

EDK 8.2 MicroBlaze Tutorial in Spartan

Objectives

This tutorial will demonstrate process of creating and testing a MicroBlaze system design using the Embedded Development Kit (EDK). The tutorial contains these sections:

- System Requirements
- MicroBlaze System Description
- Tutorial Steps

The following steps are described in this tutorial:

- Starting XPS
- Using the Base System Builder Wizard
- Create or Import IP Peripheral
- Design Modification using Platform Studio
- Implementing the Design
- Defining the Software Design
- Downloading the Design
- Debugging the Design
- Performing Behavioral Simulation of the Embedded System

System Requirements

You must have the following software installed on your PC to complete this tutorial:

• Windows 2000 SP2/Windows XP

Note: This tutorial can be completed on Linux or Solaris, but the screenshots and directories illustrated in this tutorial are based on the Windows Platform.

- EDK 8.2i or later
- ISE 8.2i sp1 or later
- Familiarity with steps in the Xilinx ISE 8 In-Depth Tutorial

http://www.xilinx.com/support/techsup/tutorials/tutorials8.htm

In order to download the completed processor system, you must have the following hardware:

• Xilinx Spartan-3 Evaluation Board (3S200 FT256 -4)

- Xilinx Parallel -4 Cable used to program and debug the device
- Serial Cable

Note: It should be noted that other hardware could be used with this tutorial. However, the completed design has only been verified on the board specified above. The following design changes are required:

- Update pin assignments in the system.ucf file
- Update board JTAG chain specified in the download.cmd

MicroBlaze System Description

In general, to design an embedded processor system, you need the following:

- Hardware components
- Memory map
- Software application

Tutorial Design Hardware

The MicroBlaze (MB) tutorial design includes the following hardware components:

- MicroBlaze
- Local Memory Bus (LMB) Bus
 - ◆ LMB_BRAM_IF_CNTLR
 - BRAM_BLOCK
- On-chip Peripheral Bus (OPB) BUS
 - OPB_MDM
 - ♦ OPB_UARTLITE
 - 3 OPB_GPIOs
 - ♦ OPB_EMC

Tutorial Design Memory Map

The following table shows the memory map for the tutorial design as created by Base System Builder.

Dovico	Add	ress	Sizo	Commont
Device	Min	Мах	3126	Comment
LMB_BRAM	0x0000_0000	0x0000_1FFF	8K bytes	LMB Memory
OPB_MDM	0x4140_0000	0x4140_FFFF	64K bytes	MDM Module
OPB_UARTLITE	0x4060_0000	0x4060_FFFF	64K bytes	Serial Output
OPB_GPIO	0x4002_0000	0x4002_FFFF	64K bytes	LED output
OPB_GPIO	0x4000_0000	0x4000_FFFF	64K bytes	Push Buttons
OPB_GPIO	0x4004_0000	0x4004_FFFF	64K bytes	DIP switches
SRAM (EMC MEM0)	0x2010_0000	0x201F_FFFF	512K bytes	SRAM Memory

Table 1: Tutorial Design Memory Map

Tutorial Steps

SetUp

• Spartan-3 board with a RS-232 terminal connected to the serial port and configured for 57600 baud, with 8 data bits, no parity and no handshakes.

Creating the Project File in XPS

The first step in this tutorial is using the Xilinx Platform Studio (XPS) to create a project file. XPS allows you to control the hardware and software development of the MicroBlaze system, and includes the following:

- An editor and a project management interface for creating and editing source code
- Software tool flow configuration options

You can use XPS to create the following files:

- Project Navigator project file that allows you to control the hardware implementation flow
- Microprocessor Hardware Specification (MHS) file

Note: For more information on the MHS file, refer to the "Microprocessor Hardware Specification (MHS)" chapter in the Platform Specification Format Reference Manual.

• Microprocessor Software Specification (MSS) file

Note: For more information on the MSS file, refer to the "Microprocessor Software Specification (MSS)" chapter in the Platform Specification Format Reference Manual..

XPS supports the software tool flows associated with these software specifications. Additionally, you can use XPS to customize software libraries, drivers, and interrupt handlers, and to compile your programs.

Starting XPS

- To open XPS, select Start → Programs → Xilinx Platform Studio 8.2i → Xilinx Platform Studio
- Select Base System Builder Wizard (BSB) to open the Create New Project Using BSB Wizard dialog box shown in Figure 1.
- Click Ok.
- Use the Project File **Browse** button to browse to the folder you want as your project directory.
- Click Open to create the system.xmp file then Save.
- Click Ok to start the BSB wizard.

Note: XPS does not support directory or project names which include spaces.

Project file		
H:/edk_tutorials/edk_81/mb/MB_tutorial/system.xmp		Browse
Advanced options (optional)		

Figure 1: Create New Project Using Base System Builder Wizard

Defining the System Hardware

MHS and MPD Files

The next step in the tutorial is defining the embedded system hardware with the Microprocessor Hardware Specification (MHS) and Microprocessor Peripheral Description (MPD) files.

MHS File

The Microprocessor Hardware Specification (MHS) file describes the following:

- Embedded processor: either the soft core MicroBlaze processor or the hard core PowerPC (only available in Virtex-II Pro and Virtex-4 FX devices)
- Peripherals and associated address spaces
- Buses

• Overall connectivity of the system

The MHS file is a readable text file that is an input to the Platform Generator (the hardware system building tool). Conceptually, the MHS file is a textual schematic of the embedded system. To instantiate a component in the MHS file, you must include information specific to the component.

MPD File

Each system peripheral has a corresponding MPD file. The MPD file is the symbol of the embedded system peripheral to the MHS schematic of the embedded system. The MPD file contains all of the available ports and hardware parameters for a peripheral. The tutorial MPD file is located in the following directory:

\$XILINX_EDK/hw/XilinxProcessorIPLib/pcores/<peripheral_name>/data

Note: For more information on the MPD and MHS files, refer to the "Microprocessor Peripheral Description (MPD)" and "Microprocessor Hardware Specification (MHS)" chapters in the Embedded System Tools Guide.

EDK provides two methods for creating the MHS file. Base System Builder Wizard and the Add/Edit Cores Dialog assist you in building the processor system, which is defined in the MHS file. This tutorial illustrates the Base System Builder.

Using the Base System Builder Wizard

Use the following steps to create the processor system:

- In the Base System Builder Select "I would like to create a new design" then click **Next**.
- In the Base System Builder Select Board Dialog select the following, as shown in Figure 2:
 - ♦ Board Vendor: Xilinx
 - Board Name: Spartan-3 Starter Board
 - ♦ Board Revision: E

