.v**, and **wait_state_gen.v**.

Creating a Design

This section walks you through creating an example of a design—arm_top.bdf (see Figure 1).

This design includes the ARM-based processor core, alt_exc_upcore, and a memory-mapped slave peripheral, pld_slave. Pld_slave accepts two operands and an operator that tells it whether to add, subtract, or multiply the operands. The result is written to a location in a register file.

Figure 2 shows a block design file representation of the pld_slave.

Figure 1. Arm_top.bdf

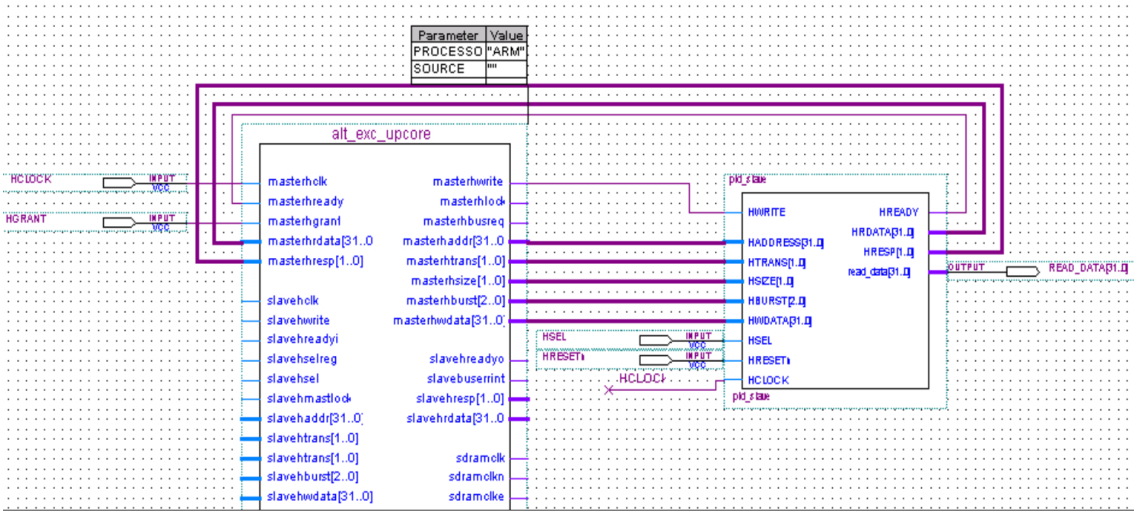
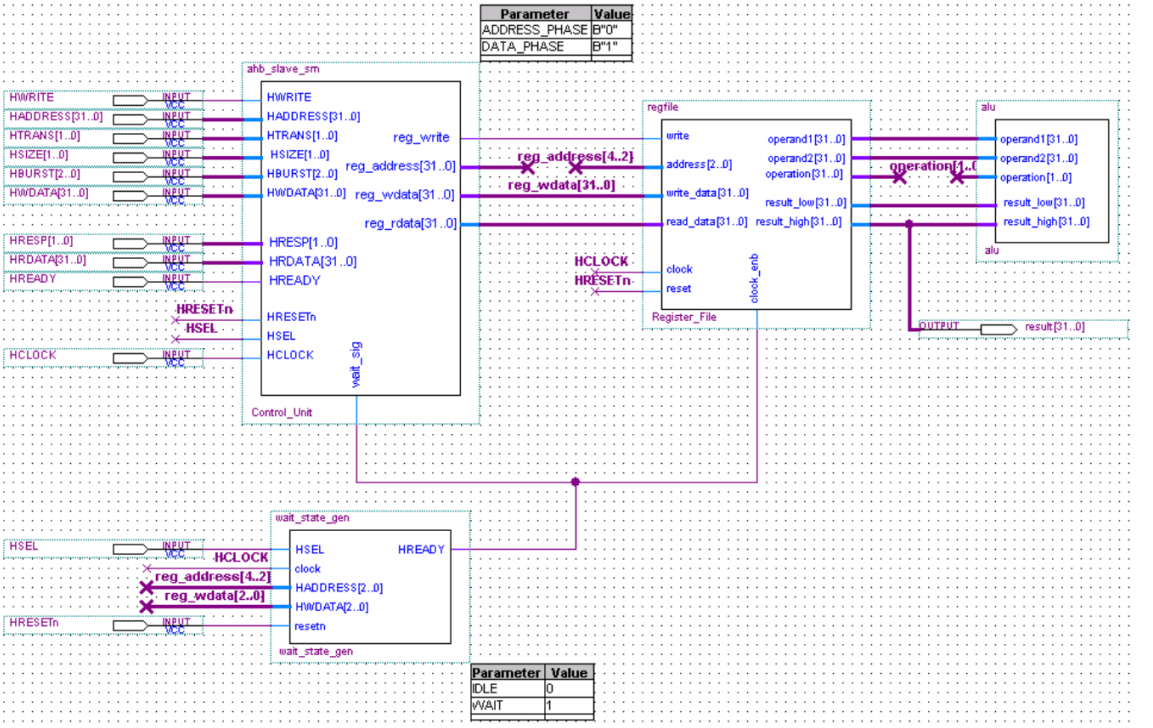


Figure 2. Pld_slave



Creating a design involves the following:

- Creating a new Quartus project and a top-level block design file (**.bdf**). The **.bdf** contains alt_exc_upcore and pld_slave.
- Synthesizing pld_slave in Leonardo Spectrum and importing the EDIF output into the Quartus software via a block symbol file (**.bsf**).
- Using the Quartus software to place and route the top-level design see *Compilation* on page 21).

Creating a Quartus Project

To create a new project, perform the following steps:

1. Open the Quartus software.
2. Select **New Project Wizard** (File menu).
3. Browse to the **\arm_tutorial** directory.
4. Type in `arm_tutorial` as the project name.

5. Type in `arm_top` as the top-level entity name.
6. Click **Next**.
7. Click **Finish** to create the project.

Creating a New Block Design File

To create the top-level design entity (**arm_top.bdf**) of the **arm_tutorial** project, perform the following steps:

1. Select **New** (File menu).
2. In the **Design Files** tab, select **Block Diagram/Schematic File**.
3. Click **OK**. A new **Block Editor** window appears.
4. Select **Save As** (File menu).
5. Specify **\arm_tutorial** as the folder for saving the file.
6. In the **File Name** box, type in `arm_top.bdf`.
7. Ensure **Add File to Current Project** is turned on.
8. Click **Save**. This saves the file and adds it to the project.

Instantiating Alt_exc_upcore

To instantiate the **alt_exc_upcore** into the top-level design file, **arm_top.bdf**, perform the following steps:.

1. Select **Symbol** (Insert menu).
2. In the **Symbol** dialog box **Libraries** list, expand the **\quartus\libraries** folder, expand the **megafunctions** folder, and expand the **embedded_logic** folder.
3. In the **embedded_logic** folder, select the **alt_exc_upcore** symbol.
4. Click **OK**.
5. Place the symbol by clicking on an empty space in the **Block Editor** window.

Synthesizing Pld_slave in Leonardo Spectrum

The Quartus software synthesizes **pld_slave.v** into EDIF format before being included in a top-level design. To synthesize **pld_slave.v** using Leonardo Spectrum, perform the following steps:

1. Open Leonardo Spectrum.
2. Select **New Project** (File menu), to create a new project.
3. In the **Technology tab**, select APEX™ 20KE as the device family, EP20K1000EFC672 as the device, and –1 as the speed grade.
4. Ensure the **Map I/O Registers** option is unchecked.
5. In the **Input** tab, click on the **Working Directory** icon and select **\arm_tutorial** as your working directory.
6. Click on the **Open Files** icon and select **pld_slave.v**, **alu.v**, **ahb_slave_sm.v**, **regfile.v**, and **wait_state_gen.v**.
7. Click **Open** to include the files in the project.
8. In the **Optimize** tab, ensure the **Add I/O Pads** option is unchecked.
9. In the **Output** tab, specify **pld_slave.edf** as the output filename.
10. Click **Apply**.
11. Click **Run Flow** to begin synthesis.