 I would like 	to create a system for the fo	llowing developmen	t board	
Board <u>v</u> endor:	Xilins	52		-
Board n <u>a</u> me:	Spartan-3 Starter Board			•
Board <u>r</u> evision:	E			•
Note: Visit the	vendor website for additiona	al board support mate	erials.	
Vendor's Webs	<u>ite</u>	Contact Info		
Download Thin	d Party Board Definition Files	1		
soaru uescripiit	JU			
Spartan-3 Sta includes 1 RS	rter Kit Board utilizes Xilinx S 232 serial port, 2 256kx16 fa	partan-3 XC3S200-4 ast SRAM.8 DIP swit	FT256 device. The tches, 4 push butto	e board ons. four digital
Spartan-3 Sta includes 1 RS 7 segment LE SRAMs are ci	rter Kit Board utilizes Xilinx S 232 serial port, 2 256kx16 fa Ds, 8 LEDs, 1 VGA port, 1 F ombined to form a 32 bit data	partan-3 XC3S200-4 ast SRAM,8 DIP swit PS/2 port. Push butto a bus.	IFT256 device. Th tches, 4 push butto on 1 is used as syst	e board ons, four digital tem reset. 2
Spartan-3 Sta includes 1 RS 7 segment LE SRAMs are ci	rter Kit Board utilizes Xilinx S 232 serial port, 2 256kx16 fa Ds, 8 LEDs, 1 VGA port, 1 F ombined to form a 32 bit data	partan-3 XC3S200-4 ast SRAM,8 DIP swii °S/2 port. Push butto a bus.	IFT256 device. The tches, 4 push butto on 1 is used as syst	e board ons, four digital tem reset. 2
Spartan-3 Sta includes 1 RS 7 segment LE SRAMs are ci	rter Kit Board utilizes Xilinx S 232 serial port, 2 256kx16 fa Ds, 8 LEDs, 1 VGA port, 1 F ombined to form a 32 bit data	partan-3 XC3S200-4 ast SRAM,8 DIP swii 2572 port. Push butto a bus.	IFT256 device. Th tches, 4 push buttc on 1 is used as syst	e board ons, four digital tem reset. 2
Spartan-3 Sta includes 1 RS 7 segment LE SRAMs are co	rter Kit Board utilizes Xilinx S 232 serial port, 2 256kx16 fr Ds, 8 LEDs, 1 VGA port, 1 F ombined to form a 32 bit data	partan-3 XC3S200-4 ast SRAM,8 DIP swii 2S/2 port. Push butte a bus.	IFT256 device. Th tches, 4 push buttc on 1 is used as syst	e board ons, four digital tem reset. 2
Spartan-3 Sta includes 1 RS 7 segment LE SRAMs are co	rter Kit Board utilizes Xillmx S 232 serial port, 2 256kx16 fa Ds, 8 LEDs, 1 VGA port, 1 F ombined to form a 32 bit data	partan-3 XC35200-4 set SRAM,8 DIP swil 'S/2 port. Push butto a bus.	IFT 256 device. Th tches, 4 push buttc on 1 is used as syst	e board ons, four digital tem reset. 2
Spartan-3 Sta includes 1 RS 7 segment LE SRAMs are cr	rter Kit Board utilizes Xilinx S 232 serial port, 2 256kx16 fa Ds, 8 LEDs, 1 VGA port, 1 F ombined to form a 32 bit data	partan-3 XC35200-4 ast SRAM,8 DIP swii '9/2 port. Push butto a bus.	IFT 256 device. Th tches, 4 push buttc on 1 is used as syst	e board ons, four digital tem reset. 2

Figure 2: BSB: Select a Board

- Click *Next*. MicroBlaze is the only processor option for this board.
- Click *Next*. You will now specify several processor options as shown in Figure 3:

Base System Bui	lder – Con	figure Mici	oBlaze		?)
MicroBla:	ze				
System wide setting	s				
<u>R</u> eference clock fre	equency:	Processo frequency	r-Bus clock x		
50.00	MHz	50.00	▼ MHz		
Ensure that your bo <u>R</u> eset polarity:	ard is confi Active H	gured for the	specifed frequency	y.	
Processor configura	tion				
-Debug I/F		1			
On-chip H/W	debua mo	dule			
C ⊠MD with SA C No <u>d</u> ebug	W debug sl	ub			
Micr	l ø8laze		ocal memory Data and Instruction Use BRAM)	r.	
Cache setup	1		8KB 工	1	1
No Cache			C Enable c	ache <u>l</u> ink	
Enable floating	point unit (<u>F</u>	(PU)			
More Info			< Back	Next >	Cancel

Figure 3: Configure Processor

The following is an explanation of the settings specified in Figure 3:

- System Wide Setting:
 - Reference clock frequency: This is the on board frequency of the clock.
 - Processor-Bus clock frequency: This is the frequency of the clock driving the processor system.
- Processor Configuration:
 - Debug I/F:
 - On-Chip H/W Debug module: When the H/W debug module is selected; an OPB MDM module is included in the hardware system. This introduces hardware intrusive debugging with no software stub required. This is the recommended way of debugging for MicroBlaze system.
 - XMD with S/W Debug stub: Selecting this mode of debugging interface introduces a software intrusive debugging. There is a 1200-byte stub that is located at 0x00000000. This stub communicates with the debugger on the host through the JTAG interface of the OPB MDM module.
 - No Debug: No debug is turned on.

Note: For more information about the Xilinx Microprocessor Debugger (XMD), refer to the Xilinx

Microprocessor Debugger (XMD) chapter in the Embedded System ToolsReference Manual.

- Users can specify the size of the local instruction and data memory.
- Cache setup:
 - No Cache: No caching will be used
 - Enable OPB cache: Caching will be used through the OPB bus
 - Enable cache link: Caching will be used through the FSL bus
- You can also specify the use of the floating point unit (FPU).
- Click Next

Select the peripheral subset as shown in Figure 4 and Figure 5. It should be noted that the number of peripheral shown on each dialog box is dynamic based upon your computers resolution.

Note: The Baud rate for the OPB UARTLITE must be updated to 57600.

(Spartan-3 Starter Board Revision E se select the ID devices which you would like to use: devices	
Peripheral: OPB UARTLITE	Data Sheet
Baudrate (bits per seconds): 57600 💌 Data bit <u>s</u> : 8 💌 Parit <u>y</u> : NONE 💌	
EDs_88it Peripheral: OPB GPI0	Data Sheet
T Use interrupt	
- T LED_7SEGMENT	Data Sheet

Figure 4: Configure I/O Interfaces

- OPB UARTLITE baudrate \rightarrow 57600
- LED_7SEGMENT peripheral \rightarrow deselect
- Click Next

Push_Buttons_3Bit		Data Sheet
	<u>ت</u>	
DIP_Switches_8Bit		 Data Sheet
Peripheral: OPB GPIO		
SRAM_256Kx32		
Peripheral: OPB EMC	×	

Figure 5: Configure Additional I/O Interfaces

- Click *Next* on the second Configure Additional IO Interfaces page.
- Click *Next* through the Add Internal Peripherals page as we will not add any in this example.

This completes the hardware specification and we will now configure the software settings.

Using the Software Setup dialog box as shown in Figure 6, specify the following software settings:

- Standard Input (STDIN) \rightarrow RS232
- Standard Output (STDOUT) \rightarrow RS232
- Sample Application Selection \rightarrow Memory Test

Base Sys	tem Builder - Software !	5etup		?
Devices to	o use as standard input and s	standard output		
STD <u>I</u> N:	R\$232	•		
STD <u>O</u> UT:	RS232	<u>.</u>		
Sample ap	plication selection			
Select the include a	e sample C application that ye linker script.	ou would like to have gener	ated. Each application wil	d
🔽 Memo	ry <u>t</u> est			
Illustra	ate system aliveness and per	form a basic read/write test	to each memory in your sy	stem
Eeriph	neral selftest			
Perfor	m a simple self-test for each p	peripheral in your system.		
More Info		< <u>B</u> ack	<u>N</u> ext> <u>C</u> ar	ncel

Figure 6: Software Setup

• Click Next

Select the me	mory devices which will be used to hold the following program sections:
nstruction:	imb_cntir
<u>D</u> ata:	dimb_cntir 🗾
<u>S</u> tack/Heap:	dimb_cntir
WARNING f you have plu use a debugg	aced the Instruction or Data section of this program in an external memory, you mu er, bootloader, or ACE file to initialize memory before you can run this program!
WARNING f you have plu use a debugg	aced the Instruction or Data section of this program in an external memory, you mu ler, bootloader, or ACE file to initialize memory before you can run this program!
WARNING If you have pluse a debugg	aced the Instruction or Data section of this program in an external memory, you mu ler, bootloader, or ACE file to initialize memory before you can run this program!
WARNING If you have pl use a debugg	laced the Instruction or Data section of this program in an external memory, you mu ler, bootloader, or ACE file to initialize memory before you can run this program!
WARNING — If you have pl use a debugg	laced the Instruction or Data section of this program in an external memory, you mu ler, bootloader, or ACE file to initialize memory before you can run this program!
WARNING — If you have pl use a debugg	aced the Instruction or Data section of this program in an external memory, you mu ter, bootloader, or ACE file to initialize memory before you can run this program!
WARNING If you have pl use a debugg	laced the Instruction or Data section of this program in an external memory, you mu ler, bootloader, or ACE file to initialize memory before you can run this program!