Leonardo Spectrum issues several messages that do not affect the design:

- ahb_slave_include.v", line 73: Warning text macro (OPER) redefined - replace with new definition
- ahb_slave_include.v", line 74: Warning text macro (RELOW) redefined - replace with new definition
- ahb_slave_include.v", line 74: Warning text macro (REHIG) redefined - replace with new definition

12. When the synthesis is finished, select **Save Project** (File menu) and save as **pld_slave.lsp**.
13. The synthesis run results in the **pld_slave.edf** file. Close Leonardo Spectrum.

Creating a .bsf for Pld_slave

You must create a block symbol file (**.bsf**) representation of the **.edf** file generated by Leonardo Spectrum synthesis, before it can be included in the top-level Quartus **.bdf**.

To create the **.bsf** file for **pld_slave.edf**, perform the following steps:

1. Open **pld_slave.edf**.
2. Select **Create Symbol Files for Entities in Current File**.

The Quartus software completes the generation of the symbol. **pld_slave.bsf** is available for instantiation in the top-level design file.

Instantiating Pld_slave

To instantiate **pld_slave.bsf**, into the top-level design file, **arm_top.bdf**, perform the following steps:

1. Select **Symbol** (Insert menu).
2. Expand the **Project** folder and select **pld_slave.bsf**.
3. Click **OK**.
4. Instantiate the symbol by clicking on an empty space in the **.bdf**.
5. Connect the logic and I/O Pins

To complete the top-level design, you must connect the signals between the blocks and also connect the necessary I/O pins to the input and output ports. To connect the signals, perform the following steps:

1. Click the **Orthogonal Node Tool** button on the toolbar.
2. Click the pin stub of the alt_exc_upcore node (see Table 2) to define the start of the node. Drag the pointer to draw a line that connects to the corresponding pld_slave node (see Table 2).
3. Repeat step 2 for each pair of signals (see Table 2).

Table 2. Nodes

Alt_exc_upcore Node	Pld_slave Node
masterhwrite	HWRITE

masterhaddr[31..0]	HADDRESS[31..0]
masterhtrans[1..0]	HTRANS[1..0]
masterhsize[1..0]	HSIZE [1..0]
masterburst[2..0]	HBURST[2..0]
masterhwdata[31..0]	HWDATA[31..0]
masterhrdata[31..0]	HRDATA[31..0]
masterhresp[1..0]	HRESP[1..0]
masterhready	HREADY

You must instantiate input and output pins in the **arm_top.bdf** file.

To enter I/O pins, perform the following steps:

1. Click the **Symbol Tool** button on the **Block Editor** toolbar. The **Symbol** dialog box opens with the **Repeat-insert mode** option turned on.

When the **Repeat-insert mode** option is on, an outline of the selected symbol remains attached to the cursor, regardless of how many times you click, which allows you to place multiple copies of the symbol. When you want to stop placing copies of a symbol, press Esc or choose **Cancel** (right mouse button pop-up menu).

2. In the **Symbol** dialog box, in the **Libraries** list, expand the **d:\quartus\libraries** folder, expand the **primitives** folder, and expand the **pin** folder.

3. In the **pin** folder, select the input primitive.

4. Click **OK**.

5. Click an empty space four times to insert a total of four input symbols on the left-hand side of the file. Press Esc.

6. Repeat steps 1 to 5 to insert and position an output pin symbol in the file. Figure 1 shows the locations.

7. Name the pins by double clicking on them. Table 3 shows the input pin names; Table 4 shows the output pin names.

Table 3. Input Pin Names

Input Pin Name	Connect To
HCLOCK	masterhclk (alt_exc_upcore); HCLOCK (pld_slave)
HGRANT	masterhgrant (alt_exc_upcore)
HSEL	HSEL (pld_slave)
HRESETn	HRESETn (pld_slave)

Table 4. Output Pin Names

Output Pin Name	Connect To
READ_DATA[31..0]	read_data[31..0] (pld_slave)

After entering the inputs and outputs, you must connect them to the appropriate ports on alt_exc_upcore, and pld_slave.

To connect the pins and primitives by drawing node and bus lines, perform the following steps:

1. Click the **Orthogonal Node Tool** button on the toolbar.
2. Click the pin stub of the clk input pin to define the start of the node, and then drag the pointer to draw a line that connects to the pin stub of the alt_up_core masterhclk port. Draw another line to connect it to the clock port of pld_slave.
3. Repeat step 2 for each pair of signals (see Tables 3 and 4). The symbol where each node is located is specified in parentheses.

Behavioral Simulation

You can perform behavioral simulation on the design before performing compilation and place and route in the Quartus software. You can create a top-level Verilog HDL file from your **arm_top.bdf** using the **Create HDL File** utility in the Quartus software. You can use the resulting **arm_top.v** and lower level Verilog HDL files to simulate the ARM-based processor-PLD system in ModelSim. The Excalibur bus transactor supplies the **.dat** files required to simulate bus transactions.

For more information on the Excalibur bus transactor, refer to the *Excalibur Bus Transactor User Guide*.

To generate the **arm_top.v** file, perform the following steps:

1. Open **arm_top.bdf** in the Quartus software.
2. Select **Create HDL Design File for Current File** (Tools menu).
3. Click **Yes**, if you are prompted to save changes to the file.
4. Verify that HDL file generation is successful, in the Quartus software.
5. Cut and paste the **arm_top.v** file from the **\arm_tutorial** folder into the **\arm_tutorial\behavioral_sim** folder.

Arm_top.v must not be left in the **\arm_tutorial** folder, as it causes conflicts with **arm_top.bdf** in the Quartus compilation section.

Generating .dat Files for Simulation

The Excalibur bus transactor is an executable that supports functional simulation of the ARM stripe master and slave ports. It accepts **input.dat** as its input and generates a **mastercommands.dat** file. The **mastercommands.dat** file is interpreted by **ALTERA_MF.V**, which simulates read/write instructions issued to the PLD by the embedded processor via the STRIPE-to-PLD interface.

The **input.dat** file (see *Appendix B* for an example) specifies the bus transactions to be simulated. The first command idles the AHB system for seven clock cycles. The second command writes a burst of data to address 0x4, 0x8, and 0xc. In the pld_slave, the register at address 4 is operand 1, the register at address 8 is operand 2, the operation to be done is stored at address 12, and the result is stored at address 16. An operation value of 5 specifies addition, 6 specifies subtraction, and 7 specifies multiplication. The second command writes 9 to operand 1, 6 to operand 2, and 5 to the operation register. The expected result from the third command is to read the value 15 (0xf), the sum of operand 1 and 2, from the result register. The next series of writes and read exercises the subtraction function of the slave. The last series of commands specifies multiplication of the operands.

The format of **input.dat** for the bus transactor must be specified in decimal format.

To translate the **input.dat file** to a **mastercommands.dat** file, which can be used with the **ALTERA_MF.V** file in ModelSim, perform the following steps:

1. Open a command prompt window and navigate to the \<*Quartus path*>**\bin** directory.
2. At the command prompt type in `exc_bus_translate\` <*arm_tutorial folder path*>`\behavioral_sim\input.dat`.
3. The **mastercommands.dat** file is now in the \<*Quartus path*>**\bin**.
4. Cut and paste the **mastercommands.dat** file from the \<*Quartus path*>**\bin** folder into the **\arm_tutorial\behavioral_sim** folder.

Creating a ModelSim Project

To create a ModelSim project, perform the following steps:

1. Start ModelSim and select **Run ModelSim**.
2. Click on **Create a Project** in the next window.
3. In the **New Project's Home:** window select **\arm_tutorial**.
4. In the **New Project's Name:** window type in `behavioral_sim`.
5. Click **OK**.
6. Select **No** when prompted for a new HDL source file.
7. Click **Done** on the **ModelSim Welcome** window.

ModelSim creates a work directory in the **\arm_tutorial\behavioral_sim** folder.