Figure 7: Configure Memory Test Application

Using the Configure Memory Test Application dialog box as shown in Figure 7, specify the following software settings:

- $\bullet \quad \text{Instructions} \to \text{iImb_cntlr}$
- Data \rightarrow dlmb_cntlr
- Stack/Heap \rightarrow dlmb_cntlr
- Click Next

The completed system including the memory map will be displayed as shown in Figure 8. Currently the memory map cannot be changed or updated in the BSB. If you want to change the memory map you can do this in XPS.

<	VII	IN IV®
C	ΛIL	.IINA

omponents:
High Addr
0x4140FFFF
0x4060FFFF
0x4000FFFF
0x4002FFFF
0x4004FFFF
0x201FFFFF
onents:
High Addr
0x00001FFF
oonents:
High Addr
0x00001FFF

Figure 8: Completed Processor System

- Click *Generate* and then *Finish*, to complete the design.
- Select Start Using Platform Studio and click OK.

Review

The Base System Builder Wizard has created the hardware and software specification files that define the processor system. When we look at the project directory, shown in Figure 9, we see these as system.mhs and system.mss. There are also some directories created.

- data contains the UCF (user constraints file) for the target board.
- etc contains system settings for JTAG configuration on the board that is used when downloading the bit file and the default parameters that are passed to the ISE tools.
- pcores is empty right now, but is utilized for custom peripherals.

• TestApp_Memory - contains a user application in C code source, for testing the memory in the system.

Name 🔺	Size	Туре
🛅 data		File Folder
etc		File Folder
pcores		File Folder
TestApp_Memory		File Folder
🔟 system.bsb	3 KB	BSB File
🔟 system.mhs	7 KB	MHS File
🔟 system.mss	2 KB	MSS File
🗢 system.xmp	2 KB	Xilinx XPS File

Figure 9: Project Directory

Project Options

To see the project options that Base System Builder has configured select: $Project \rightarrow Project Options$. As shown in Figure 10, the device information is specified.

Target Device-				
Architecture	Device Size	Pac <u>k</u> age	Speed Grade	e
spartan3	• xc3s200 •	ft256	-4	•
Advanced Option	ns (Optional)			
Peripheral Repo	sitory			
			Bro <u>w</u> se	
Custom Makefile	(instead of XPS gen	erated Makefile)		
			Browse	

Figure 10: Project Options - Device and Repository

Select: *Hierarchy and Flow*. This window is shown in Figure 11. This window provides the opportunity to export the processor system into an ISE project as either the top level system or a sub-module design.

Implement Design in XPS (XPS Flows)	
Effort Level to Run FPGA Implementation Tools	
Xflow (single iteration)	•
Implement Design in ISE (Export to Project Navigator F	low: DEPRECATED)
Implement Design in ISE (Export to Project Navigator F ISE File C:\escalation\82 tests\proinav\system.ise	low: DEPRECATED)
Implement Design in ISE (Export to Project Navigator F ISE File C:\escalation\82_tests\projnav\system.ise Processor Design is a sub-module (Uppheck for top-le	low: DEPRECATED)
Implement Design in ISE (Export to Project Navigator F ISE File C:\escalation\82_tests\projnav\system.ise Processor Design is a sub-module (Uncheck for top-le	low: DEPRECATED)
Implement Design in ISE (Export to Project Navigator F ISE File C:\escalation\82_tests\projnav\system.ise Processor Design is a sub-module (Uncheck for top-le Top level instance name system_i	low: DEPRECATED)

Figure 11: Project Options - Hierarchy and Flow

Create or Import IP Peripheral

One of the key advantages of building an embedded system in an FGPA is the ability to include customer IP and interface that IP to the processor. This section of the tutorial will walk through the steps necessary to include a custom IP core.

- In XPS, select *Hardware* → *Create or Import Peripheral…* to open the Create and Import Peripheral Wizard.
- Click Next. Select Create templates for a new peripheral.

By default the new peripheral will be stored in the project_directory/pcores directory. This enables XPS to find the core for utilization during the embedded system development.

• Click *Next*. In the Create Peripheral – Name and Version dialog, enter custom_ip as the name of the peripheral. This is shown in Figure 12.

XILINX°

Indicate the na	me and version of yo	ur peripheral.			
nter the name of y	our peripheral. This r	name will be used as th	ne top HDL design entity		
Name: [custom_	ip				
Version: 1.00.a	Minor revision:	Hardware/Softwar	e compatibility revision:		
			e compatibility revision.		
		, <u> </u>			
Logical library nar	me: custom_ip_v1_00)_a			
All HDL files (eith the logical library YPS project whe	er created by you or named above. Any o re this peripheral is us	generated by this tool) ther logical libraries rel sed or in EDK reposito	used to implement this p ferred to in your HDL are pries indicated in the XPS	eripheral must be assumed to be av	compiled into vailable in the

Figure 12: Create Peripheral - Name and Version

- Click *Next*. In the Create Peripheral Bus Interface dialog, select On-Chip Peripheral Bus (OPB), as this is the bus to which the new peripheral will be connected.
- Click Next. The Create Peripheral IPIF Services dialog enables the selection of several services. For additional information regarding each of these services, select More Info. Select the User logic S/W register support option.

♥ Creat IPIF Se Indi	t e Periphera ervices icate the IPIF s	I - IPIF Services	heral.	<u>? ×</u>
Your pe decodin peripher On-chip Peripher	ripheral will be ig, this module ral.	connected to the OPB bus th also offers other commonly us Interrupt Controller Byte Steering RST Read FFO	rough the OPB IP interface (IPIF) module. ed services. Using these services may sig Basic slave service and support Common and typically required by most peripherals for operations like logic control, status report, and etc. [S/W reset and MIR] [User logic interrupt support [V] User logic S/W register support	Besides standard functions like address nificantly simplify the implementation of your Advance slave service and support Typically required by peripherals that need data buffering or multiple memory/address spaces access. Burst transaction support EIFO User logic address range support
il Bus	opb_ipif		Master service and support Typically required by complex peripher data transfers between regions. DMA C Simple mode C Eacket mode Scatter Gather User logic master support	als like Ethernet and PCI for command
More	Info		< <u>B</u> a	ck <u>N</u> ext> <u>C</u> ancel

Figure 13: Create Peripheral - IPIF Services

• Click **Next**. In the Create Peripheral – User S/W Register dialog, change the Number of software accessible registers to 4.