Compiling the Test Bench

To compile the test bench, perform the following steps:

1. Select **Compile** (Design menu).
2. To add the include file directory, select **Default Options**.
3. Select the **Verilog Tab** and click the **Add Include Dir** button.
4. Browse to **\arm_tutorial.**
5. Click **Open**.

6. Click **OK**.
7. Compile these files in order: **alu.v**, **regfile.v**, **wait_state_gen.v**, **ahb_slave_sm.v**, **pld_slave.v.**
8. Browse to \arm_tutorial\behavioral_sim.
9. Compile **arm_top.v**.
10. Compile **alt_exc_upcore_0.v**.
11. Browse to <*Quartus path*>**\eda\sim_lib.**
12. Compile **ALTERA_MF.V**.
13. Click **Done.**

Simulating

To simulate, perform the following steps:

1. Select **Load New Design** (Design menu).
2. Scroll down and find the **arm_top** module, select it and click **Load**.

Several timescale warnings, which do not affect the simulation, may appear.

3. Select **Execute Macro** (Macro menu).
4. Select **wave.do** and click **OK**.
5. Select **Execute Macro** (Macro menu).
6. Select **sim_arm_top.do** and click **OK**.

Wave.do sets up the format of the ModelSim waveform window with particular signals in the design. It shows the main AHB signals, inputs, and outputs from the design.

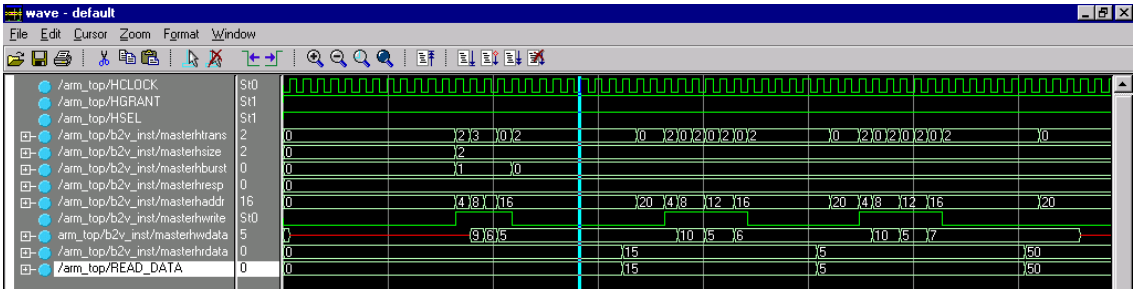
You may see errors regarding unknown options in the waveform depending on your ModelSim version. If the signals are not displayed on the waveform, add the signals (see Figure 3) and re-execute **sim_arm_top.do**.

Sim_arm_top.do contains the following commands that provide stimulus to the Modelsim simulation:

```
restart -f
force -drive -repeat 10ns /slave/HCLOCK 0 0ns, 1 5ns
force -drive /slave/HSEL 1 0ns
force -drive /slave/HGRANT 1 0ns
force -drive /slave/HRESETn 0 0ns
force -drive /slave/HRESETn 1 10ns
run 30000ns
```

Open the ModelSim wave window to view the simulation results. Figure 3 shows the wave window, values are shown in decimal format for readability.

Figure 3. Wave Output



The results can also be seen in text format in the **output.dat** file located in the **\arm_tutorial\behavioral_sim** folder. The output shows the master commands, addresses, and data values for each transaction specified in the **mastercommands.dat** file. See *Appendix A*, which shows the **mastercommands.dat** contents. The simulation confirms that writes and read to the pld_slave were performed correctly using the AHB interface.

Compilation

The Quartus Compiler consists of a series of modules that check the design for errors, synthesize the logic, fit the design into an Altera device, and generate output files for simulation, timing analysis, and device programming.

Compiler Settings

To create Compiler settings, specify EDA tool settings, and compile the design, perform the following steps:

1. Select **Compile Mode** (Processing menu).

2. Choose **Compiler Settings** (Processing menu). The **General** tab of the **Compiler Settings** dialog box appears automatically.

The **General** tab displays only the default compiler general settings created by the Quartus software, when the project was initially created. These default settings are given the name of the top-level design entity in the project, **arm_top**.