🖗 Create Peripheral - User 5/W Register	<u>?</u> ×
User S/W Register Configure the software accessible registers in your peripheral.	
The software accessible registers will be implemented in the user-logic byte, half-word or word boundaries. The following fields determine the	module of your peripheral. These registers are addressable on the characteristics of the registers.
Number of software accessible registers:	
Data width of each register: 32 💌 bit	
_ Write Mode	
Instead of the usual <i>acknowledge write</i> behavior, an alternative kin <i>posted write</i> behavior, the IPIF unconditionally acknowledges the w reduces latency and improves performance. When posted writes an data immediately to local storage.	d of write behavior, <i>posted write</i> , is also supported. Under the rite transactions to the OPB on the earliest clock cycle, thus e enabled, it is assumed that the custom user logic will retire the
Enable posted write behavior	
C Disable posted write behavior for normal acknowledged write be	havior
C Allow dynamic posted/acknowledged write behavior controlled	ay user logic (IP2Bus_PostedWrInh)
More Info	<u> </u>

Figure 14: Create Peripheral - User S/W Register

- Click *Next*. In the Create Peripheral IP Interconnect (IPIC).
- Click *Next*. In the Create Peripheral (OPTIONAL) Peripheral Simulation Support dialog, a Bus Functional Model (BFM) simulation environment can be generated. This tutorial will not cover BFM simulation. Leave the option unchecked.
- Click *Next*. In the Create Peripheral (OPTIONAL) Peripheral Implementation Support dialog, uncheck the Generate ISE and XST project files to help you implement the peripheral using XST flow.
- Click *Next* and then *Finish*.

The Create or Import Peripheral Wizard creates a new directory called custom_ip_v1_00_a in the pcores directory. This new directory contains the following:

Name 🔺	Size Type
ata	File Folder
devl	File Folder
🛅 hdl	File Folder

Figure 15: Custom IP Directory Structure

The following is a description of the files located in each directory:

HDL source file(s)

- MB_tutorial\pcores\custom_ip_v1_00_a\hdl
 - vhdl/custom_ip.vhd

This is the template file for your peripheral's top design entity. It configures and instantiates the corresponding IPIF unit in the way you indicated in the wizard GUI and connects it to the stub user logic where the user logic should get implemented. You are not expected to modify this template file except in certain marked places for adding user specific generics and ports.

vhdl/user_logic.vhd

This is the template file for the stub user logic design entity, either in VHDL or Verilog, where the actual functionalities should get implemented. Some sample code may be provided for demonstration purpose.

- XPS interface file(s)
- MB_tutorial\pcores\ custom_ip_v1_00_a\data
 - custom_ip_v2_1_0.mpd

This Microprocessor Peripheral Description file contains interface information of your peripheral so that other EDK tools can recognize the peripheral.

custom_ip_v2_1_0.pao

This Peripheral Analysis Order file defines the analysis order of all the HDL source files that are used to compile your peripheral.

Driver source file(s)

MB_tutorial\drivers\ custom_ip_v1_00_a\src:

custom_ip.h

This is the software driver header template file, which contains address offsets of software addressable registers in your peripheral, as well as some common masks, simple register access macros and function declarations.

- custom_ip.c
 - This is the software driver source template file to define all applicable driver functions.

custom_ip_selftest.c

This is the software driver self test example file which contain self test example code to test various hardware features of your peripheral.

makefile

This is the software driver makefile to compile drivers.

Now that the template has been created, the user_logic.vhd file must be modified to incorporate the custom IP functionality.

- Open user_logic.vhd. Currently the code provides an example of reading and writing to four 32-bit registers. For the purpose of this tutorial, this code will not be modified.
- Close user_logic.vhd

In order for XPS to add the new custom IP core to the design, the pcores directory must be rescanned. This can be accomplished by selecting $Project \rightarrow Rescan User Repositories$. XPS also automatically rescans the pcores directory when the project is opened.

Design Modification using Platform Studio

Once a design has been created with the Base System Builder, it can be modified from within the System Assembly view.

PMM	Name	Bus Connection	IP Type	IP Version
	🗄 👁 microblaze_0		microblaze	5.00.a
	_ ⊕		opb_v20	1.10.c
] 🗄 👁 ilmb		lmb_v10	1.00.a
] Ė≪dimb		lmb_v10	1.00.a
] ∰. ≪>debug_module		opb_mdm	2.00.a
	t		Imb_bram_if_cntlr	2.00.a
	🖶 🗢 ilmb_cntlr		lmb_bram_if_cntlr	2.00.a
			opb_uartlite	1.00.Ь
I	🛓 🗄 - 🧼 LED s_88 it		opb_gpio	3.01.Ь
I	⊕- → Push_Buttons_3Bit		opb_gpio	3.01.Ь
I	🗄 👁 DIP_Switches_8Bit		opb_gpio	3.01.Ь
I			opb_emc	2.00.a
	i∰. ≪>Imb_bram		bram_block	1.00.a
		0	util_bus_split	1.00.a
	t≟⊷ ≪>dcm_0		dcm_module	1.00.a

Figure 16: System Assembly View

Double clicking on any of the IP' s listed in the System Assembly View allows modification of that particular IP. The System Assembly View has the following filters:

Bus Interface filter: With the Bus Interface activated, the patch panel to the left of the System Assembly View gets activated. The bus connectivity of the core is shown when the hierarchy of the IP is expanded.

<u>Ports filter</u>: With this filter on, the port connections appear when the hierarchy of the IP is expanded. You need to activate this filter to be able to add external ports.

<u>Addresses filter</u>: The IP' s addresses can be viewed when expanding the IP. This is where you can generate addresses for the IP' s.

The IP Catalog tab shows all of the IP that is available to use in the EDK project. To add new IP:

- Bring the IP Catalog tab forward.
- Expand the Project Repository hierarchy
- Drag and drop the IP into the System Assembly View or double click on the IP.



Figure 17: Inserting IP

With the Bus Interface filter still activated:

- Press the Connection Filter button and select All
- Expand the custom_ip_0 instance
- Highlite the slave OPB connection (SOPB)
- Select the No Connection pull down menu and change it to mb_opb

Name	Bus Connection	IP Type	IP Version
Ė. ∞microblaze_0		microblaze	4.00.a
Ė≪mb_opb		opb_v20	1.10.c
🗄 - 🧼 ilmb		lmb_v10	1.00.a
🗄 🧼 dlmb		lmb_v10	1.00.a
🗄 🗢 👁 debug_module		opb_mdm	2.00.a
🗄- 🗢 dimb_cntir		lmb_bram_if_cr	ntlr 1.00.b
🗄 👁 ilmb_cntlr		lmb_bram_if_cr	ntlr 1.00.6
🗄 🗢 RS232		opb_uartlite	1.00.Ь
🗄 - 🧼 LED s_8Bit		opb_gpio	3.01.Ь
🗄 - 🗢 Push_Buttons_3Bit		opb_gpio	3.01.Ь
🗄 👁 DIP_Switches_8Bit		opb_gpio	3.01.Ь
🗄 - 🗢 SRAM_256Kx32		opb_emc	2.00.a
Ė≪custom_ip_0		custom_ip	1.00.a
SOPB	mbopb 💌]	
🗄 🗢 Imb_bram		bram_block	1.00.a
🗄 - SRAM_256Kx32_util_bus_	_split_0	util_bus_split	1.00.a
Ė⊷≪>dcm_0		dcm_module	1.00.a



- Now select the Ports filter
- Press the Connection Filter button and select All
- Expand the custom_ip_0 instance
- Highlite the OPB_Clk port
- Select the Default Connection pull down menu and change the clock connection to sys_clk_s

The state of the second				
The SPAM 256K 22				
E-Scustom ip 0				
OPB_CIk	svs clk s		Clk	
···· OPB_Rst	Default Connection (OPB_Rst)	1	Rst	
SI_DBus	Default Connection (SI_DBus)	0		[0:1
SI_errAck	Default Connection (SI_errAck)	0		
SI_retry	Default Connection (SI_retry)	0		
SI_toutSup	Default Connection (SI_toutSup) (
SI_xferAck	Default Connection (SI_xferAck) ()		
···· OPB_ABus	Default Connection (OPB_ABus	s)		[0:1
···· OPB_BE	Default Connection (OPB_BE)	I		[0:1
OPB_DBus	Default Connection (OPB_DBu:	s)		[0:1
OPB_RNW	Default Connection (OPB_RNW	/) [
NPR select	Default Connection (NPR selec	નો I		

Figure 19: Changing port connections

Note: Right clicking on the Name column in the System Assembly View provides more filtering options.