To select the device family and device, perform the following steps:

1. In the **Compiler Settings** dialog box, click the **Chips and Devices** tab.
2. In the **Family** list, select EXCALIBUR_ARM.
3. Under **Target Device** tab, select **Specific device selected in Available Devices** list.
4. In the **Available Device** list, select EPXA10F1020C1.
5. To accept the defaults for the remaining compiler settings, click **OK**.

Before compiling the design, you must specify EDA tool settings in the Quartus software.

To specify the appropriate EDA tool settings for use when compiling a design synthesized with the Leonardo Spectrum software, perform the following steps:

1. Choose **EDA Tool Settings** (Project menu). The **EDA Tool Settings** dialog box appears.
2. Under **Design entry/synthesis tool**, select Leonardo Spectrum.
3. Under Simulation tool, select ModelSim (Verilog HDL output from the Quartus software)
4. Click **OK**.

Compiling the Design

To compile the arm_top design, choose **Start Compilation** (Processing menu).

The compiler begins to compile the arm_top design entity, and any subordinate design entities, using the arm_top compiler settings. The results of the compilation are updated in the **Compilation Report** window. Compilation takes approximately 10 minutes.

The compiler may generate one or more of the following warning messages that do not affect the outcome of your design.

- Port slavehwrite of type altupcore and instance inst9 is missing a source signal
- Port slavehreadyi of type altupcore and instance inst9 is missing a source signal
- Timing characteristics of device EPXA10F1020C1 are preliminary
- Found pins functioning as undefined clocks and/or memory enables

Timing Simulation

Compiling alt_up_core in the Quartus software generates the files you need to simulate the design in ModelSim. Three files, **arm_top.vo**, **arm_top_v.sdo**, and **arm_top_modelsim.xrf**, are created in the **\altera\arm_tutorial\simulation\modelsim** folder. You can use these files, and the Excalibur bus transactor, to simulate the ARM-based embedded processor PLD system in ModelSim.

In the absence of a stripe model, simulation of alt_exc_upcore is not performed; functional simulation of the stripe master and slave ports is performed.

Generating DAT files for Simulation

For further information on the **input.dat** file, the **mastercommands.dat** file see *Generating .dat Files for Simulation* on page 18

To translate the **input.dat** file to a **mastercommands.dat** file to be used with the **apex20ke_atoms.v** file in ModelSim, perform the following steps:

1. Open a DOS command window and navigate to the \<*Quartus path*>**\bin** directory.
2. At the command prompt type `exc_bus_translate\<arm_tutorial path>\input.dat`.
3. There should now be a **mastercommands.dat** file in the \<*Quartus path*>**\bin** folder.
4. Cut and paste the **mastercommands.dat** file from the \<*Quartus path*>**\bin** folder into the **\arm_tutorial\simulation\modelsim** folder.

Creating a ModelSim Project

To create a ModelSim project, perform the following steps:

If ModelSim is open, select **End Simulation** (Design menu) and select **Yes**, which exits any previous project simulations. Select **New** and **New Project** (File menu) and continue with step 3 below.

1. Start ModelSim and choose Run ModelSim from the selection of choices.
2. Click on **Create a Project** in the next window.
3. In the **New Project's Home** window select **\altera\arm_tutorial\simulation**.
4. In the **New Project's Name** window type `modelsim`.
5. Click **OK**. If you are prompted to unload the loaded project, select OK.
6. Select **No** when prompted for a new HDL source file.
7. Click **Done** on the **ModelSim Welcome** window.

ModelSim creates a work directory in the \simulation\modelsim directory that the Quartus software created at the end of compilation.

Compiling the Test bench

To compile the test bench, perform the following steps:

1. Copy the files **slavememory.cfg.dat** and **sim_arm_top.do** from the **\arm_tutorial** directory into the **\arm_tutorial\simulation\modelsim** sub-directory.
2. Select **Compile** (Design menu).
3. In the **Files of type** box select **All files(*.*)**.
4. Ensure you are in the **\altera\arm_tutorial\simulation\modelsim** directory, select **arm_top.vo** and click **Compile**.
5. In the \<*Quartus path*>**\eda\sim_lib** folder select **apex20ke_atoms.v** and click **Compile**.

6. Click on **Done**.

Simulating

To simulate, perform the following steps:

1. Select **Load New Design** (Design menu).
2. Find and select the arm_top module and click **Load**.
3. Select **Execute Macro** (Macro menu).
4. Select **sim_arm_top.do** and click **OK**.

It can take up to 30 minutes for simulation to complete, if you are using the ModelSim Altera OEM version.

Sim_arm_top.do contains the following commands that provide stimulus to the Modelsim simulation.

```
restart -f
force -drive -repeat 100ns /arm_top/HCLOCK 0 0ns, 1
50ns
force -drive /arm_top/HSEL 1 0ns
force -drive /arm_top/HGRANT 1 0ns
force -drive /arm_top/HRESETn 0 0ns
force -drive /arm_top/HRESETn 1 10ns
run 30000ns
```

The simulation runs at 30 MHz corresponding to the f_{MAX} of the arm_top design. Without the arithmetic logic unit, the design runs at up to 81 MHz. The multiplication operation is the main contributor to the design's critical path.

View the simulation results by opening the **output.dat** file (in the **\arm_tutorial\simulation\modelsim**) folder. The output shows the master commands, addresses, and data values for each transaction specified in the **mastercommands.dat** file. *Appendix A* details the mastercommands.dat contents; *Appendix C* the expected output.dat simulation results. The simulation confirms that writes and read to the pld_slave were performed correctly using the AHB interface. You can also view the simulation result in the ModelSim waveform window by adding the input pins, output pins, and desired signals to the wave window.

Notes:

The contents of the **mastercommands.dat** are:

```
////////////////////////////////////////////////////////////////////////////////////////////
// FORMAT
// / _____________________________________| reserved (0000)
/// /
//| | +++++++++++++++++++++++++++++++++ valid transaction 1=USE BUS 0=SILENT
//| |+ ____________________________________
//| |+ /  _______________________________| address
//| |+/   /
//| |+|   | ++++++++++++++++++++++++++ write 1=WRITE 0=READ
//| |+|   |+ ___________________________
//| |+|   |+ /  _________________| write data / expected read data
//| |+|   |+/   /
//| |+|   |+|   | ++++++++++++++++ reserved (0) (1=>lock)
//| |+|   |+|   |+
//| |+|   |+|   |+ /--------------- reserved (0) (1=>check read data)
//| |+|   |+|   |+/
//| |+|   |+|   |+| ************** transaction 0=IDLE, 1=BUSY, 2=NONSEQ, 3=SEQ
//| |+|   |+|   |+|*
//| |+|   |+|   |+|* /------------ reserved (0)
//| |+|   |+|   |+|*/
//| |+|   |+|   |+|*| ++++++++++++ burst 0=SINGLE, 1=INCR, 2=WRAP4, 3=INCR4, 4=WRAP8,
//| |+|   |+|   |+|*|+       5=INCR8, 6=WRAP16, 7=INCR16
//| |+|   |+|   |+|*|+
//| |+|   |+|   |+|*|+ /---------- size 0=BYTE, 1=16 BIT, 2=32 BIT
//| |+|   |+|   |+|*|+/ _________
//| |+|   |+|   |+|*|+| / _______| repeat 0000..FFFF
//| |+|   |+|   |+|*|+|/ /
//| |+|   |+|   |+|*|+|| |
// / /  / /  / / / // /
// / /  / /  / / / // /
// |+|  |+|  |+|*|+|| | Line 1, idle(7);
00000000000000000000000000000007//0: idle CYCLES=7h=7
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 8, write(4, 32, 3, 9, 6, 5);
00001000000041000000090020120000//1: write 32-bits NONSEQ INC DATA=9h=9 ADDR=4
00001000000081000000060030120000//2: write 32-bits SEQ INC DATA=6h=6 ADDR=8
000010000000c1000000050030120000//3: write 32-bits SEQ INC DATA=5h=5 ADDR=12
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 17, read(16, 32, 1);
00001000000100000000000020020000//4: read 32-bits NONSEQ SINGLE ADDR=16
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 19, write(4, 32, 1, 10);
00001000000041000000a0020020000//5: write 32-bits NONSEQ SINGLE DATA=ah=10 ADDR=4
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 20, write(8, 32, 1, 5);
00001000000081000000050020020000//6: write 32-bits NONSEQ SINGLE DATA=5h=5 ADDR=8
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 21, write(12, 32, 1, 6);
000010000000c1000000060020020000//7: write 32-bits NONSEQ SINGLE DATA=6h=6 ADDR=12
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 22, read(16, 32, 1);
```

```
00001000000100000000000020020000//8: read 32-bits NONSEQ SINGLE ADDR=16
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 24, write(4, 32, 1, 10);
0000100000004100000000a0020020000//9: write 32-bits NONSEQ SINGLE DATA=ah=10 ADDR=4
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 25, write(8, 32, 1, 5);
00001000000081000000050020020000//10: write 32-bits NONSEQ SINGLE DATA=5h=5 ADDR=8
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 26, write(12, 32, 1, 7);
000010000000c1000000070020020000//11: write 32-bits NONSEQ SINGLE DATA=7h=7 ADDR=12
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| | Line 27, read(16, 32, 1);
00001000000100000000000020020000//12: read 32-bits NONSEQ SINGLE ADDR=16
// |+|  |+|  |+|*|+|| |
// |+|  |+|  |+|*|+|| |
```