Select the Addresses filter to define an address for the newly added custom_ip peripheral. The address can be assigned by entering the Base Address or the tool can assign an address. For the purpose of this tutorial, the tool will be used to assign an address.

- Change the size if the dlmb_cntlr and ilmb_cntlr to 8K.
- Click Generate Addresses.

A message in the console window will state that the address map has been generated successfully. The design is now ready to be implemented.

Implementing the Design

Now that the hardware has been completely specified in the MHS file, you can run the Platform Generator. Platform Generator elaborates the MHS file into a hardware system consisting of NGC files that represent the processor system. Then the Xilinx ISE tools will be called to implement the design for the target board. To generate a netlist and create the bit file, follow these steps:

- Start ISE by selecting Start \rightarrow Programs \rightarrow Xilinx ISE 8.2i \rightarrow Project Navigator.
- In ISE, select $File \rightarrow New Project$ to create a new Project Navigator project.

• In the New Project dialog box shown in Figure 20, browse to the XPS project directory and then enter the Project Name, project_navigator.

New Project Wizard - Create New Project		
Enter a Name and Location for the Project Project Name:	Project Location	
project_navigator	H:\edk_tutorials\edk_81\mb\MB_tutorial\project_nav	í <u></u>
Select the Type of Top-Level Source for the Project-		
Top-Level Source Type:		
HDL		•
More Info	< Back Next > C	ancel
More Info	< Back Next > C	and

Figure 20: ISE New Project

• Click **Next**. Configure the Device and Design flow as shown in Figure 21. It should be noted that these settings need to be consistent with the XPS project settings.

Property Name	Value	
Product Category	All	•
Family	Spartan3	•
Device	XC3S200	•
Package	FT256	•
Speed	-4	•
Top-Level Source Type	HDL	<u>.</u>
Synthesis Tool	XST (VHDL/Verilog)	•
Simulator	Modelsim (PE/SE) Mixed	•
Enable Enhanced Design Summary		
Enable Message Filtering		
Display Incremental Messages		

Figure 21: New Project - Device and Design Flow

- Click *Next*. ISE has the ability to add an XPS project file as a new source file. However, the tutorial will not cover this aspect.
- Browse up into the XPS project and add the system.xmp in the New Project Wizard Add Existing Sources dialog window.
- Deselect the Copy to Project checkbox
- Click Next
- Click Finish
- Click OK
- Select the system.xmp source file and double click on the View HDL Instantiation Template.

Once the process has completed the editor window will contain the instantiation template called system.vhi.

 In ISE, select *Project* → *New Source*. Select the VHDL module and name it system_stub.vhd in the project_navigator directory. Then instantiate the system.vhi into system_stub.vhd:

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating

---- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity system_stub is

port (

fpga_0_RS232_RX_pin : in std_logic;

fpga_0_RS232_TX_pin : out std_logic;

fpga_0_LEDs_8Bit_GPIO_d_out_pin : out std_logic_vector(0 to 7);

fpga_0_Push_Buttons_3Bit_GPIO_in_pin : in std_logic_vector(0 to 2);

fpga_0_DIP_Switches_8Bit_GPIO_in_pin : in std_logic_vector(0 to 7);

fpga_0_SRAM_256Kx32_Mem_A_pin : out std_logic_vector(12 to 29);

fpga_0_SRAM_256Kx32_Mem_DQ_pin : inout std_logic_vector(0 to 31);

fpga_0_SRAM_256Kx32_Mem_OEN_pin : out std_logic_vector(0 to 0);

fpga_0_SRAM_256Kx32_Mem_CEN_pin : out std_logic_vector(0 to 0);

fpga_0_SRAM_256Kx32_Mem_CEN_1_pin : out std_logic_vector(0 to 0);

fpga_0_SRAM_256Kx32_Mem_WEN_pin : out std_logic;

fpga_0_SRAM_256Kx32_Mem_BEN_pin : out std_logic_vector(0 to 3);

sys_clk_pin : in std_logic;

sys_rst_pin : in std_logic

);

end system_stub;

architecture Behavioral of system_stub is

COMPONENT system

PORT(

fpga_0_RS232_RX_pin : IN std_logic;

fpga_0_Push_Buttons_3Bit_GPIO_in_pin : IN std_logic_vector(0 to 2);

fpga_0_DIP_Switches_8Bit_GPIO_in_pin : IN std_logic_vector(0 to 7);

sys_clk_pin : IN std_logic;

sys_rst_pin : IN std_logic;

fpga_0_SRAM_256Kx32_Mem_DQ_pin : INOUT std_logic_vector(0 to 31);

fpga_0_RS232_TX_pin : OUT std_logic;

fpga_0_LEDs_8Bit_GPIO_d_out_pin : OUT std_logic_vector(0 to 7);

fpga_0_SRAM_256Kx32_Mem_A_pin : OUT std_logic_vector(12 to 29);

fpga_0_SRAM_256Kx32_Mem_OEN_pin : OUT std_logic_vector(0 to 0);

fpga_0_SRAM_256Kx32_Mem_CEN_pin : OUT std_logic_vector(0 to 0);

fpga_0_SRAM_256Kx32_Mem_CEN_1_pin : OUT std_logic_vector(0 to 0);

fpga_0_SRAM_256Kx32_Mem_WEN_pin : OUT std_logic;

fpga_0_SRAM_256Kx32_Mem_BEN_pin : OUT std_logic_vector(0 to 3)

);

END COMPONENT;

begin

Inst_system: system PORT MAP(

fpga_0_RS232_RX_pin => fpga_0_RS232_RX_pin,

fpga_0_RS232_TX_pin => fpga_0_RS232_TX_pin,

fpga_0_LEDs_8Bit_GPIO_d_out_pin => fpga_0_LEDs_8Bit_GPIO_d_out_pin,

fpga_0_Push_Buttons_3Bit_GPIO_in_pin => fpga_0_Push_Buttons_3Bit_GPIO_in_pin,

fpga_0_DIP_Switches_8Bit_GPIO_in_pin => fpga_0_DIP_Switches_8Bit_GPIO_in_pin,

fpga_0_SRAM_256Kx32_Mem_A_pin => fpga_0_SRAM_256Kx32_Mem_A_pin,

fpga_0_SRAM_256Kx32_Mem_DQ_pin => fpga_0_SRAM_256Kx32_Mem_DQ_pin,

fpga_0_SRAM_256Kx32_Mem_OEN_pin => fpga_0_SRAM_256Kx32_Mem_OEN_pin(0 to 0),

fpga_0_SRAM_256Kx32_Mem_CEN_pin => fpga_0_SRAM_256Kx32_Mem_CEN_pin(0 to 0),

fpga_0_SRAM_256Kx32_Mem_CEN_1_pin => fpga_0_SRAM_256Kx32_Mem_CEN_1_pin(0 to 0),

fpga_0_SRAM_256Kx32_Mem_WEN_pin => fpga_0_SRAM_256Kx32_Mem_WEN_pin,

fpga_0_SRAM_256Kx32_Mem_BEN_pin => fpga_0_SRAM_256Kx32_Mem_BEN_pin,

```
sys_clk_pin => sys_clk_pin,
sys_rst_pin => sys_rst_pin
);
```

end Behavioral;

By creating system_stub.vhd to the Project Navigator project the hierarchy is updated as shown in Figure 22.

Sources for: Sy	nthesis/Implement	ation	×
© project_ <mark> xc3s200</mark> 1 1 €	navigator -4ft256 system_stub - STRL system_i - system (.	JCTURE (system_ ./system.xmp)	stub. vhd)
। बार्ड Sources	👸 Snapshots	Libraries	

Figure 22: Project Navigator Project Hierarchy

- In ISE, select *Project* \rightarrow *Add Source*. Select the system.ucf file in the <xps_project>\data directory.
- Select system_stub.vhd and double click on Generate Programming File to implement the design and generate a bit file.

ISE will call XPS to generate the EDK to create the following directories:

- $\circ~$ hdl contains the VHDL files that define the processor system
- o implementation contains the NGC files
- synthesis contains the projects and information from synthesizing the files in the hdl directory to create those in the implementation directory

Defining the Software Design

Now that the hardware design is completed, the next step is defining the software design. There are two major parts to software design, configuring the Board Support Package (BSP) and writing the software applications. The configuration of the BSP includes the selection of device drivers and libraries.

Configuration of the BSP

Configuration of the BSP is done using the Software Platform Settings dialog. In XPS, select **Software** \rightarrow

Software Platform Settings... This will open the Software Platform Settings dialog box as shown in Figure 23.

The Software Platform Settings dialog box contains four views. Each of these views is used to control all aspects of the BSP creation.

The Software Platform view allows the user to modify processor parameters, driver, operating system and libraries. The following Operating Systems are supported:

- o Standalone
- o xilkernel
- o uclinux
- o nucleus

No changes are required in this view.

Software Platform DS and Libraries Drivers	CPU D Proces	Processor Settings river: cpu 💌 sor Parameters:	CPU Dri	ver Version: 1.00.a 💌		
nterrupt Handlers	Name	,	Current V-	alue Default Value	Туре	Description
		— xmdstub_peripheral — archiver — compiler_flags — archiver — compiler	debug_mo -g mb-ar mb-acc	dule none -g mb-ar	peripheral_in: string string	stance Debug peripheral to be used with xmdstub Extra compiler flags used in BSP and library genera Archiver used to archive libraries for both BSP general Compiler used to acchive libraries or both BSP general libraries are
	os: [OS & Library Settings	Version: 1.0	0.a 🗾 Default	sung software platform.	Provides basic processor related functions and basic
	↓] OS: [Use	OS & Library Settings standalone OS Library Versi	Version: 1.C	0.a Default	sung software platform. functions such as	Provides basic processor related functions and basic
	Use	OS & Library Settings standalone S Library Versio xilnet 2.00.4	Version: 1.C	0.a Default Description Xilinx Networking TCP/IP s	software platform. Junctions such as	Provides basic processor related functions and basic
	OS:	OS & Library Settings standalone S Library Versic xilnet 2.00.4 xilmfs 1.00.4	Version: [1.0	0.a Default Description Xilinx Networking TCP/IP s Xilinx Memory File System	software platform. Software platform. Sunctions such as	Provides basic processor related functions and basic
	Use □	OS & Library Settings standalone S Library Versic xilnet 2.00.a xilmfs 1.00.a xilfile 1.00.a	Version: 1.0	0.a Default Os like Description Xilinx Networking TCP/IP s Xilinx Memory File System Provides file system call acc	software platform. iunctions such as ack library	Provides basic processor related functions and basic
		OS & Library Settings standalone S Library Versic xilnet 2.00. xilmfs 1.00. xilfile 1.00. xilfile 1.00.	Version: 1.0	Description Zilinx Networking TCP/IP s Xilinx Memory File System Provides file system call acc Provides read/write routine:	software platform. iunctions such as ack library ess to simple seria s to access files str	Provides basic processor related functions and basic

Figure 23: Software Platform Settings Dialog

• Select the OS and Libraries view as shown in Figure 24. This view allows the user to configure OS and library parameters. No changes are required.

Processor Informatio Processor Instance:	n microblaze_0 💌				
Software Platform	Configuration for OS: standalone v1.00.a	3			
Do and Elbranes	Name	Current Value	Default Value	Туре	Description
Drivers	🗄 - standalone				
Interrupt Handlers	need_xil_malloc	false	false	bool	Is xil_malloc required ?
	stdout	RS232	none	peripheral_in	stance stdout peripheral
	stdin	RS232	none	peripheral_in	stance stdin peripheral
	<u> <u> </u> </u>	false	false	bool	Predecode FPU exceptions and save o
	±. enable_sw_intrusive_profiling	false	false	bool	Enable S/W Intrusive Profiling on Hardv

Figure 24: OS and Libraries view

• Select the Drivers view. This view allows you to select the software versions for the peripherals in the system as shown in Figure 25. Notice that the driver version is independent of the HW version.

Processor Informa Processor Instanc	tion e: microblaze_0	2				
Software Platform	Drivers Configuration	n:				
OS and Libraries	Peripheral	HW version	Instance	Driver		Version
Drivers	opb_v20	1.10.c	mb_opb	opbarb	-	1.02.a
Interrupt Handler:	opb_mdm	2.00.a	debug_module	uartlite	•	1.00.Ь
	Imb_bram_if_cntlr	1.00.Ь	dimb_cntir	bram	•	1.00.a
	Imb_bram_if_cntlr	1.00.Ь	ilmb_cntlr	bram		1.00.a
	opb_uartlite	1.00.Ь	RS232	uartlite	•	1.00.Ь
	opb_gpio	3.01.Ь	LEDs_8Bit	gpio	-	2.00.a 💌
	opb_gpio	3.01.Ь	Push_Buttons_3Bit	gpio	•	2.00.a 💌
	opb_gpio	3.01.Ь	DIP_Switches_8Bit	gpio		2.00.a 💌
	opb_emc	2.00.a	SRAM_256Kx32	56Kx32 emc 💽 2.		2.00.a
	custom_ip	1.00.a	custom_ip_0	custom ip	•	1.00.a

Figure 25: Drivers view

The Interrupt Handlers view allows you to modify the parameters for the interrupts. This project does not have any interrupts.

- Click OK
- In XPS, select Software → Generate Libraries and BSPs to run LibGen and create the BSP which
 includes device drivers, libraries, configures the STDIN/STDOUT, and Interrupt handlers associated with
 the design.

LibGen creates the following directories in the microblaze_0 directory, shown in Figure 26:

- o code: contains the compiled and linked application code in an ELF file
- include: contains the header files for peripherals included in the design (such as xgpio.h and xuartlite.h)
- o lib: contains the library files (such as libc.a and libxil.a)
- o libsrc: contains the source files used to create libraries

Note: For more information on these files, refer to the Embedded System Tools Guide.

ile Folder
ile Folder
ile Folder
ile Folder

Figure 26: MicroBlaze Drivers Directories

Building the User Application

In EDK 8.2, XPS provides the ability for the user to create multiple software projects. These projects can include source files, header files, and linker scripts. Unique software projects allow the designer to specify the following options for each software project:

- o Specify compiler options
- o Specify which projects to compile
- o Specify which projects to download
- o Build entire projects

Software application code development can be managed by selecting the Applications tab as shown in Figure 27. The Base System Builder (BSB) generates a sample application which tests a subset of the peripherals included in the design.

Project	Applications	IP Catalog	
Software F	Projects		
	d Software Appli	cation Project	
🕅 Del	fault: microblaze	_0_bootloop	
🎆 Del	fault: microblazej	_0_xmdstub	
🗄 - 🎆 Pro	oject: TestApp	_Memory	
🕂 • Pro	ocessor: microbla	aze_0	
Ex	ecutable: H:\edl	<_tutorials\edk_	81\mb\MB_tutorial\TestApp_Memory\executable.elf
ģ-Co	mpiler Options		
	Linker Script: H	l://edk_tutorial:	:\edk_81\mb\MB_tutorial\TestApp_Memory\src\TestApp_Mei
	Mode: EXECU	TABLE	
	Stack Size:		
	Heap Size:		
Ę-So	urces		
	H:\\edk_tutori	als\edk_81\mb\	MB_tutorial\TestApp_Memory\src\TestApp_Memory.c
iHe	eaders		
•			

Figure 27: Applications Tab

Compiling the Code

Using the GNU GCC Compiler, compile the application code as follows:

• Select Software → Build All User Applications to run mb-gcc. Mb-gcc compiles the source files.



Figure 28: XPS Output Window - Software Compiled

Downloading the Design

Now that the hardware and software designs are completed, the device can be configured. Follow these steps to download and configure the FGPA:

• Connect the host computer to the target board, including connecting the Parallel-JTAG cable and the serial cable.

- Start a hyperterminal session with the following settings:
 - o com1 This is dependant on the com port your serial cable is connected to.
 - o Bits per second:57600
 - Data bits: 8
 - o Parity: none
 - o Stop bits: 1
 - o Flow control: none
- Connect the board power
- In ISE, Select system_stub.vhd in the source window
- In the process window, double click on Update Bitstream with Processor Data
- In the process window, double click on Configure Device (iMPACT) under Generate Programming File
- With iMPACT, configure the FPGA using system_stub_download.bit located in the project_navigator directory choosing to bypass all of the other chips in the JTAG chain

After the configuration is complete, you should see a display similar to that in shown in Figure 29:

Demo - HyperTerr File Edit View Call	minal Iransfer Help DA 😰				
Enterin Starting M Running Running Running Exiting	g main() emoryTest 32-bit te 16-bit te 8-bit tes main() -	 for SRAM stPASS stPASS tPASSE -	1_256K× SED ! SED ! ED !	32:	
Connected 0:42:58	Auto detect	57600 8-N-1	SCROLL	CAPS	

Figure 29: Hyperterminal Output

Debugging the Design

Now that the device is configured, you can debug the software application directly via the MDM interface. GDB connects to the MicroBlaze core through the MDM and the Xilinx Microprocessor Debug (XMD) engine utility as shown in Figure 30. XMD is a program that facilitates a unified GDB interface and a TCL (Tool Command Language) interface for debugging programs and verifying microprocessor systems. The XMD engine is used with MicroBlaze and PowerPC GDB (mb-gdb & powerpc-eabi-gdb) for debugging. Mb-gdb and powerpc-eabi-gdb

communicate with XMD using the remote TCP protocol and control the corresponding targets. GDB can connect to XMD on the same computer or on a remote Internet computer.

To debug the design, follow these steps:

- In XPS, select *Debug* → *XMD Debug Options*. The XMD Debug Options dialog box allows the user to specify the connections type and JTAG Chain Definition. Three connection types are available for MicroBlaze:
 - Simulator enables XMD to connect to the MicroBlaze ISS
 - Hardware enables XMD to connect to the MDM peripheral in the hardware
 - Stub enables XMD to connect to the JTAG UART or UART via XMDSTUB
 - Virtual platform enables a Virtual (C model) to be used (not covered in this tutorial)
- Verify that Hardware is selected.
- Select Save.
- Select **Debug** → Launch XMD.

12

```
ex C:\EDK\bin\nt\xmd.exe
                                                                                                                                                                                                                                                           - 🗆 ×
 Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 8.1 Build EDK_I.18.4
Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.
 XMD%
Loading XMP File..
Processor(s) in System ::
Microblaze(1) : microblaze_0
Address Map for Processor microblaze_0
(0x00000000-0x00001fff) dlmb_cntlr dlmb
(0x20010000-0x00001fff) ilmb_cntlr ilmb
(0x20010000-0x2001fff) SRAM_256Kx32 mb_opb
(0x40000000-0x4000ffff) Push_Buttons_3Bit
(0x40020000-0x4002ffff) LEDs_BBit mb_opb
(0x40040000-0x4004ffff) DIP_Switches_8Bit
(0x40600000-0x4060ffff) RS232 mb_opb
(0x41400000-0x4140ffff) debug_module mb_opb
(0x78400000-0x7840ffff) custom_ip_0 mb_opb
                                                                                                                                                               mb_opb
                                                                                                                                                               mb_opb
 Connecting to cable (Parallel Port - LPT1).

Checking cable driver.

Driver windrvr6.sys version = 7.0.0.0. LPT base address = 0378h.

ECP base address = 0778h.

ECP hardware is detected.

Cable connection established.

Connecting to cable (Parallel Port - LPT1) in ECP mode.

Checking cable driver.

Driver xpc4drvr.sys version = 1.0.4.0. LPT base address = 0378h.

Cable Type = 1, Revision = 0.

Setting cable speed to 5 MHz.

Cable connection established.
  JTAG chain configuration
                               ID Code
01414093
                                                                                IR Length
                                                                                                                           Part Name
XC3S200
  Device
                                                                                              6
                               05045093
                                                                                                                           XCFØ2S
  Assuming, Device No: 1 contains the MicroBlaze system
Connected to the JTAG MicroProcessor Debug Module (MDM)
  No of processors =
                                                               1
  MicroBlaze Processor 1 Configuration :
 ...4.00.a
Instruction Cache Support.....off
Data Cache Support....off
Exceptions Support....off
FPU Support.....off
FSL DCache Support.....off
Hard Divider Support.....off
Hard Multiplier Support.....off
MSR clr/set Instruction Support....off
JTAG MDM Connected to MicroBlaze 1
Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD2.
   XMD% _
```



- In XPS, select **Debug** \rightarrow **Launch Software Debugger** to open the GDB interface.
- In GDB, select **File** \rightarrow **Target Settings** to display the Target Selection dialog box as shown in Figure 31.
- Click OK.

XILINX[®]

		🔽 Set breakpoint at 'main'
Connecti	on	- V Set breakpoint at 'exit'
arget:	Remote/TCP : XMD	
lostname:	localhost	Set breakpoint at
ort:	1234	Display Download Dialog
		Use xterm as inferior's tty
Maura Orakia		

Figure 31: GDB Target Selection

- In GDB, select $File \rightarrow Open...$
- Select executable.elf in the TestApp_Memory directory. The C code is visible because the Create symbols for debugging (-g option) is selected by default in the compiler options.
- In GDB, select $File \rightarrow Exit$.
- In the Applications window of XPS, double click on the Project: TestApp_Memory label.
- In the Debug and Optimization tab set the Optimization Level to No Optimization
- Click OK.

	Debug and Uptimization Path	s Advanced
Optimization	n Parameters	
Optimizatio	n Level No Optimization 💌	
🖵 Use Glo	obal Pointer Optimization	
🔽 Genera	ate Debug Symbols	
Create	Symbols for Debugging (-g option)	
C Create	Symbols for Assembly (-gstabs option	1)
Note: If bot information	th optimization level and debug optio may not correlelate to source code.	n are set, the

Figure 32: Compiler Options

- Recompile the code
- Load the new executable.elf into GDB

• Select $Run \rightarrow Run$

There is an automatic breakpoint at main. GDB allows you to single step the C or assembly code.

Note: The default values displayed in the Registers Window are in hex, while the values displayed in the Source Window are in decimal.

Performing Behavioral Simulation of the Embedded System

Performing a behavioral simulation of the complete system, which includes the embedded processor system, is a powerful verification technique. In order to perform a behavioral simulation of the complete system in ISE, the simulation file for the embedded system must be generated.

First, increase the Baud rate of the UART so that simulation of the UART can happen more quickly. Remember to change the Baud rate value back to 57600 before downloading to the Spartan3 demo board.

- In XPS double-click on the MHS file
- Change the value of PARAMETER C_BAUDRATE to 3125000 (value of C_CLK_FREQ/16)
- Save the MHS file and close it
- XPS, select *Project* → *Project Options*. In the Project Options dialog box select the HDL and Simulation tab.

Browse to the precompiled EDK Library and Xilinx Library as shown in Figure 33. It should be noted that the paths will be different to match you system. For additional information on compiling the simulation libraries refer to the Embedded System Tools Reference Manual Chapter 3.

X

~		
Project Options		
Device and Repository Hierarchy and Flow	HDL and Simulation	
HDL C Verilog	-Simulator Compile Scrip • ModelSim • NC	t Sim C None
Simulation Libraries Path <u>E</u> DK Library (Output Directory of compedklib)	L 	
C:/mti_simulation_libraries/edk		Bro <u>w</u> se
⊠ilinx Library (Output Directory of compxlib)		
C:/mti_simulation_libraries/ise		Bro <u>w</u> se
If libraries have not been compiled:	npile	
Simulation Models		
Behavioral C Structural	C Timing	
Allow Mixed Language Behavioral Files		

Figure 33: Project Options - HDL and Simulation tab

- Click Ok.
- Select *Simulation* → *Generate Simulation HDL Files*. This will generate all of the EDK HDL Simulation files in the EDK\simulation\behavioral directory created by SimGen.
- Now that the EDK simulation files have been created, the ISE simulation environment can be created.
- In ISE, select system_stub.vhd and double click on Create New Source in the Process Window.
- In the New Source dialog, select the source type as "VHDL Test Bench" and the File Name as "testbench"
- Click *Next*. Select system_stub as the source file to which the testbench will be associated.
- Click *Next* and *Finish*.

Now select Behavioral Simulation in the Sources window as shown in Figure 34.

Sources for: Synthesis/Implementation proje Synthesis/Implementation Synthesis/Implementation Behavioral Simulation Post-Translate Simulation Post-Map Simulation Post-Map Simulation Post-Route Simulation System.ucr (/uata/system.ucr)			<u>}</u>
Synthesis/Implementation Synthesis/Implementation Behavioral Simulation Post-Translate Simulation Post-Map Simulation Post-Route Simulation system.ucr (/uata/system.ucr)	Sources for:	Synthesis/Implementation	
xc3s Behavioral Simulation Post-Translate Simulation Post-Map Simulation Post-Route Simulation Post-Route Simulation Post-Route Circulator System.	🤄 🗑 proje	Synthesis/Implementation	
Post-Translate Simulation Post-Map Simulation Post-Route Simulation Post-Route Simulation Post-Route Circulatarsystem.ucr	⊨- 🖾 xc3s	Behavioral Simulation	
Post-Map Simulation Post-Route Simulation		Post-Translate Simulation	
Post-Route Simulation		Post-Map Simulation	
······ [···· [···· system.uci (7uata/system.uci)		Post-Route Simulation	
		re system.ucr (/uata/system.ucr)	

Figure 34: Behavioral Simulation View

Testbench.vhd will now open in the ISE Editor Window.

Scroll to the bottom of the file and remove the following code:

tb : PROCESS
BEGIN

-- Wait 100 ns for global reset to finish wait for 100 ns;

-- Place stimulus here

wait; -- will wait forever END PROCESS;

Add the following code:

tb_clk : PROCESS

BEGIN

sys_clk_pin <= '1'; wait for 10 ns; sys_clk_pin <= '0'; wait for 10 ns; END PROCESS;

tb_reset : PROCESS

BEGIN

sys_rst_pin <= '1'; wait for 5 us;</pre>

sys_rst_pin <= '0'; wait;</pre>

END PROCESS;

fpga_0_RS232_RX_pin <= fpga_0_RS232_TX_pin;

In order to populate the BRAMs with the TestApp_Memory Application, a configuration statement must be created. Add the following after the testbench architecture:

```
configuration testbench_vhd_conf of testbench_vhd is
for behavior
for uut: system_stub
for Behavioral
for Inst_system: system
use configuration work.system_conf;
end for;
end for;
end for;
end for;
save and close the testbench.vhd file.
```

Select testbench.vhd in the ISE Source Window. Expand the ModelSim Simulator in the process window then right-click on the Simulate Behavioral Model and select Properties.

Change the simulation run time to 0ns, select Use Configuration Name and insert testbench_vhd_conf in the Configuration Name field as shown in Figure 35.

Simulation Run Time	Ons	
Simulation Resolution	Default (1 ps)	-
VHDL Syntax	93	•
Use Explicit Declarations Only		
Use Configuration Name		
Configuration Name	testbench_vhd_conf	



Click on the OK button.

Double-click on the Simulate Behavioral Model to simulate your processor design.

To see the output of the UART, type in the following command in the Modelsim console window:

add wave -radix ascii /testbench_vhd/uut/ inst_system /rs232/rs232/opb_uartlite_core_i/opb_uartlite_tx_i/fifo_dout

At the command prompt type "run 300us" to begin running the simulation. It will take several thousand uS to run the design to simulate the functionality of the design because of the printf routines. You should see a Modelsim wave form similar to the one shown in Figure 36.

💼 wave - default																+ 0	
/testbench_vhd/fpg	1					UNU		Ŵ			ЛПЦ						\Box
/testbench_vhd/sys	1				1			1									
/testbench_vhd/sys	U				_			_			_						
	000	000				_					_						
⊕- ✓ /testbench_vhd/tpg		000000			_												
/testbench_vhd/tpg								IU LUL			┘╵╹						
															_		
⊕- ✓ /testbench_vhd/tpg		00000.		*****	(†))))) † †)))	1)1	≬ı ÿ	• •	<u> </u>))	<u> </u>		
		<u>[</u>															
⊕→ /testbench_vhd/tpg		<u> </u>															
					_			_									
/testbench_vhd/tpg																	
		<u>um</u>				·	υ									-vv	
		l	_ <u>}-</u>	<u> , , F</u>	<u>;]n]t</u>)e j	<u>r li lu</u>	ya y	<u>lm_</u> }	عابا	ᆡᆜ		<u>}-</u>]	J	787	
																	17
Now	30000000 ps		11111			1111					111		1111				
Eursor 1	Û ne	, [0 ps]					4(Jus							00	us	
					C											- NJ	-
																~ ~	1
wave wave																	<u>«</u> »

Figure 36: Simulation results