Appendix B
Contents of Input.dat

The contents of **input.dat** are:

```
idle(7);

write (4, 32, 3, 9 ,6, 5); /* burst test */
read (16, 32, 1); /* read the 32-bit value from address 16*/

write (4, 32, 1, 10); /* write the 32-bit value 10 into address 4 */
write (8, 32, 1, 5); /* write the 32-bit value 5 into address 8 */
write (12, 32, 1, 6); /* write the 32-bit value 6 into address 12 to add the operands */
read (16, 32, 1); /* read the 32-bit value from address 16*/

write (4, 32, 1, 10); /* write the 32-bit value 10 into address 4 */
write (8, 32, 1, 5); /* write the 32-bit value 5 into address 8 */
write (12, 32, 1, 7); /* write the 32-bit value 7 into address 12 to add the operands */
read (16, 32, 1); /* read the 32-bit value from address 16*/
```

Notes:

Appendix C
Contents of Output.dat

The contents of **output.dat** are:

```
MASTER: trans=[ 2] addr=[00000004] WRITE data=[00000009] expected=[00000009]   WORD OKAY
MASTER: trans=[ 3] addr=[00000008] WRITE data=[00000006] expected=[00000006]   WORD OKAY
MASTER: trans=[ 4] addr=[0000000c] WRITE data=[00000005] expected=[00000005]   WORD OKAY
MASTER: trans=[ 5] addr=[00000010] READ data=[0000000f] expected=[00000000]   WORD OKAY
MASTER: trans=[ 6] addr=[00000004] WRITE data=[0000000a] expected=[0000000a]   WORD OKAY
MASTER: trans=[ 7] addr=[00000008] WRITE data=[00000005] expected=[00000005]   WORD OKAY
MASTER: trans=[ 8] addr=[0000000c] WRITE data=[00000006] expected=[00000006]   WORD OKAY
MASTER: trans=[ 9] addr=[00000010] READ data=[00000005] expected=[00000000]   WORD OKAY
MASTER: trans=[ 10] addr=[00000004] WRITE data=[0000000a] expected=[0000000a]   WORD OKAY
MASTER: trans=[ 11] addr=[00000008] WRITE data=[00000005] expected=[00000005]   WORD OKAY
MASTER: trans=[ 12] addr=[0000000c] WRITE data=[00000007] expected=[00000007]   WORD OKAY
MASTER: trans=[ 13] addr=[00000010] READ data=[00000032] expected=[00000000]   WORD OKAY
```

Notes: