AutoESL Tutorial: Integrating with EDK

UG889 (v2012.1) April 24, 2012

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
14/24/2012	2012.1	Initial Xilinx release.

Table of Contents

AESL AXI-LITE Example for EDK

Introduction

Document Overview

This document describes how to create an Embedded Developer Kit (EDK) PCORE with an AXI-LITE interface from the AutoESL high-level synthesis tool and shows the steps needed to integrate the generated PCORE with the MicroBlaze processor using the Xilinx Platform Studio (XPS) Tool Suite.

The reference design has been verified on the Avnet Spartan-6 LX9 MicroBoard.

Figure 1-1 shows the Avnet MicroBoard kit.

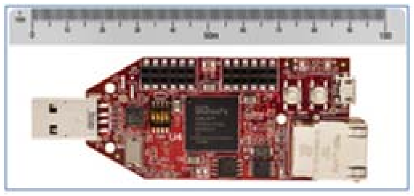


Figure 1-1: **Avnet Spartan-6 LX9 MicroBoard**

Software Requirements

The following software is required to test this reference design:

- Xilinx ISE WebPACK with the EDK add-on or ISE Embedded Edition version 14.1
- Installed Silicon Labs CP210x USB-to-UART Bridge Driver (see *Silicon Labs CP210x USB-to-UART Setup Guide*, listed at http://em.avnet.com/s6microboard)
- AutoESL version 2011.4.2

Reference Design

Design Overview

The reference design consists of an EDK MicroBlaze processor with custom PCORE generated from the AutoESL tool.

The reference design, AXI_Lite_Interface, can be copied from the examples/tutorial directory in the AutoESL installation area.

The MicroBlaze processor based design was created using the XPS Base System Builder. Figure 1-2 shows the final design created and provided with this document.

Refer to http://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_n_bsb_using_bsb.htm for XPS Base System Builder usage.

The reference design with the MicroBlaze processor runs the standalone board support package software with a simple C application that prompts you to enter values for each input variable and outputs the result.

AutoESL PCORE Functionality

The AutoESL PCORE functionality is an 8 bit adder. The focus of this document is the interface of the pcore to the MicroBlaze processor through the AXI-Lite interface, not the functionality of the pcore.

The AutoESL module has three variables, `A`, `B` and `C`. `A` and `B` are input variables while `C` is an output variable. These three variables will be mapped to three registers in the generated PCORE.

Any AutoESL module has at least three control signals, `AP_START`, `AP_IDLE`, and `AP_START`. These signals are mapped to register in the generated PCORE.

The `AP_START` register is used to control the start of the PCORE and `AP_IDLE` indicates when the module operation is done.

The timing diagram of these three signals is shown in Figure 1-19.

Additional registers are present in the PCORE to support interrupts.

Block Diagram

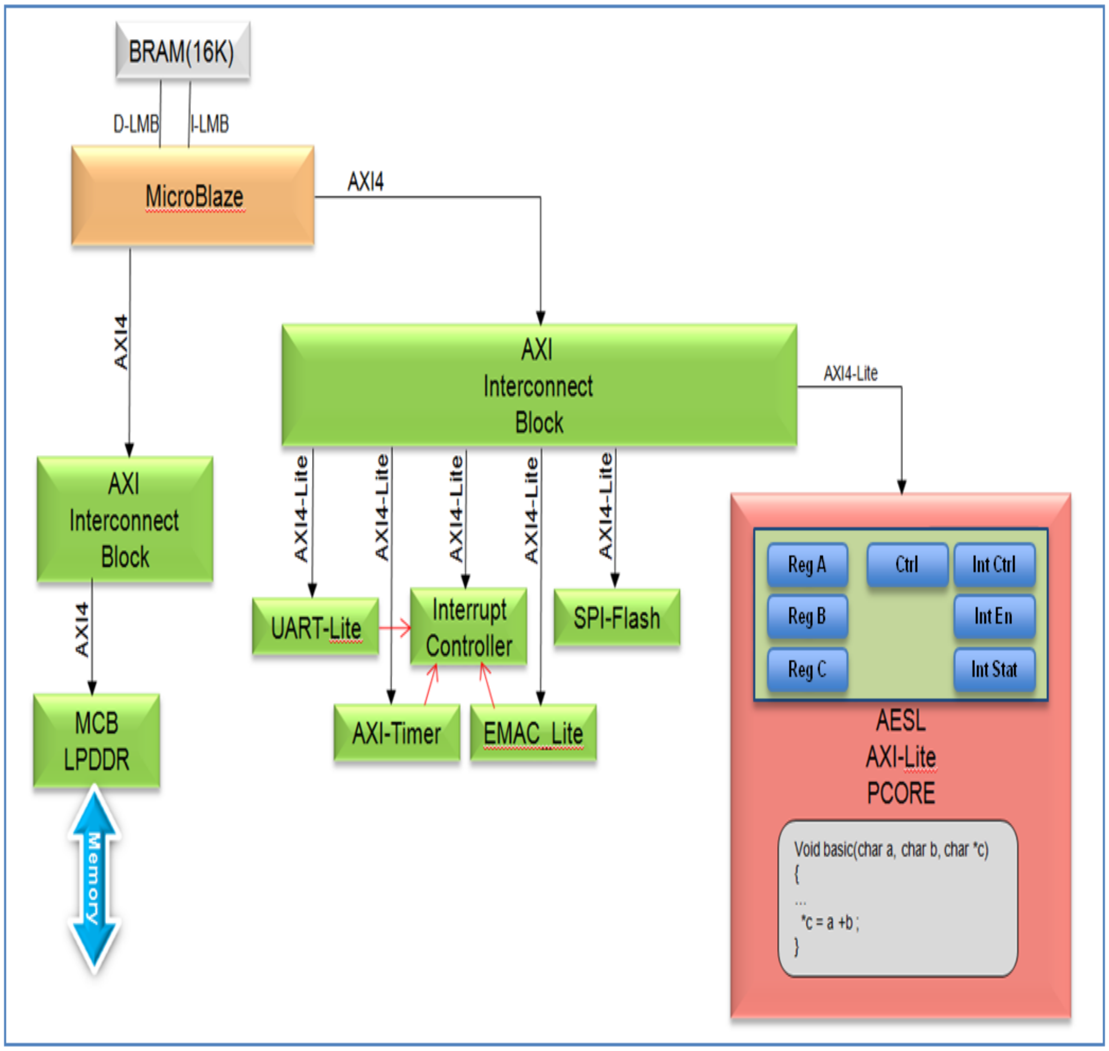


Figure 1-2: **Block Diagram**

Figure 1-2 shows a block diagram of the system. The complete architecture consists of:

- MicroBlaze processor
- 616K of BRAM to run code for the MicroBlaze processor
- Custom PCORE created using the AutoESL tool
- UART used for communication with the MicroBlaze processor
- Interrupt controller (not used in the demo part of design)

- LPDDR (not used in the demo but part of the design)
- AXI-Timer (not used in the demo but part of the design)
- SPI-Flash (not used in the demo but part of the design)
- Ethernet-Lite (not used in the demo but part of the design)
- Two AXI-Interconnects

Creating EDK PCORE with AXI-LITE

The steps to create the AXI-Lite interface PCORE with the functionality described in the following section:

1. Open the AutoESL Project basic.prj

   Start AutoESL, select Open Project then navigate to basic.prj and select it.

   Refer to the AutoESL Tutorial Introduction for details on how to create an AutoESL project. The AutoESL_Tutorial_Introduction.pdf is located under the doc directory where the AutoESL tool is installed. For example:
   C:/Xilinx/2012.1/AutoESL/doc/AutoESL_Tutorial_Introduction.pdf.

2. Figure 4 shows the same code with explanation.

   To generate PCOREs using AutoESL, the header file ap_interfaces.h must be included. The ap_interfaces.h header file is simply a convenient way to define macros which apply standard AutoESL directives as pragmas.

   In the example, we will make use of the AP_INTERFACE_REG_AXI4_LITE and the AP_CONTROL_BUS_AXI macros.

   The AP_INTERFACE_REG_AXI4_LITE macro is used to define that the three function arguments (a, b and c) be implemented as registers that are accessed through an AXI4-Lite interface.

   - Each port is specified as being in group BUS_A. This means they will all be grouped into the same AXI4 Lite interface called BUS_A.
   - The RTL interface is set to type ap_none. This means the RTL implementation will have only data ports: there will be no associated acknowledge or valid signals with each data port and hence no associated register in the interface.

   The AP_CONTROL_BUS_AXI macro is used add the block level IO protocol signals to an AXI4-Lite interface.

- The control signals AP_START, AP_DONE and AP_IDLE are created by default (the default function interface is ap_ctrl_hs) when AutoESL synthesizes the top-level function.
- Specifying then name BUS_A ensures these signals are grouped into the same AXI4 Lite interface as the other ports.

Table 2 describes all the registers created by AutoESL for the generated PCORE.

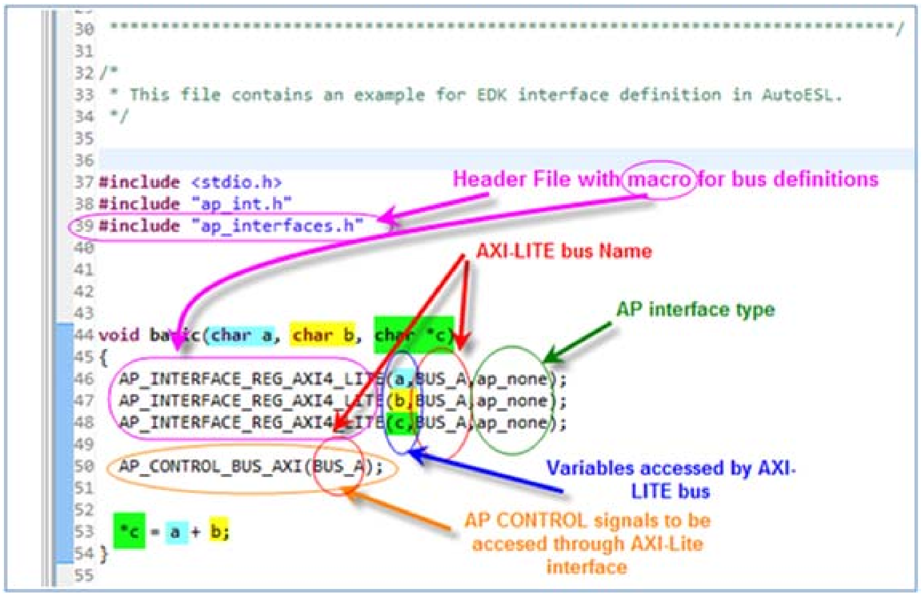
Creating EDK PCORE

The steps to create the EDK PCORE (functionality described in the AutoESL PCORE Functionality section) with the AXI-Lite interface are:

1. Open the AutoESL project basic.prj. Details on using AutoESL can be found in the AutoESL tutorial. Refer to *AutoESL Tutorial: Introduction (UG871)*.

The `AutoESL_Tutorial_Introduction.pdf` file is located under the doc directory where the AutoESL tool is installed. For example,
`C:/xilinx/2011.4.2/2011.4.2/AutoESL/doc/AutoESL_Tutorial_Introduction.pdf`

2. Open the AutoESL project basic.prj. Details on using AutoESL can be found in the AutoESL tutorial shown in Figure 1-3.


```
***************************************************************************/

/*
 * This file contains an example for EDK interface definition in AutoESL.
 */


#include <stdio.h>
#include "ap_int.h"
#include "ap_interfaces.h"




void basic(char a, char b, char *c)
{
  AP_INTERFACE_REG_AXI4_LITE(a,BUS_A,ap_none);
  AP_INTERFACE_REG_AXI4_LITE(b,BUS_A,ap_none);
  AP_INTERFACE_REG_AXI4_LITE(c,BUS_A,ap_none);

  AP_CONTROL_BUS_AXI(BUS_A);


  *c = a + b;
}
```
Header File with macro for bus definitions
AXI-LITE bus Name
AP interface type
Variables accessed by AXI-LITE bus
AP CONTROL signals to be accesed through AXI-Lite interface

Figure 1-3: **C Code**

3. Click the **Synthesis** button as shown in Figure 1-4.

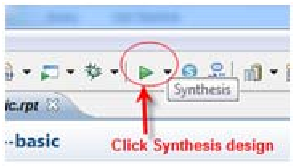


Figure 1-4: **Synthesis**

4. Click the **RTL Implementation** button as shown in Figure 1-5.

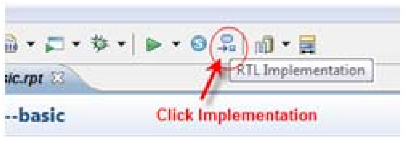
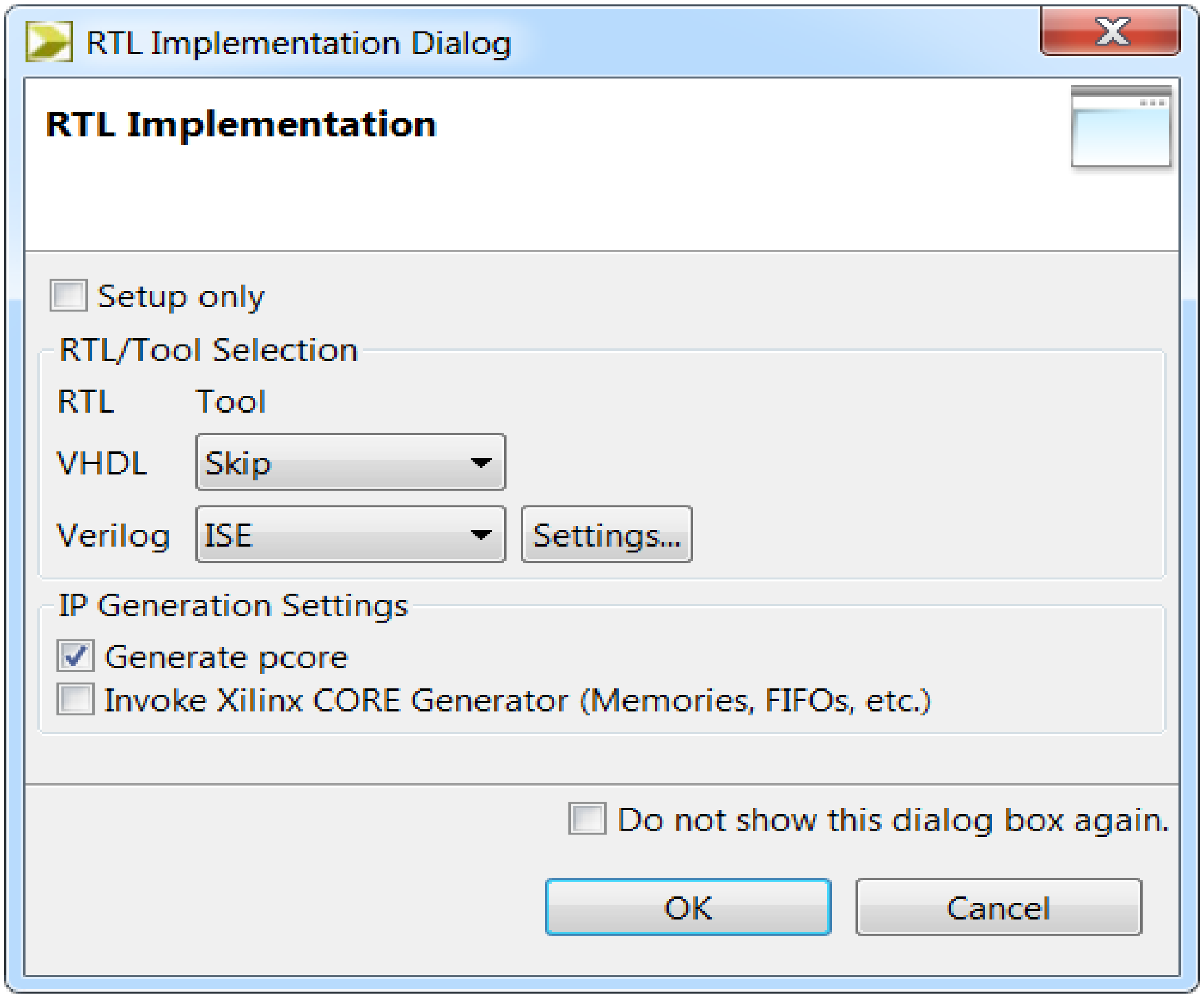


Figure 1-5: **RTL Implementation**

5. Select **Generate pcore** from the RTL Implementation Dialog window as shown in Figure 1-6.

   In addition to creating the PCORE, these setting execute ISE for RTL synthesis. The ISE executable must be in the windows search path for ISE to launch: refer to the AutoESL Installation Guide.


RTL Implementation Dialog

RTL Implementation

☐ Setup only

RTL/Tool Selection

RTL	Tool
VHDL	Skip
Verilog	ISE

IP Generation Settings

☑ Generate pcore
☐ Invoke Xilinx CORE Generator (Memories, FIFOs, etc.)

☐ Do not show this dialog box again.

Figure 1-6: **RTL Implementation Dialog**

The generated PCORE will be under the `impl` directory of the selected solution, `solution1`, as shown in Figure 1-7.

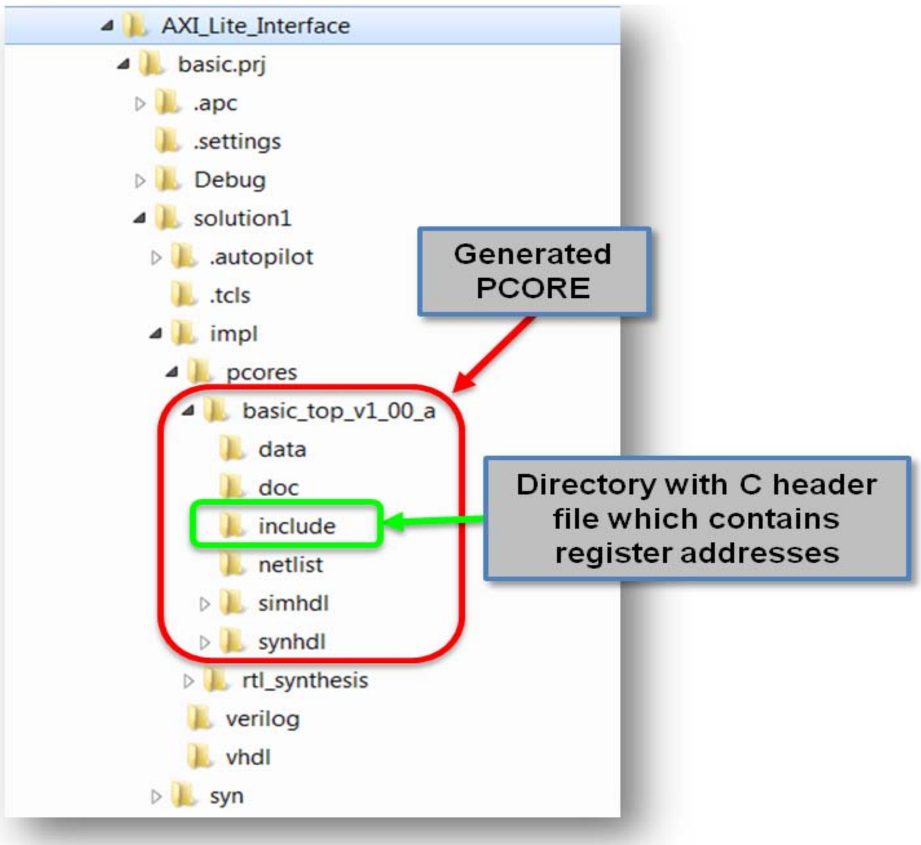


Figure 1-7: **Generated PCORE Location**

PCORE Register List

As stated in the AutoESL PCORE Functionality section, the PCORE has seven registers. Three registers represent the passed in arguments (`A`, `B`, and `C`) of the C code. The other fourregisters represent the control register for `AP` control signals `AP_START`, `AP_DONE`, and `AP_IDLE` and three interrupt control registers.

Table 1-1: **PCORE Registers**

Register Name	Width	R/W	Default Value	Address offset	Description
Control	3	R/W	0	0x00	Bit 0 - ap_start (Read/Write/SC) Bit 1 - ap_done (Read/COR) Bit 2 - ap_idle (Read) SC = Self Clear, COR = Clear on Read
Global Interrupt Control	1	R/W	0	0x04	Bit 0 - Enable all interrupts.
Interrupt enable Register	1	R/W	0	0x08	Bit 0 - ap_done signal.
Interrupt Status Register	1	R/W	0	0x0c	Bit 0 - ap_done signal (Read/TOW) TOW = Toggle on Write
`A`	8	R/W	0	0x14	Variable `A`.
`B`	8	R/W	0	0x1c	Variable `B`.
`C`	8	R/W	0	0x24	Variable `C`.

Integrating Generated PCORE

The steps to integrate generated PCORE with the MicroBlaze Processor using the XPS Tool Suite are:

1. Copy generated PCORE from AutoESL directory structure to XPS directory structure, as shown in Figure 1-8.

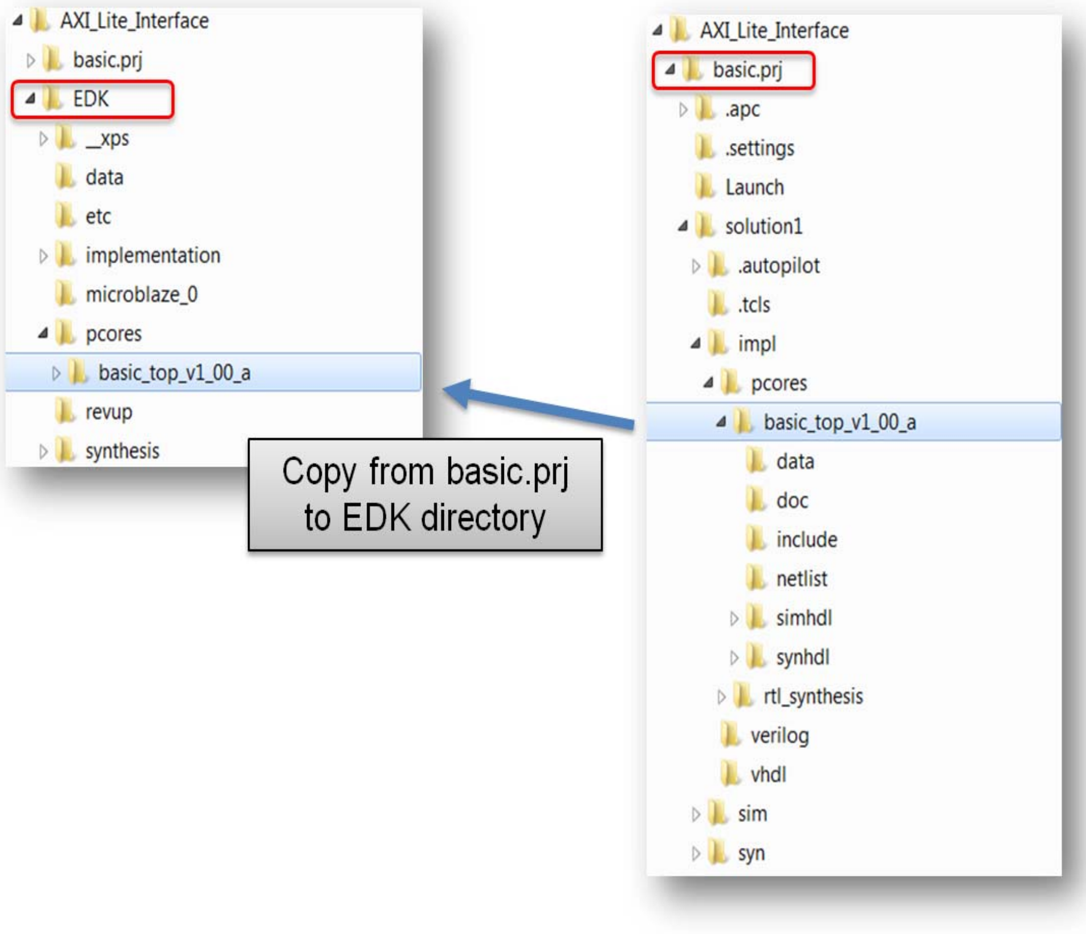


Figure 1-8: **XPS and AutoESL Directory Structures**

2. In XPS, add the generated PCORE from the IP catalog by selecting and clicking with left mouse on **PCORE**. The new PCORE will be under the `Project Local PCores` directory, as shown in Figure 1-9.

   When the connection pop-up window appears, accept connecting to instance microblaze_0.

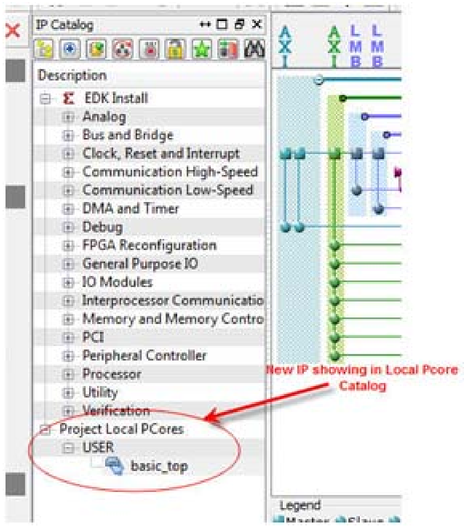


Figure 1-9: **Add Generated PCORE**

3. If PCORE is not listed under the `Project Local PCores` directory then you will need to direct XPS to rescan user repository.

   Select **Project > Rescan User Repository**, as shown in Figure 1-10.

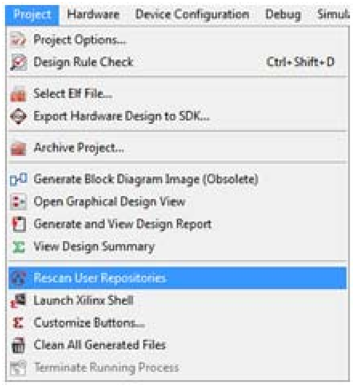


Figure 1-10: **Rescan User Repositories**

4. Change the default Reset Polarity of the generated pcore from Active Low.

   a. Double click on **PCORE** to customize it, as shown in Figure 1-11.

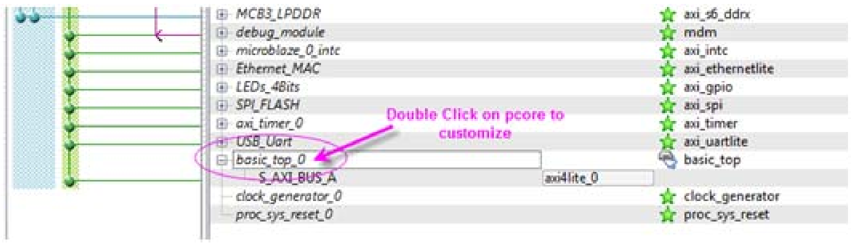
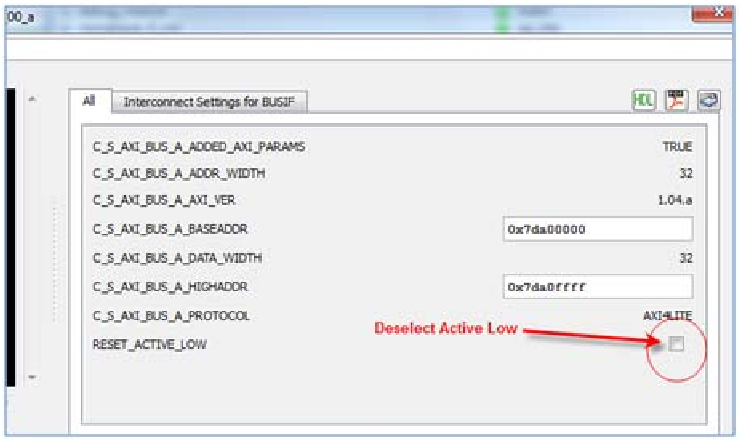


Figure 1-11: **Customize PCORE**

b. Deselect RESET_ACTIVE_LOW signal, as shown in Figure 1-12.

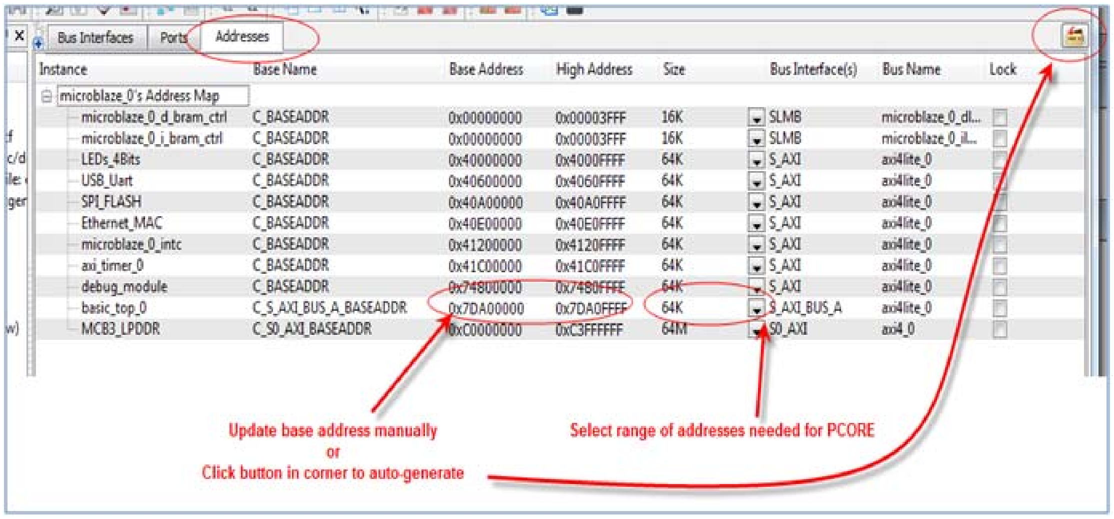


C_S_AXI_BUS_A_ADDED_AXI_PARAMS	TRUE
C_S_AXI_BUS_A_ADDR_WIDTH	32
C_S_AXI_BUS_A_AXI_VER	1.04.a
C_S_AXI_BUS_A_BASEADDR	0x7da00000
C_S_AXI_BUS_A_DATA_WIDTH	32
C_S_AXI_BUS_A_HIGHADDR	0x7da0ffff
C_S_AXI_BUS_A_PROTOCOL	AXI4LITE
RESET_ACTIVE_LOW	☐

Figure 1-12: **Deselect Active Low**

5. Set the PCORE base address.

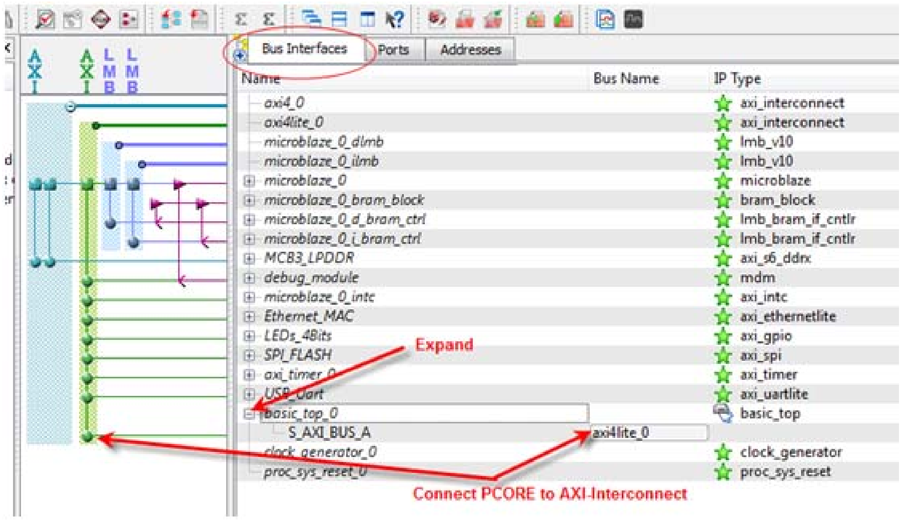
   The base address can be set from different locations. One location is through the PCORE customization window (see Figure 1-12). The other location is the Addresses tab in the Assembly window (see Figure 1-13). The Assembly window is the preferred place because the full memory map for the MicroBlaze processor can be seen which prevents memory overlap errors. The ability to auto-generate addresses is also available from the Assembly window.


Bus Interfaces
Ports
Addresses
Update base address manually
or
Click button in corner to auto-generate
Select range of addresses needed for PCORE

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	microblaze_0_dl...	
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	microblaze_0_il...	
LEDs_4Bits	C_BASEADDR	0x40000000	0x4000FFFF	64K	S_AXI	axi4lite_0	
USB_Uart	C_BASEADDR	0x40600000	0x4060FFFF	64K	S_AXI	axi4lite_0	
SPI_FLASH	C_BASEADDR	0x40A00000	0x40A0FFFF	64K	S_AXI	axi4lite_0	
Ethernet_MAC	C_BASEADDR	0x40E00000	0x40E0FFFF	64K	S_AXI	axi4lite_0	
microblaze_0_intc	C_BASEADDR	0x41200000	0x4120FFFF	64K	S_AXI	axi4lite_0	
axi_timer_0	C_BASEADDR	0x41C00000	0x41C0FFFF	64K	S_AXI	axi4lite_0	
debug_module	C_BASEADDR	0x74800000	0x7480FFFF	64K	S_AXI	axi4lite_0	
basic_top_0	C_S_AXI_BUS_A_BASEADDR	0x7DA00000	0x7DA0FFFF	64K	S_AXI_BUS_A	axi4lite_0	
MCB3_LPDDR	C_S0_AXI_BASEADDR	0xC0000000	0xC3FFFFFF	64M	S0_AXI	axi4_0	

Figure 1-13: **Set PCORE Base Address**

6. Connect PCORE to AXI-Interconnect from the Bus Interfaces tab, as shown in Figure 1-14.



Name	Bus Name	IP Type
axi4_0		axi_interconnect
axi4lite_0		axi_interconnect
microblaze_0_dlmb		lmb_v10
microblaze_0_ilmb		lmb_v10
microblaze_0		microblaze
microblaze_0_bram_block		bram_block
microblaze_0_d_bram_ctrl		lmb_bram_if_cntlr
microblaze_0_i_bram_ctrl		lmb_bram_if_cntlr
MCB3_LPDDR		axi_s6_ddrx
debug_module		mdm
microblaze_0_intc		axi_intc
Ethernet_MAC		axi_ethernetlite
LEDs_4Bits		axi_gpio
SPI_FLASH		axi_spi
axi_timer_0		axi_timer
USB_Uart		axi_uartlite
basic_top_0		basic_top
S_AXI_BUS_A	axi4lite_0	
clock_generator_0		clock_generator
proc_sys_reset_0		proc_sys_reset

Figure 1-14: **Connect PCORE to AXI-Interconnect**

7. Connect clocks and rest signals.

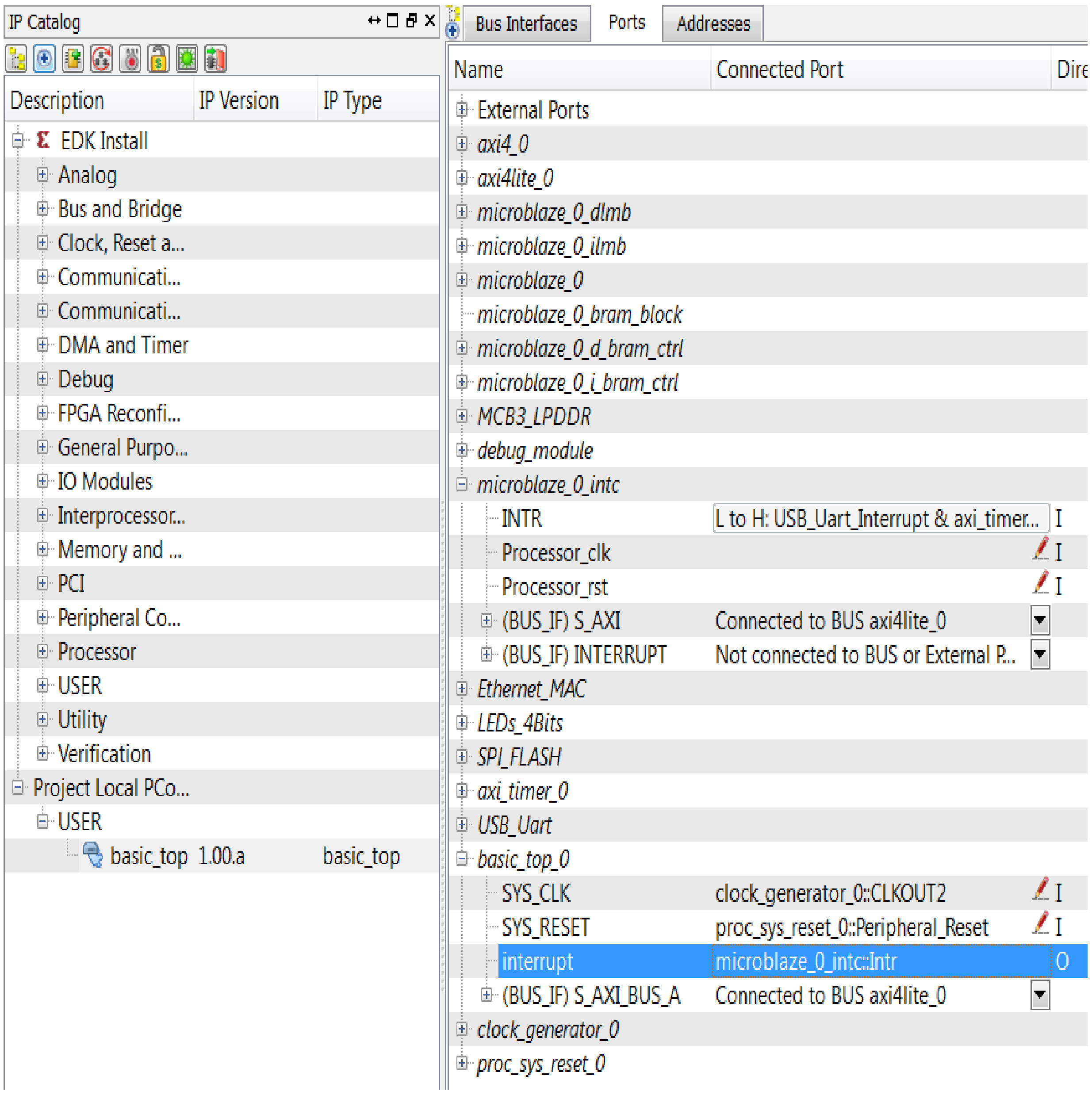
   You need to go to the Ports tab to see the other ports that needs connections (see Figure 1-15).

   As shown in Figure 1-15, on instance, basic_top_0:

   ◦ Connect port SYS_CLK to clock_generator_0: CLKOUT2

   ◦ Connect port SYS_RST to proc_sys_reset_0: Peripheral_Reset

- Add port interupt basic_top_0 to the list of connected inerupts
- Confirm port (BUS_IF) is connect to clock_generator_0: CLKOUT2



IP Catalog

Description	IP Version	IP Type
EDK Install		
Analog		
Bus and Bridge		
Clock, Reset a...		
Communicati...		
Communicati...		
DMA and Timer		
Debug		
FPGA Reconfi...		
General Purpo...		
IO Modules		
Interprocessor...		
Memory and ...		
PCI		
Peripheral Co...		
Processor		
USER		
Utility		
Verification		
Project Local PCo...		
USER		
basic_top	1.00.a	basic_top

Bus Interfaces | Ports | Addresses

Name	Connected Port	Dire
External Ports		
axi4_0		
axi4lite_0		
microblaze_0_dlmb		
microblaze_0_ilmb		
microblaze_0		
microblaze_0_bram_block		
microblaze_0_d_bram_ctrl		
microblaze_0_i_bram_ctrl		
MCB3_LPDDR		
debug_module		
microblaze_0_intc		
INTR	L to H: USB_Uart_Interrupt & axi_timer...	I
Processor_clk		I
Processor_rst		I
(BUS_IF) S_AXI	Connected to BUS axi4lite_0	
(BUS_IF) INTERRUPT	Not connected to BUS or External P...	
Ethernet_MAC		
LEDs_4Bits		
SPI_FLASH		
axi_timer_0		
USB_Uart		
basic_top_0		
SYS_CLK	clock_generator_0::CLKOUT2	I
SYS_RESET	proc_sys_reset_0::Peripheral_Reset	I
interrupt	microblaze_0_intc::Intr	O
(BUS_IF) S_AXI_BUS_A	Connected to BUS axi4lite_0	
clock_generator_0		
proc_sys_reset_0		

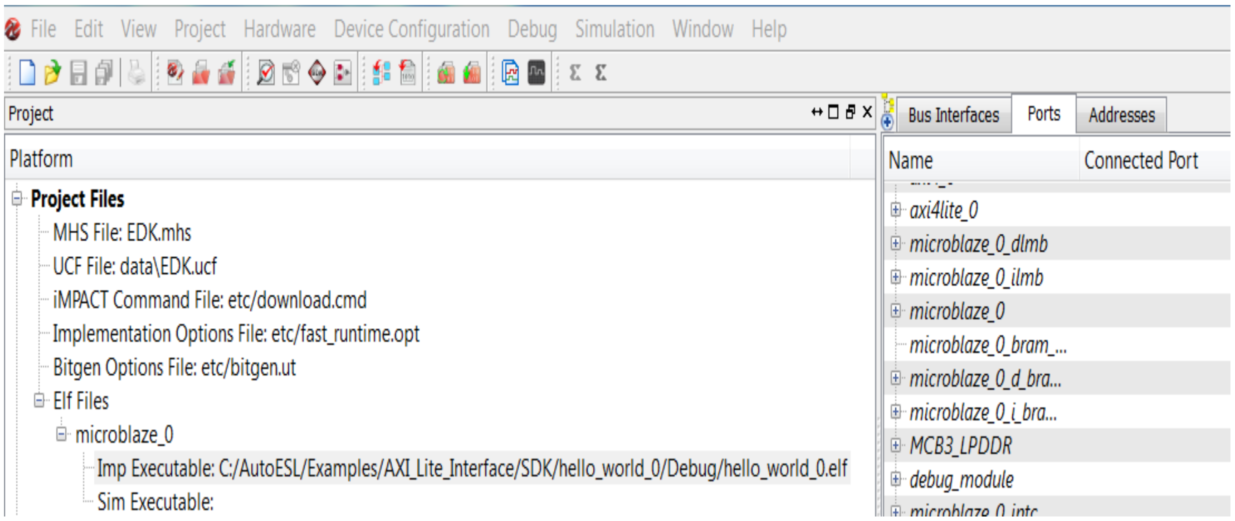
Figure 1-15: **Connect Clocks and Rest Signals**

Generating the FPGA Bitstream

A software application image is needed to initialize the BRAM. The MicroBlaze processor will run the software application after reset.

1. Refer to the Creating Application Software section for steps on creating an ELF file. In this example, the application software has already been compiled into file hello_world_0.elf.

   Figure 1-16 shows how to select the ELF file to initialize the BRAM.


File Edit View Project Hardware Device Configuration Debug Simulation Window Help
Project
Bus Interfaces Ports Addresses
Name Connected Port

Platform
- Project Files
  - MHS File: EDK.mhs
  - UCF File: data\EDK.ucf
  - iMPACT Command File: etc/download.cmd
  - Implementation Options File: etc/fast_runtime.opt
  - Bitgen Options File: etc/bitgen.ut
  - Elf Files
    - microblaze_0
      - Imp Executable: C:/AutoESL/Examples/AXI_Lite_Interface/SDK/hello_world_0/Debug/hello_world_0.elf
      - Sim Executable:

axi4lite_0
microblaze_0_dlmb
microblaze_0_ilmb
microblaze_0
microblaze_0_bram_...
microblaze_0_d_bra...
microblaze_0_i_bra...
MCB3_LPDDR
debug_module
microblaze_0_intc

Figure 1-16: **Select ELF File**

2. Generate the bitstream by selecting **Update Bitstream** from the Device Configuration menu, as shown in Figure 1-17.

   Note: This will perform two serial steps:

   a. Generate the FPGA bitstream `<project_name>.bit` in the implementation directory.

   b. Initialize the BRAM with the ELF file selected and generate a `download.bit` file. The `download.bit` file will be under the implementation directory.

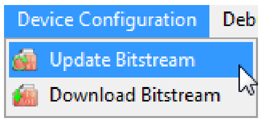
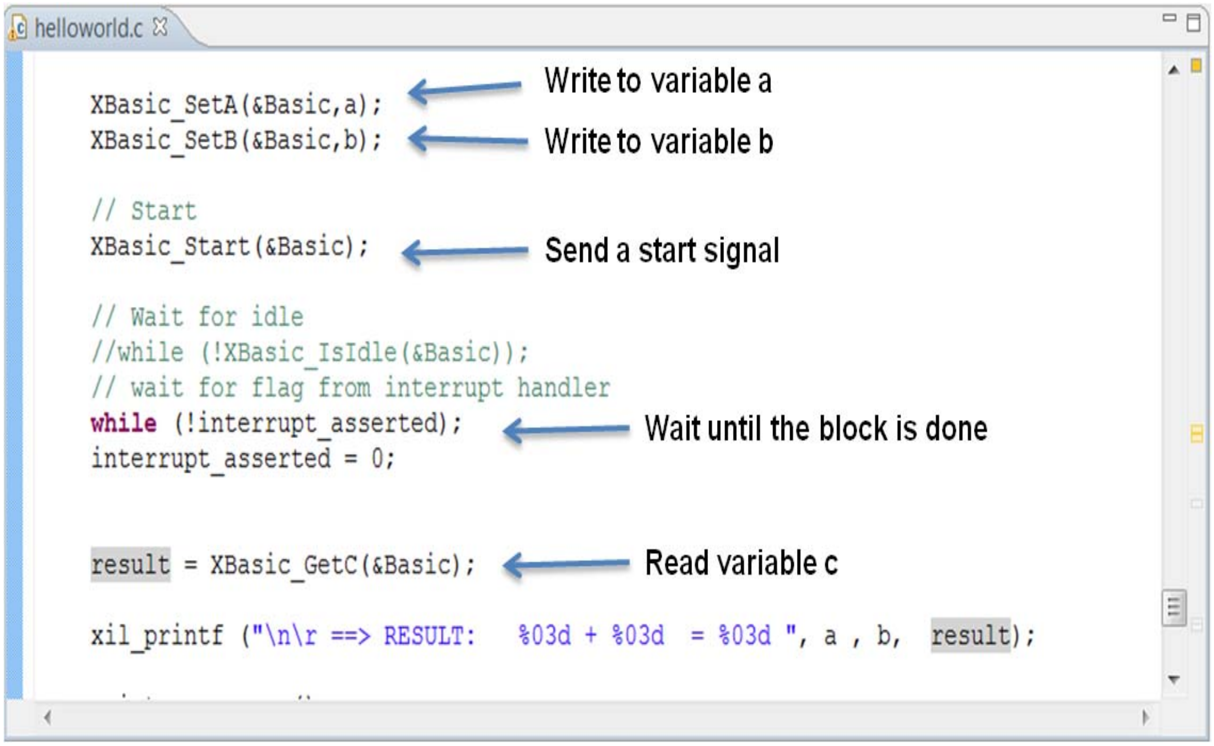


Figure 1-17: **Generate Bitstream**

Controlling the Generated PCORE

The generated pcore has six registers accessible by the MicroBlaze processor through the AXI-Lite interface. C code is needed to read and write with these registers, as shown in Figure 1-18.



helloworld.c

```
XBasic_SetA(&Basic,a);
XBasic_SetB(&Basic,b);

// Start
XBasic_Start(&Basic);

// Wait for idle
//while (!XBasic_IsIdle(&Basic));
// wait for flag from interrupt handler
while (!interrupt_asserted);
interrupt_asserted = 0;


result = XBasic_GetC(&Basic);

xil_printf ("\n\r ==> RESULT:   %03d + %03d  = %03d ", a , b,  result);
```

Figure 1-18: **C Code to Read and Write with Registers**

The AutoESL pcore provides C functions which allows the ports to be accessed. In this example, these functions are in xbasic.c and header file xbasic.h. Header file xbasic_BUS_A.h creates some useful macros.

Creating Application Software

Xilinx's Software Development Kit (SDK) is a software development environment to create and debug software applications. Features include project management, multiple build configurations, a feature rich C/C++ code editor, error navigation, a debugging and profiling environment, and source code version control. For more SDK information refer to http://www.xilinx.com/tools/sdk.htm for detail information on SDK.

The steps for creating application software using SDK are:

This project is already provided in the tutorial. The steps here show how to re-create this.

1. The hardware description of the system needs to be exported from XPS to SDK, to be able to create software application images for that system.

   Select **Project > Export Hardware Design to SDK....**, as shown in Figure 1-19**.**

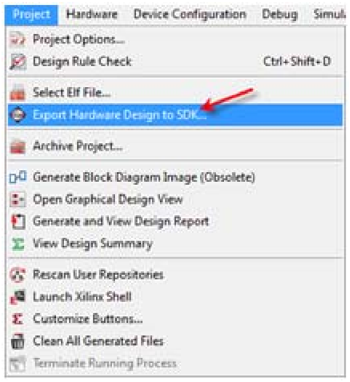
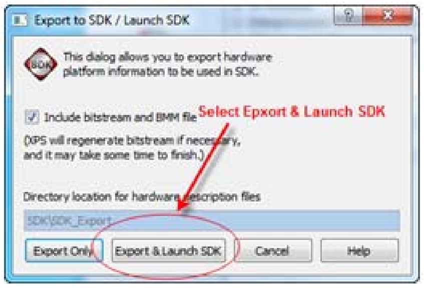


Figure 1-19: **Export Hardware Design**

2. From the Export to SDK window, click **Export & Launch SDK,** as shown in **Figure 1-20.**

   Note: This may take several minutes to complete.


Export to SDK / Launch SDK

This dialog allows you to export hardware platform information to be used in SDK.

☑ Include bitstream and BMM file

(XPS will regenerate bitstream if necessary, and it may take some time to finish.)

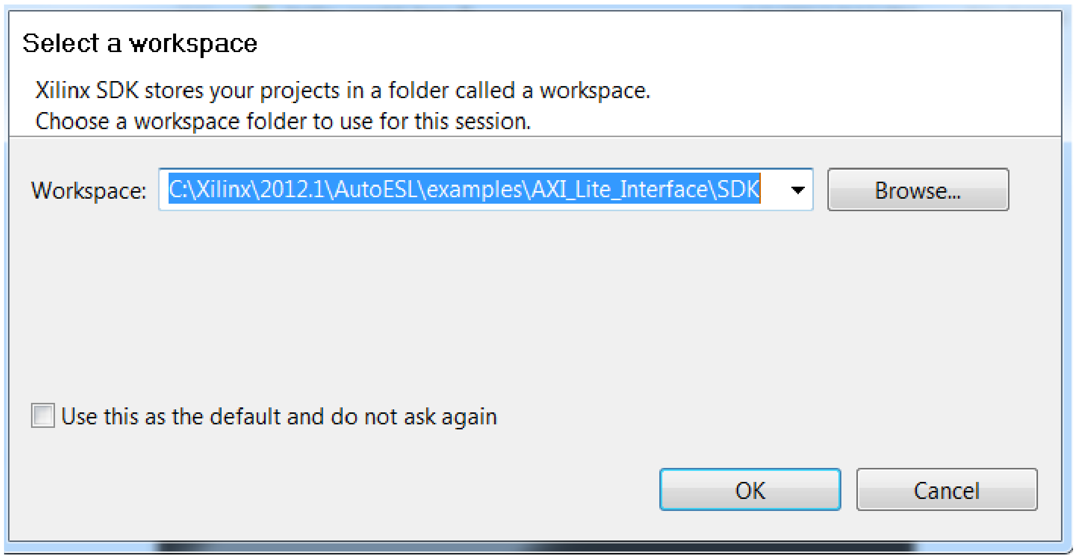
Directory location for hardware description files

SDK\SDK_Export

Figure 1-20: **Export & Launch SDK**

3. From the Workspace Launcher window, use the **Browse...** button to select a directory location for your workspace and click **OK**, as shown in Figure 1-21.

   Note: Make sure no spaces are in the path names.



Select a workspace

Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Xilinx\2012.1\AutoESL\examples\AXI_Lite_Interface\SDK

☐ Use this as the default and do not ask again

Figure 1-21: **Select a Workspace**

4. To create a new C project, select **File > New > Xilinx C Project**, as shown in Figure 1-22.

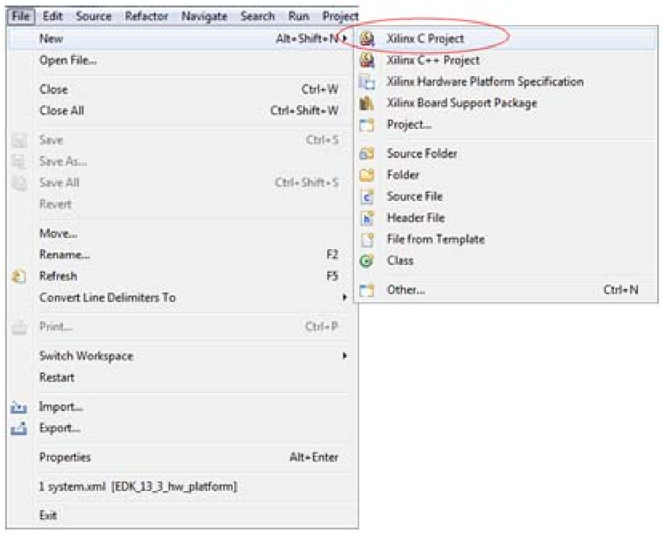
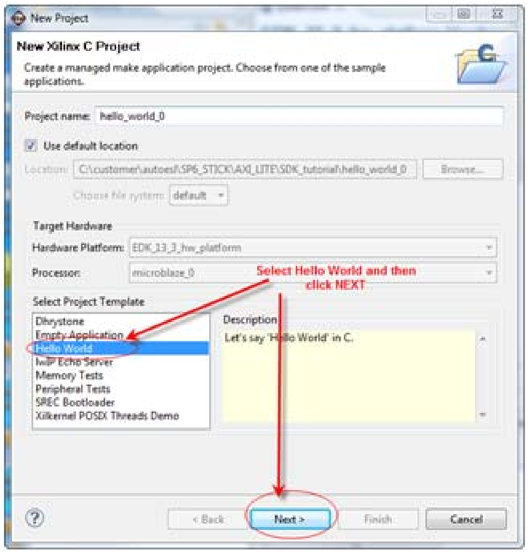


Figure 1-22: **Create New C Project**

5. Select the **Hello World** application as a starting point from the project templates and click **Next**, as shown in Figure 1-23.


New Project

New Xilinx C Project

Create a managed make application project. Choose from one of the sample applications.

Project name: hello_world_0

[✓] Use default location

Location: C:\customer\autoesl\SP6_STICK\AXI_LITE\SDK_tutorial\hello_world_0

Choose file system: default

Target Hardware

Hardware Platform: EDK_13_3_hw_platform

Processor: microblaze_0

Select Project Template

Dhrystone
Empty Application
Hello World
lwIP Echo Server
Memory Tests
Peripheral Tests
SREC Bootloader
Xilkernel POSIX Threads Demo

Description

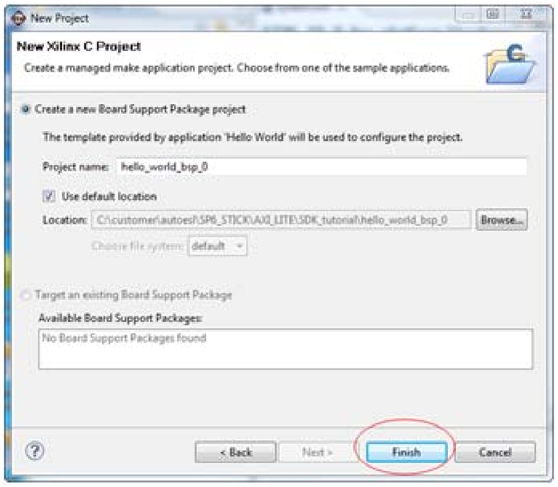
Let's say 'Hello World' in C.

Select Hello World and then click NEXT

Figure 1-23: **Select Hello World**

6. Click **Finish**, as shown in Figure 1-24.

   The application will automatically start building and create an ELF file.


New Project

New Xilinx C Project

Create a managed make application project. Choose from one of the sample applications.

● Create a new Board Support Package project

The template provided by application 'Hello World' will be used to configure the project.

Project name: hello_world_bsp_0

[✓] Use default location

Location: C:\customer\autoesl\SP6_STICK\AXI_LITE\SDK_tutorial\hello_world_bsp_0

Choose file system: default

○ Target an existing Board Support Package

Available Board Support Packages:

No Board Support Packages found

Figure 1-24: **Finish New C Project**

The ELF file is the compiled application and will be located under the debug directory (see Figure 1-25), with the application name and the elf extension. In this example `hello_world_0.elf`.

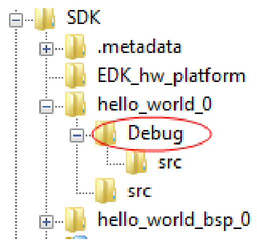


Figure 1-25: **ELF File Location**

7. Edit the `Helloworld.c` file and add code to test the generated PCORE.

   Include the C files from the AutoESL pcore sub-directory include (basic_top_v1_00_a\include) as shown in figure Figure 1-26.

   Open the helloworld.c for editing by double clicking on helloworld.c after expanding the hello_world_0 application and the src directory as shown below.

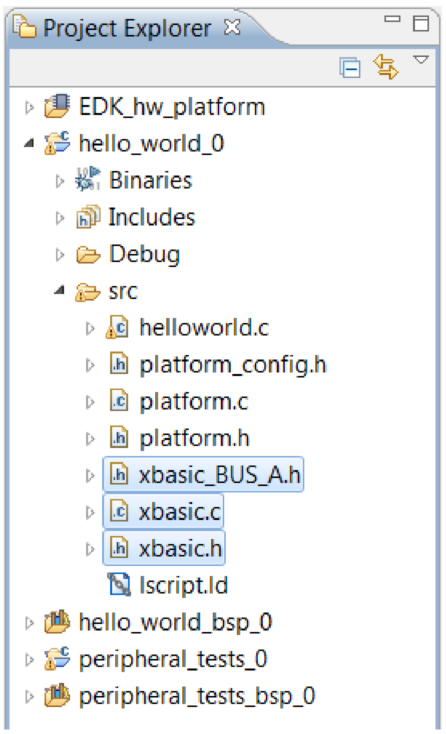


Figure 1-26: **Project Explorer**

In the helloworld.c editor change the code to match the code below:

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xil_io.h"
#include "xstatus.h"
#include "xbasic.h"  // DM added
```

```
#include "xintc.h"
#include "xil_exception.h"
#include "xuartlite_l.h"

#define pritnf xil_printf
void print(char *str);



// BASIC PCORE SETUP
XBasic Basic;
XBasic_Config Basic_Config =
{
    0,
    XPAR_BASIC_TOP_0_S_AXI_BUS_A_BASEADDR
};

int SetupBasic(void)
{
    return XBasic_Initialize(&Basic, &Basic_Config);
}

//-------------- Setup Interrupt control -----------------------------
//
#define INTC_DEVICE_ID          XPAR_INTC_0_DEVICE_ID
#define XBASIC_INTERRUPT_ID   XPAR_MICROBLAZE_0_INTC_BASIC_TOP_0_INTERRUPT_INTR
XIntc InterruptController;  /* The instance of the Interrupt Controller */

int interrupt_count = 0;   // just for statiscs
int interrupt_asserted = 0;

void XBasic_InterruptHandler(void *InstancePtr)
{

      interrupt_count++;
      // clear the interrupt
      XBasic_InterruptClear(&Basic, 1);
      // poor man semaphore
      interrupt_asserted = 1;

}


//--------------------------------------------------
int SetupInterrupt(void)
{
      int Status;

    // Initialize the interrupt controller driver so that it is ready to use.
      Status = XIntc_Initialize(&InterruptController, INTC_DEVICE_ID);
      if (Status != XST_SUCCESS)
      {
            return XST_FAILURE;
      }

      // Connect a device driver handler that will be called when an interrupt
      // for the device occurs, the device driver handler performs the specific
      // interrupt processing for the device
      Status = XIntc_Connect
```

```
                    ( &InterruptController, XBASIC_INTERRUPT_ID,
                       (XInterruptHandler)XBasic_InterruptHandler,
                       NULL
                    );
      if (Status != XST_SUCCESS)
      {
            return XST_FAILURE;
      }

      // Start the interrupt controller such that interrupts are enabled for
      // all devices that cause interrupts, specific real mode so that
      // the timer counter can cause interrupts thru the interrupt controller.
      //
      Status = XIntc_Start(&InterruptController, XIN_REAL_MODE);
      if (Status != XST_SUCCESS)  { return XST_FAILURE; }

      // Enable the interrupt for the AESL BASIC CORE
      XIntc_Enable(&InterruptController, XBASIC_INTERRUPT_ID);



    //  Initialize the exception table.
      Xil_ExceptionInit();

      // Register the interrupt controller handler with the exception table.
      Xil_ExceptionRegisterHandler(
                  XIL_EXCEPTION_ID_INT,
                  (Xil_ExceptionHandler) XIntc_InterruptHandler,
                  &InterruptController
                  );

      // Enable non-critical exceptions.
      Xil_ExceptionEnable();

    XBasic_InterruptEnable(&Basic, 1);
    XBasic_InterruptGlobalEnable(&Basic);

      return XST_SUCCESS;
}



void print_core_regs(void )
{

      xil_printf ("\n\r   A       reg [0x%08x] ", XBasic_GetA(&Basic ) );
      xil_printf ("\n\r   B       reg [0x%08x] ", XBasic_GetB(&Basic ));
      xil_printf ("\n\r   C       reg [0x%08x] ", XBasic_GetC(&Basic ) );
      xil_printf ("\n\r   DONE    reg [0x%08x] ", XBasic_IsDone(&Basic) );
      xil_printf ("\n\r   IDLE    reg [0x%08x] ", XBasic_IsIdle(&Basic) );

      xil_printf ("\n\r   INT STATS   [%d]      ", interrupt_count );

}

int ReadInt(int size)
{
  int value=0;
  char c ='0';
```

```
    int i;
    for (i=0; i <size; i++)
    {
        c=inbyte();
      if (c==' ' )
      {
         c='0';
         outbyte(c);
      }
      else if (c=='\n')
      {
        break;
        return value;
      }
      else if (c=='\r')
      {
        break;
        return value;
      }
      else
      {
        outbyte(c);
        value=value*10+c-'0';
      }
    }

    return value;
  }


  int main()
  {
        //init_platform();

      int a = 1000;
      int b = 1000;
      u32 result;

      // initialize AESL PCORE

      int status;
      status = XBasic_Initialize(&Basic, &Basic_Config);
      if (status != XST_SUCCESS) {
          xil_printf("\n\r ==> Basic failed.\n\r");
      } else {
          xil_printf("\n\r ==> Basic succeeded.\n\r");
      }

      // Initialize the interrupts (local then global)
      status = SetupInterrupt();
      if (status != XST_SUCCESS) {
          xil_printf("\n\r ==> SetupInterrupt failed.\n\r\n\r");
      } else {
          xil_printf("\n\r ==> SetupInterrupt succeeded.\n\r\n\r");
     }


      // Get and setup the data
      while (1)
```

```
        {
        print("\n\r ==========================================================");
        print("\n\r =========== START OF AESL BASIC CORE TEST =================");
        print("\n\r ================      RESULT = A + B     =================\n\r");

        while (a > 255)
        {
          print("\n\r --> Please enter number between (0-255) for variable A : ");
          a = ReadInt(3);
        }

        while (b > 255)
        {
          print("\n\r --> Please enter number between (0-255) for variable B : ");
           b = ReadInt(3);
         }

        XBasic_SetA(&Basic,a);
        XBasic_SetB(&Basic,b);

        // Start
        XBasic_Start(&Basic);

        // Wait for idle
        //while (!XBasic_IsIdle(&Basic));
        // wait for flag from interrupt handler
        while (!interrupt_asserted);
        interrupt_asserted = 0;


        result = XBasic_GetC(&Basic);

        xil_printf ("\n\r ==> RESULT:   %03d + %03d  = %03d ", a , b,  result);

        print_core_regs();

        print("\n\r ==========================================================");
        print("\n\r ==========================================================\n\r");
        // reset variable to value bigger then 255 to prompt user for new input
        a =1000;
        b =1000;

        }

        //cleanup_platform();

        return 0;
    }
```

Running the Demo on the Avnet MicroBoard

Setup Requirements

- Board

- Two USB cables connected to UART and JTAG ports of the Avnet MicroBoard and to the PC, as shown in Figure 1-30

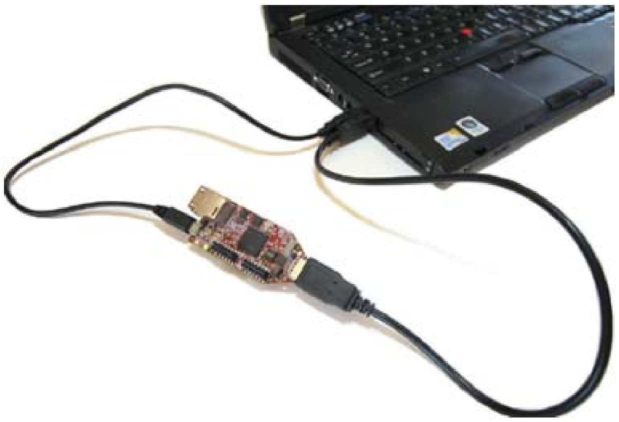
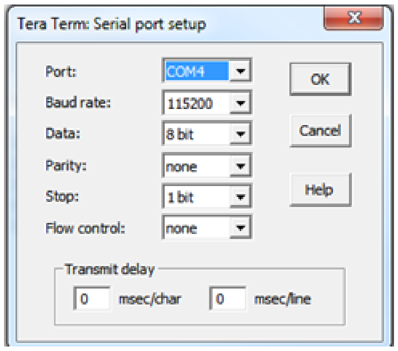


Figure 1-27: **Cable Connections**

- Hyperterminal (Tera Term) with the serial port setup shown in Figure 1-31



Port: COM4
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay
0 msec/char
0 msec/line

Figure 1-28: **Serial Port Setup**

- `Download.bit` file provided with the reference design

Finding Serial Port Number on Windows 7 PC

1. Click on the **Windows Start** button, as shown in Figure 1-32.

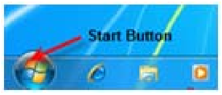


Figure 1-29: **Windows Start Button**

2. In the *Search programs and files* box, type **devmgmt.msc**.

   Windows will list devmgmt.msc in the search results window as shown in Figure 1-33.

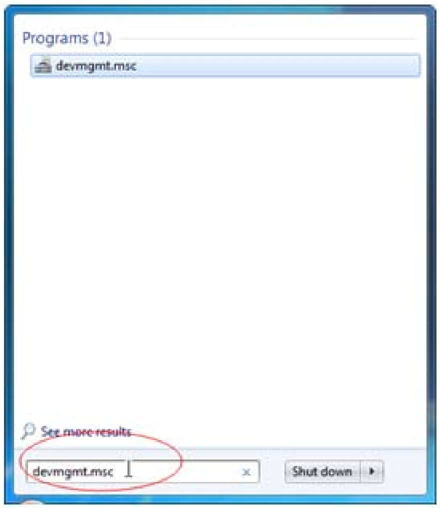


Figure 1-30: **Search for devmgmt.msc**

3. Select **devmgmt.msc** and when Windows asks for permission, click **Yes**.
4. When the Device Manager window appears, expand **Ports (COM & LPT)** to find the USB to UART COM port number, as shown in Figure 1-34.

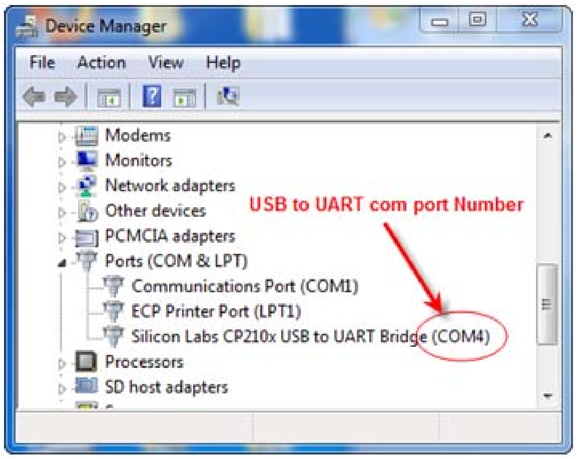


Figure 1-31: **USB to UART COM Port Number**

Running the Demo

1. Open hyperterminal (Tera Term) with the settings shown in Figure 1-31
2. Download the bit file from XPS by selecting **Device Configuration > Download Bitstream** from the Device Configuration menu as shown in Figure 1-35.

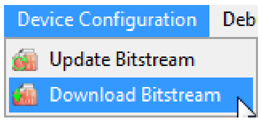
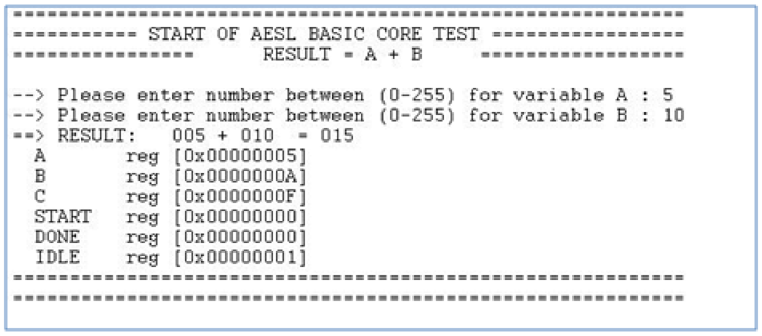


Figure 1-32: **Download Bitstream**

3. After the FPGA is configured, you will be prompted to enter values for `A` and `B`. Figure 1-36 shows an example output of the application.



```
===========================================================
========== START OF AESL BASIC CORE TEST ================
===============         RESULT = A + B      ================

--> Please enter number between (0-255) for variable A : 5
--> Please enter number between (0-255) for variable B : 10
==> RESULT:   005 + 010  = 015
  A       reg [0x00000005]
  B       reg [0x0000000A]
  C       reg [0x0000000F]
  START   reg [0x00000000]
  DONE    reg [0x00000000]
  IDLE    reg [0x00000001]
===========================================================
===========================================================
```

Figure 1-33: **Application Output**

Running Bus Functional Model Simulation

To run bus functional model simulation on generated PCORE.

1. Start XPS and select Create New Blank Project.

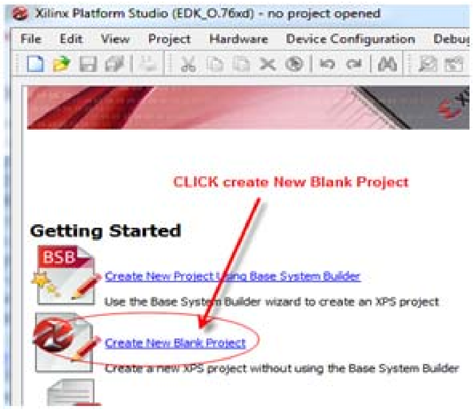
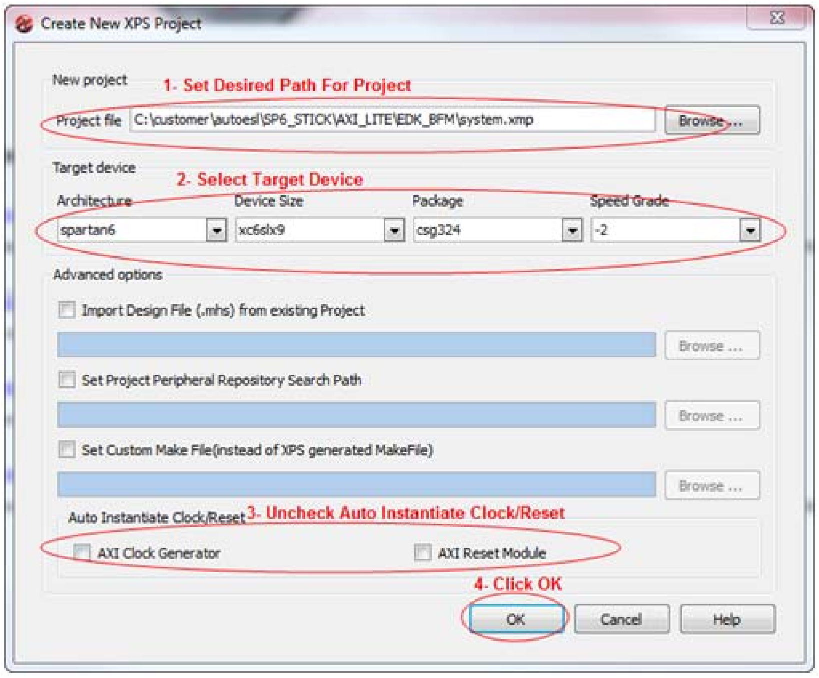

Xilinx Platform Studio (EDK_O.76xd) - no project opened
File
Edit
View
Project
Hardware
Device Configuration
Debug
CLICK create New Blank Project
Getting Started
BSB
Create New Project Using Base System Builder
Use the Base System Builder wizard to create an XPS project
Create New Blank Project
Create a new XPS project without using the Base System Builder

Figure 1-34: **Create New Blank Project**

2. Change Target Device to match the FPGA on board. Unselect Auto Instantiate Clock/Rest then click OK.

The Spartan 6 LX9 part is selected in the following example.



Create New XPS Project

New project
1- Set Desired Path For Project
Project file: C:\customer\autoesl\SP6_STICK\AXI_LITE\EDK_BFM\system.xmp [Browse ...]

Target device
2- Select Target Device

Architecture	Device Size	Package	Speed Grade
spartan6	xc6slx9	csg324	-2

Advanced options
☐ Import Design File (.mhs) from existing Project [Browse ...]
☐ Set Project Peripheral Repository Search Path [Browse ...]
☐ Set Custom Make File(instead of XPS generated MakeFile) [Browse ...]

Auto Instantiate Clock/Reset
3- Uncheck Auto Instantiate Clock/Reset
☐ AXI Clock Generator ☐ AXI Reset Module

4- Click OK
[OK] [Cancel] [Help]

Figure 1-35: **Create New XPS Project**

3. Copy the generated PCORE from AutoESL directory structure to XPS directory structure.

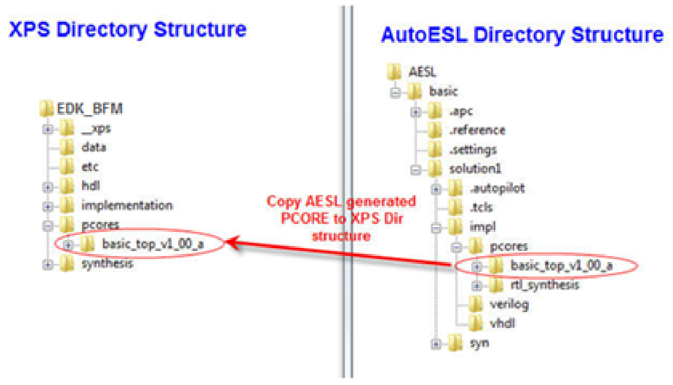


Figure 1-36: **AutoESL Directory Structure**

4. Add the following pcores into the blank XPS project by double click on each one of them:
   - AXI Interconnect
   - Basic_top
   - AXI4 Lite Master BFM
   - PCORE generated from AutoESL
   - PCORE used to simulate an AXI LITE Master (i.e. processor)

Note: If PCORE is not listed under the Project Local PCores then you will need to tell XPS to rescan user repository.

5. **Project > Rescan User Repository.**

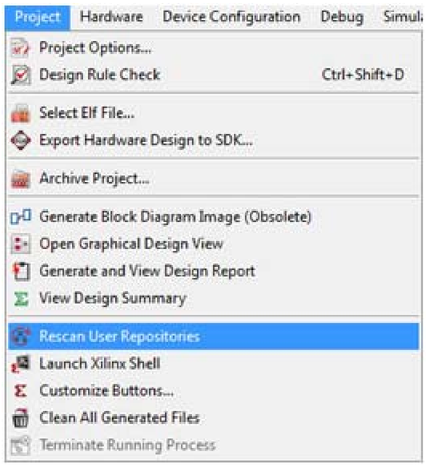


Figure 1-37: **Rescan User Repository**

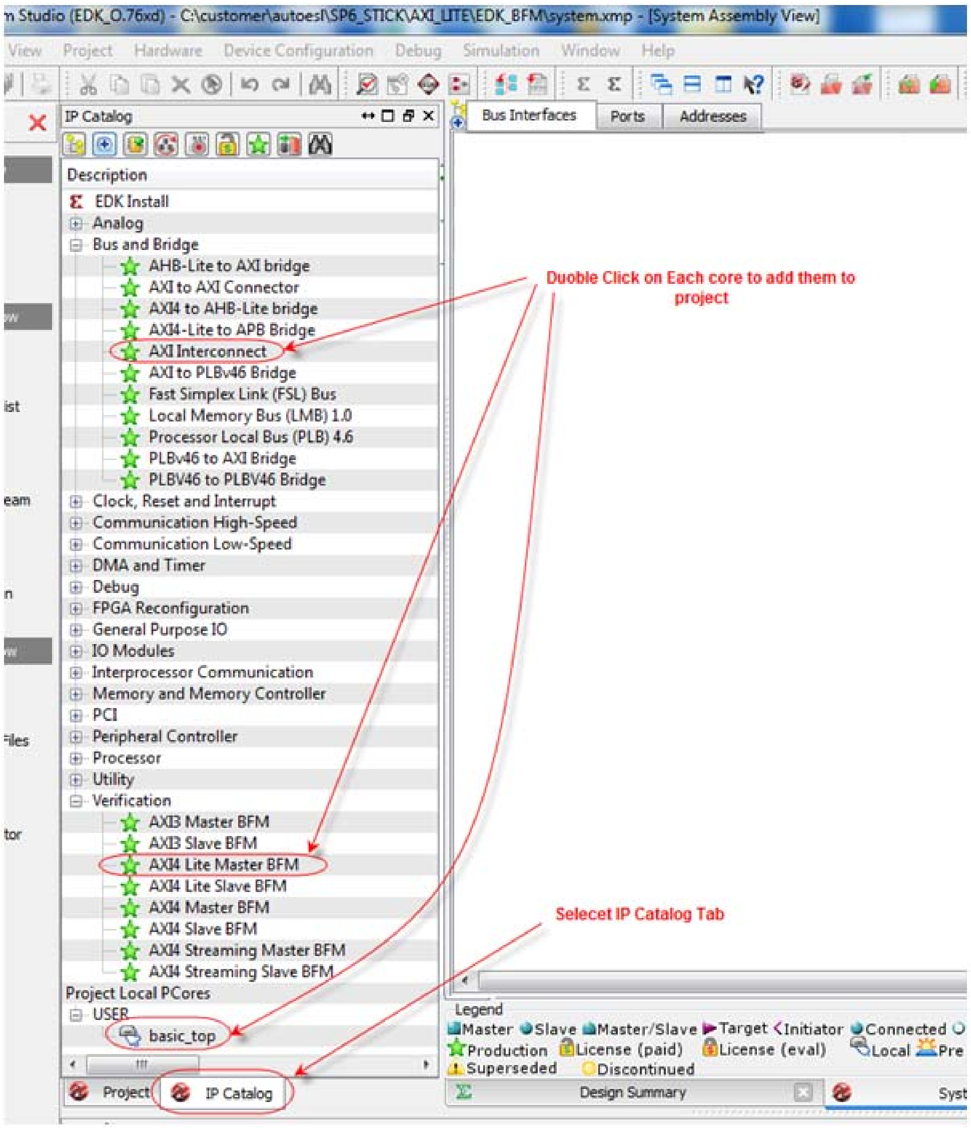
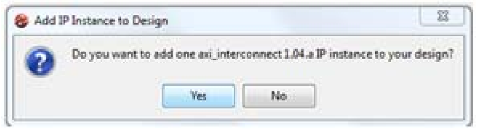


Figure 1-38: **IP Catalog Tab**

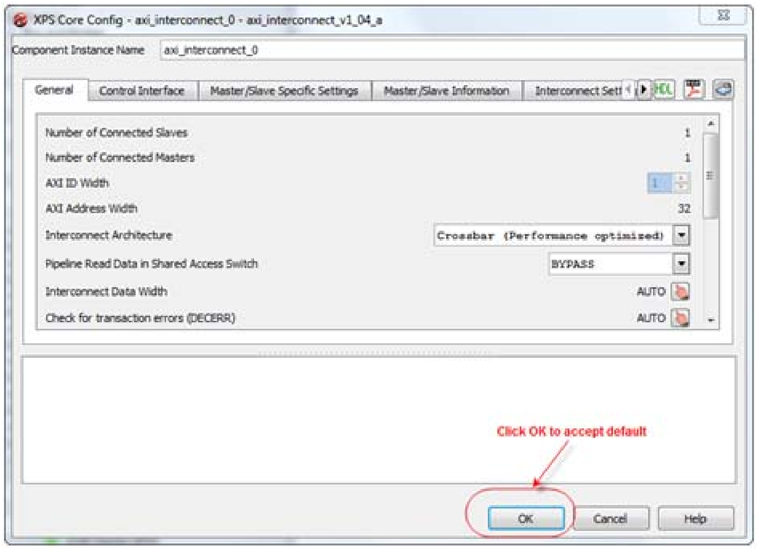
6. Double click **AXI Interconnect** and click **YES** when prompted to add PCORE.



Do you want to add one axi_interconnect 1.04.a IP instance to your design?

Figure 1-39: **Add IP Instance to Design**

When prompted to customize PCORE, click **OK** to accept default.

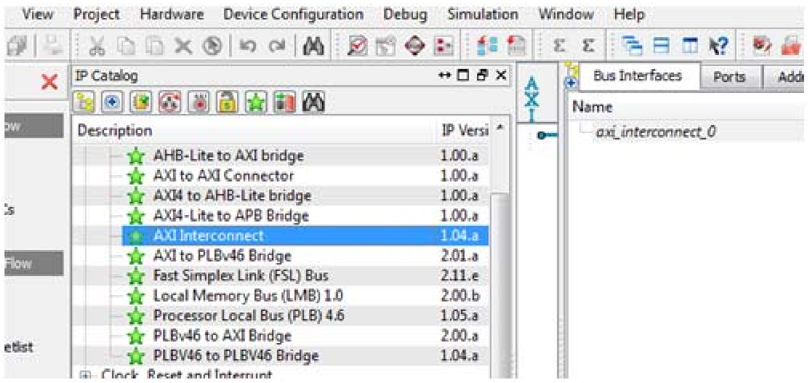

XPS Core Config - axi_interconnect_0 - axi_interconnect_v1_04_a

Component Instance Name: axi_interconnect_0

Number of Connected Slaves 1
Number of Connected Masters 1
AXI ID Width 1
AXI Address Width 32
Interconnect Architecture: Crossbar (Performance optimized)
Pipeline Read Data in Shared Access Switch: BYPASS
Interconnect Data Width: AUTO
Check for transaction errors (DECERR): AUTO

Figure 1-40: **XPS Core Config**

Figure below shows the added PCORE.

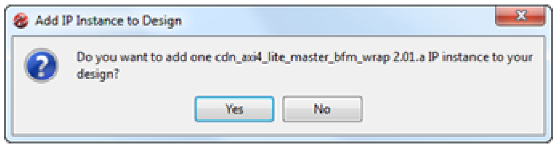


Description	IP Versi
AHB-Lite to AXI bridge	1.00.a
AXI to AXI Connector	1.00.a
AXI4 to AHB-Lite bridge	1.00.a
AXI4-Lite to APB Bridge	1.00.a
AXI Interconnect	1.04.a
AXI to PLBv46 Bridge	2.01.a
Fast Simplex Link (FSL) Bus	2.11.e
Local Memory Bus (LMB) 1.0	2.00.b
Processor Local Bus (PLB) 4.6	1.05.a
PLBv46 to AXI Bridge	2.00.a
PLBV46 to PLBV46 Bridge	1.04.a
Clock, Reset and Interrupt	

Name: axi_interconnect_0

Figure 1-41: **Added PCORE**

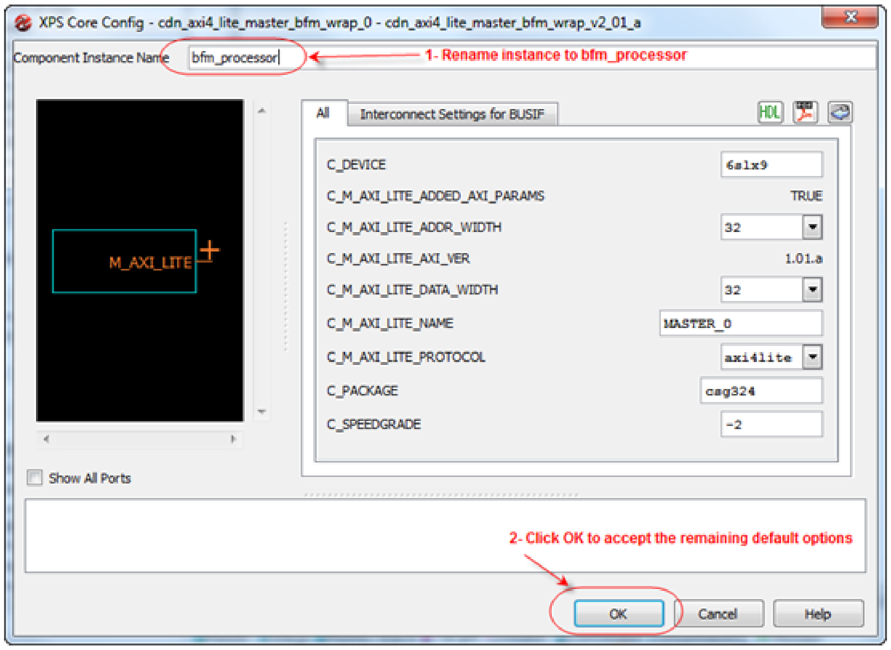
7. Double Click to add the AXI LIte Master BFM and click **YES** when prompted to add PCORE.


Add IP Instance to Design

Do you want to add one cdn_axi4_lite_master_bfm_wrap 2.01.a IP instance to your design?

Figure 1-42: **Add IP Instance to Design**

When prompted to customize PCORE, rename the component instance name to bfm_processor and click **OK**.


XPS Core Config - cdn_axi4_lite_master_bfm_wrap_0 - cdn_axi4_lite_master_bfm_wrap_v2_01_a

Component Instance Name: bfm_processor

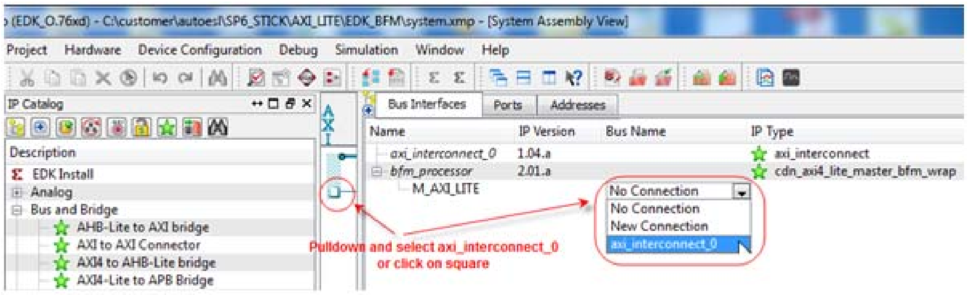
1- Rename instance to bfm_processor

C_DEVICE	6slx9
C_M_AXI_LITE_ADDED_AXI_PARAMS	TRUE
C_M_AXI_LITE_ADDR_WIDTH	32
C_M_AXI_LITE_AXI_VER	1.01.a
C_M_AXI_LITE_DATA_WIDTH	32
C_M_AXI_LITE_NAME	MASTER_0
C_M_AXI_LITE_PROTOCOL	axi4lite
C_PACKAGE	csg324
C_SPEEDGRADE	-2

2- Click OK to accept the remaining default options

Figure 1-43: **XPS Core Config**

Connect bfm_processor to the AXI interconnect as shown in figure below.

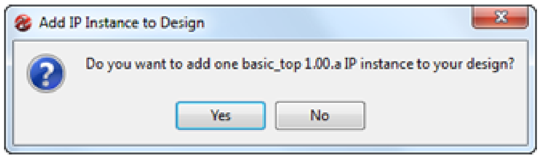

(EDK_0.76xd) - C:\customer\autoesl\SP6_STICK\AXI_LITE\EDK_BFM\system.xmp - [System Assembly View]

Name	IP Version	Bus Name	IP Type
axi_interconnect_0	1.04.a		axi_interconnect
bfm_processor	2.01.a		cdn_axi4_lite_master_bfm_wrap
M_AXI_LITE		No Connection	

Pulldown and select axi_interconnect_0 or click on square

Figure 1-44: **bmf_processor Connection**

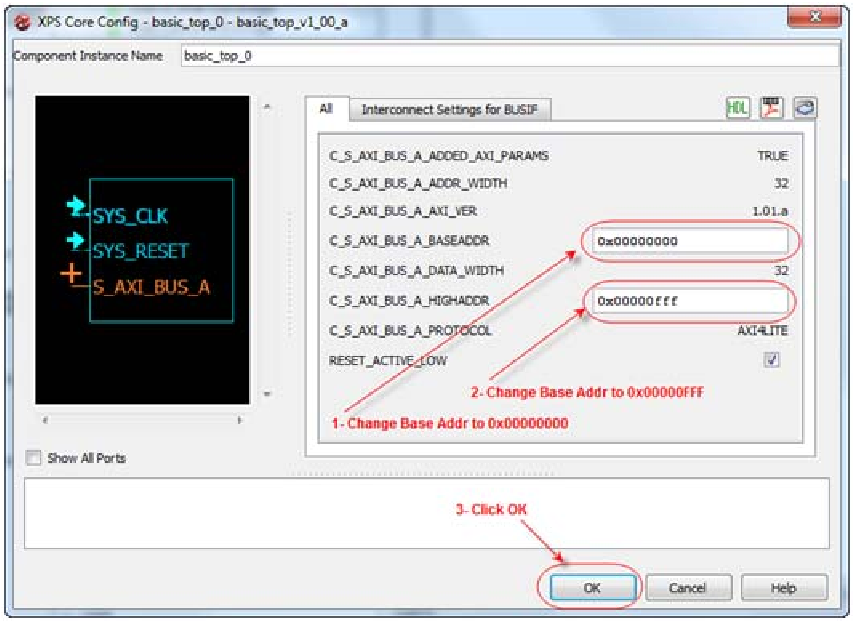
8. Double Click to add the basic_top PCORE and click **YES** when prompted to add PCORE.


Add IP Instance to Design

Do you want to add one basic_top 1.00.a IP instance to your design?

Figure 1-45: **basic_top PCORE**

When prompted to customize PCORE, change the C_S_AXI_BUS_A_BASEADDR to 0x00000000 and C_S_AXI_BUS_ to 0x00000FFF and click **OK**.

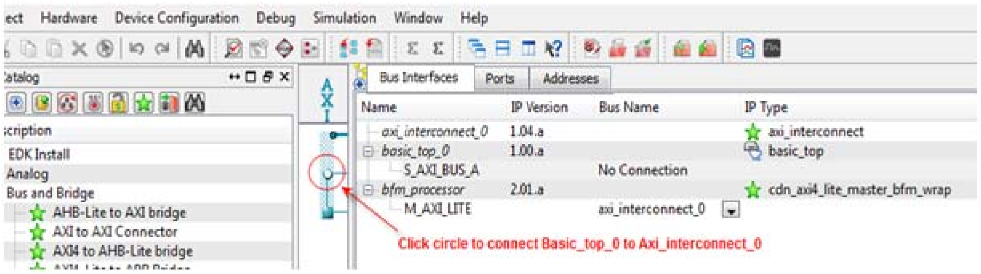

XPS Core Config - basic_top_0 - basic_top_v1_00_a
SYS_CLK
SYS_RESET
S_AXI_BUS_A
All
Interconnect Settings for BUSIF
Show All Ports
OK
Cancel
Help
1- Change Base Addr to 0x00000000
2- Change Base Addr to 0x00000FFF
3- Click OK

Component Instance Name: basic_top_0

C_S_AXI_BUS_A_ADDED_AXI_PARAMS	TRUE
C_S_AXI_BUS_A_ADDR_WIDTH	32
C_S_AXI_BUS_A_AXI_VER	1.01.a
C_S_AXI_BUS_A_BASEADDR	0x00000000
C_S_AXI_BUS_A_DATA_WIDTH	32
C_S_AXI_BUS_A_HIGHADDR	0x00000fff
C_S_AXI_BUS_A_PROTOCOL	AXI4LITE
RESET_ACTIVE_LOW	✓

Figure 1-46: **XPS Core Config**

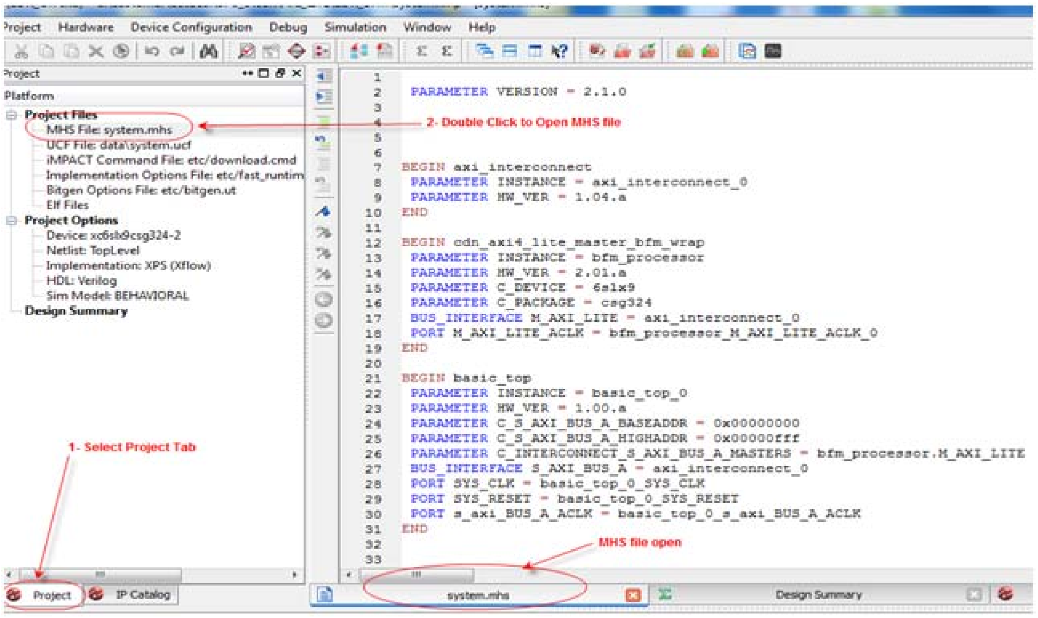
Connect basic_top_0 to the AXI interconnect as shown in figure below.



Name	IP Version	Bus Name	IP Type
axi_interconnect_0	1.04.a		axi_interconnect
basic_top_0	1.00.a		basic_top
S_AXI_BUS_A		No Connection	
bfm_processor	2.01.a		cdn_axi4_lite_master_bfm_wrap
M_AXI_LITE		axi_interconnect_0	

Figure 1-47: **AXI Interconnect**

9. Next step is to add the CLOCK and RESET connections for PCOREs. This step will be done by editing the MHS (Hardware system file) by hand.


Project Hardware Device Configuration Debug Simulation Window Help
Project
Platform
2- Double Click to Open MHS file
1- Select Project Tab
MHS file open
Project
IP Catalog
system.mhs
Design Summary

```
Project Files
    MHS File: system.mhs
    UCF File: data\system.ucf
    iMPACT Command File: etc/download.cmd
    Implementation Options File: etc/fast_runtim
    Bitgen Options File: etc/bitgen.ut
    Elf Files
Project Options
    Device: xc6slx9csg324-2
    Netlist: TopLevel
    Implementation: XPS (Xflow)
    HDL: Verilog
    Sim Model: BEHAVIORAL
Design Summary
```

```

 PARAMETER VERSION = 2.1.0



BEGIN axi_interconnect
 PARAMETER INSTANCE = axi_interconnect_0
 PARAMETER HW_VER = 1.04.a
END

BEGIN cdn_axi4_lite_master_bfm_wrap
 PARAMETER INSTANCE = bfm_processor
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_DEVICE = 6slx9
 PARAMETER C_PACKAGE = csg324
 BUS_INTERFACE M_AXI_LITE = axi_interconnect_0
 PORT M_AXI_LITE_ACLK = bfm_processor_M_AXI_LITE_ACLK_0
END

BEGIN basic_top
 PARAMETER INSTANCE = basic_top_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_S_AXI_BUS_A_BASEADDR = 0x00000000
 PARAMETER C_S_AXI_BUS_A_HIGHADDR = 0x00000fff
 PARAMETER C_INTERCONNECT_S_AXI_BUS_A_MASTERS = bfm_processor.M_AXI_LITE
 BUS_INTERFACE S_AXI_BUS_A = axi_interconnect_0
 PORT SYS_CLK = basic_top_0_SYS_CLK
 PORT SYS_RESET = basic_top_0_SYS_RESET
 PORT s_axi_BUS_A_ACLK = basic_top_0_s_axi_BUS_A_ACLK
END


```

Figure 1-48: **Editing MHS (Hardware System File) File**

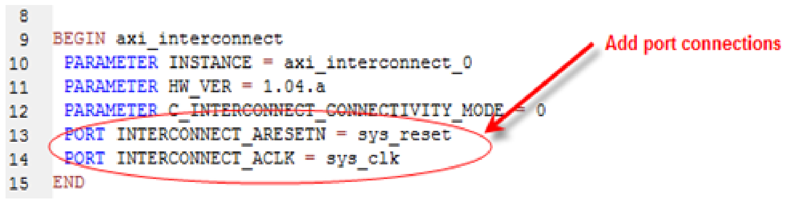
Add the sys_clk and sys_reset external port declarations at beginning of MHS file.

```
PARAMETER VERSION = 2.1.0

PORT sys_clk = sys_clk, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
PORT sys_reset = sys_reset, DIR = I, SIGIS = RST
```

Figure 1-49: **MHS File**

Connect sys_clk and sys_reset to axi_interconnect in MHS file.

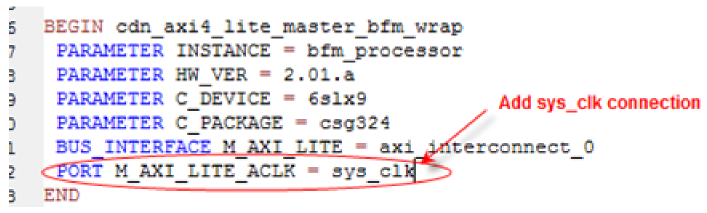


```

BEGIN axi_interconnect
 PARAMETER INSTANCE = axi_interconnect_0
 PARAMETER HW_VER = 1.04.a
 PARAMETER C_INTERCONNECT_CONNECTIVITY_MODE = 0
 PORT INTERCONNECT_ARESETN = sys_reset
 PORT INTERCONNECT_ACLK = sys_clk
END
```

Figure 1-50: **MHS File**

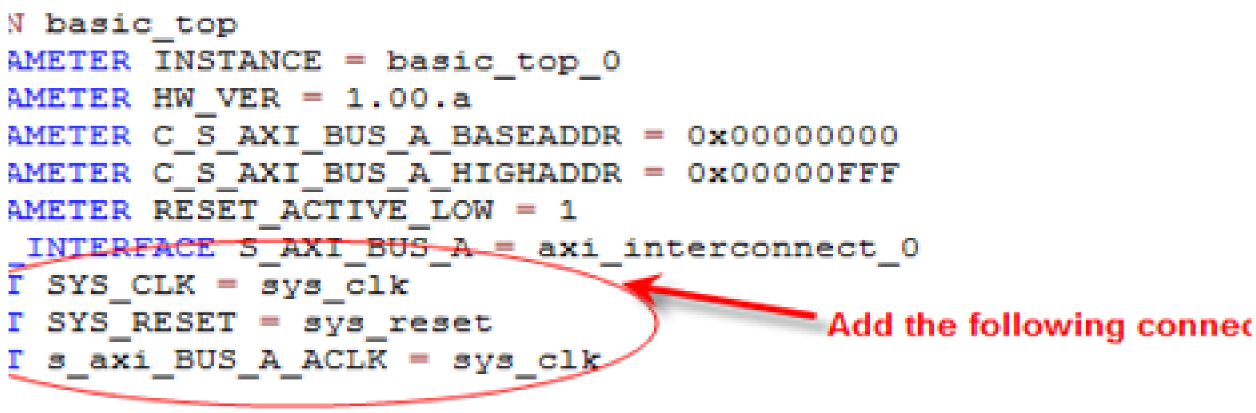
Connect sys_clk to bfm_processor PCORE in MHS file.



```
BEGIN cdn_axi4_lite_master_bfm_wrap
 PARAMETER INSTANCE = bfm_processor
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_DEVICE = 6slx9
 PARAMETER C_PACKAGE = csg324
 BUS_INTERFACE M_AXI_LITE = axi_interconnect_0
 PORT M_AXI_LITE_ACLK = sys_clk
END
```

Figure 1-51: **MHS File**

Connect the sys_clk and sys_rest to the AutoESL generated PCORE in MHS file.



```
N basic_top
AMETER INSTANCE = basic_top_0
AMETER HW_VER = 1.00.a
AMETER C_S_AXI_BUS_A_BASEADDR = 0x00000000
AMETER C_S_AXI_BUS_A_HIGHADDR = 0x00000FFF
AMETER RESET_ACTIVE_LOW = 1
_INTERFACE S_AXI_BUS_A = axi_interconnect_0
T SYS_CLK = sys_clk
T SYS_RESET = sys_reset
T s_axi_BUS_A_ACLK = sys_clk
```

Figure 1-52: **MHS File**

10. Generate the Simulation model.

    Set the Simulation Project option. In this example Verilog and behavioral are selected. Select **Project > Project** options.

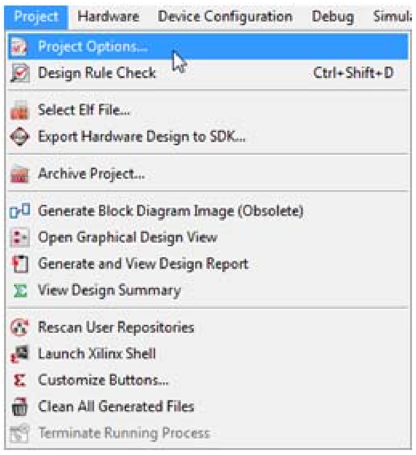

Project
Hardware
Device Configuration
Debug
Simul
Project Options...
Design Rule Check Ctrl+Shift+D
Select Elf File...
Export Hardware Design to SDK...
Archive Project...
Generate Block Diagram Image (Obsolete)
Open Graphical Design View
Generate and View Design Report
View Design Summary
Rescan User Repositories
Launch Xilinx Shell
Customize Buttons...
Clean All Generated Files
Terminate Running Process

Figure 1-53: **Project Options**

From the Project Options window select the following options:

- Design Flow
- Verilog
- Generate Test Bench Template
- Behavioral mode

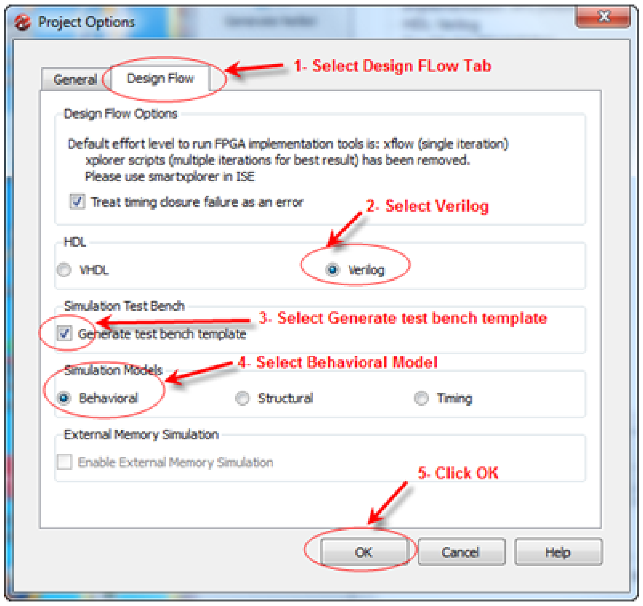

Project Options
General
Design Flow
1- Select Design FLow Tab
Design Flow Options
Default effort level to run FPGA implementation tools is: xflow (single iteration)
xplorer scripts (multiple iterations for best result) has been removed.
Please use smartxplorer in ISE
Treat timing closure failure as an error
2- Select Verilog
HDL
VHDL
Verilog
Simulation Test Bench
3- Select Generate test bench template
Generate test bench template
Simulation Models
4- Select Behavioral Model
Behavioral
Structural
Timing
External Memory Simulation
Enable External Memory Simulation
5- Click OK
OK
Cancel
Help

Figure 1-54: **Project Options**

To Generate the Simulation file click **Simulation > Generate Simulation HDL Files**.

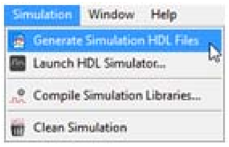
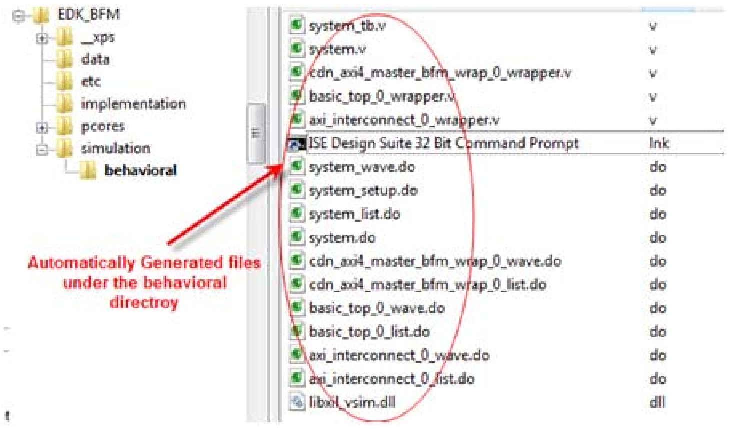

Simulation
Window
Help
Generate Simulation HDL Files
Launch HDL Simulator...
Compile Simulation Libraries...
Clean Simulation

Figure 1-55: **Simulation File**

This step will create the simulation directory structure to the XPS project.


Automatically Generated files under the behavioral directroy

```
EDK_BFM
  _xps
  data
  etc
  implementation
  pcores
  simulation
    behavioral
```

```
system_tb.v                                  v
system.v                                     v
cdn_axi4_master_bfm_wrap_0_wrapper.v         v
basic_top_0_wrapper.v                        v
axi_interconnect_0_wrapper.v                 v
ISE Design Suite 32 Bit Command Prompt       lnk
system_wave.do                               do
system_setup.do                              do
system_list.do                               do
system.do                                    do
cdn_axi4_master_bfm_wrap_0_wave.do           do
cdn_axi4_master_bfm_wrap_0_list.do           do
basic_top_0_wave.do                          do
basic_top_0_list.do                          do
axi_interconnect_0_wave.do                   do
axi_interconnect_0_list.do                   do
libxil_vsim.dll                              dll
```

Figure 1-56: **Simulation Directory Structure**

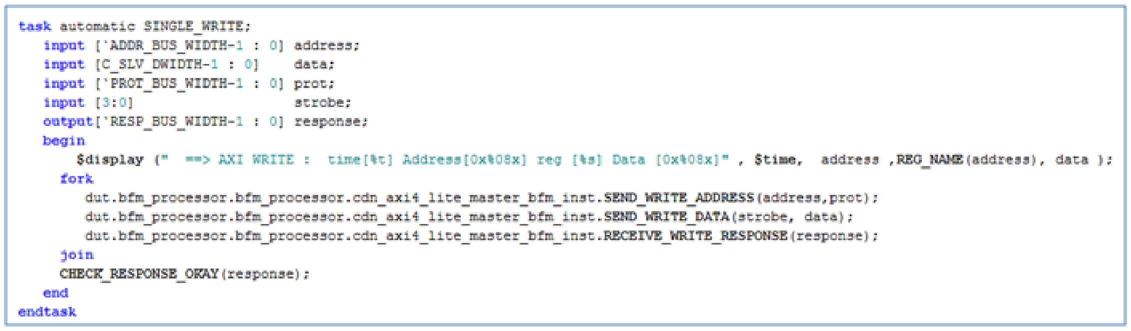
11. Edit the system_tb.v and add code to read/write the AutoESL generated PCORE.

    The system_tb.v is simply a template and the code has to be added to read/write the PCORE registers.

    The BFM for the AXI4-Lite Master has predefined API for TASK to initiate transactions on the AXI4 interface. For detailed information on API please refer to document AXI Bus Functional Model DS824.

    Two main tasks were added to the testbench to facilitate the reading/writing if the registers. These two tasks used a combination of the API defined in document DS824.

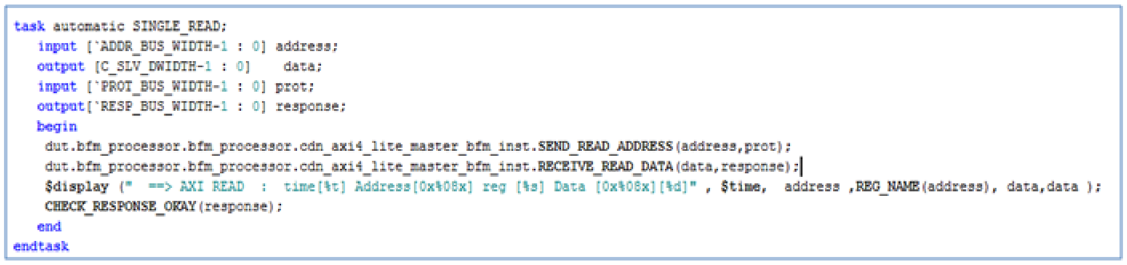
    The write task is as follows:



```
task automatic SINGLE_WRITE;
   input [`ADDR_BUS_WIDTH-1 : 0] address;
   input [C_SLV_DWIDTH-1 : 0]    data;
   input [`PROT_BUS_WIDTH-1 : 0] prot;
   input [3:0]                   strobe;
   output[`RESP_BUS_WIDTH-1 : 0] response;
   begin
      $display ("  ==> AXI WRITE :  time[%t] Address[0x%08x] reg [%s] Data [0x%08x]" , $time,  address ,REG_NAME(address), data );
     fork
        dut.bfm_processor.bfm_processor.cdn_axi4_lite_master_bfm_inst.SEND_WRITE_ADDRESS(address,prot);
        dut.bfm_processor.bfm_processor.cdn_axi4_lite_master_bfm_inst.SEND_WRITE_DATA(strobe, data);
        dut.bfm_processor.bfm_processor.cdn_axi4_lite_master_bfm_inst.RECEIVE_WRITE_RESPONSE(response);
     join
     CHECK_RESPONSE_OKAY(response);
   end
endtask
```

Figure 1-57: **Write Task**

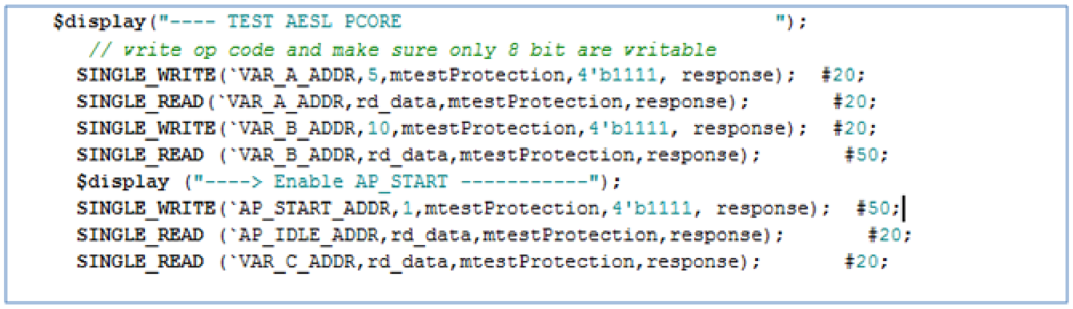
The read task is as follows:



```
task automatic SINGLE_READ;
   input [`ADDR_BUS_WIDTH-1 : 0] address;
   output [C_SLV_DWIDTH-1 : 0]    data;
   input [`PROT_BUS_WIDTH-1 : 0] prot;
   output[`RESP_BUS_WIDTH-1 : 0] response;
   begin
    dut.bfm_processor.bfm_processor.cdn_axi4_lite_master_bfm_inst.SEND_READ_ADDRESS(address,prot);
    dut.bfm_processor.bfm_processor.cdn_axi4_lite_master_bfm_inst.RECEIVE_READ_DATA(data,response);
    $display ("  ==> AXI READ  :  time[%t] Address[0x%08x] reg [%s] Data [0x%08x][%d]" , $time,  address ,REG_NAME(address), data,data );
    CHECK_RESPONSE_OKAY(response);
   end
endtask
```

Figure 1-58: **Read Task**

Testing for the PCORE is written using the above tasks. The figure below shows the example code.



```
$display("---- TEST AESL PCORE                              ");
   // write op code and make sure only 8 bit are writable
  SINGLE_WRITE(`VAR_A_ADDR,5,mtestProtection,4'b1111, response);  #20;
  SINGLE_READ(`VAR_A_ADDR,rd_data,mtestProtection,response);       #20;
  SINGLE_WRITE(`VAR_B_ADDR,10,mtestProtection,4'b1111, response);  #20;
  SINGLE_READ (`VAR_B_ADDR,rd_data,mtestProtection,response);       #50;
  $display ("----> Enable AP_START -----------");
  SINGLE_WRITE(`AP_START_ADDR,1,mtestProtection,4'b1111, response);  #50;
  SINGLE_READ (`AP_IDLE_ADDR,rd_data,mtestProtection,response);       #20;
  SINGLE_READ (`VAR_C_ADDR,rd_data,mtestProtection,response);       #20;
```

Figure 1-59: **Code Example**

Please refer to the file system_tb.v in the simulation directory for the complete code.

12. Running the simulation.

To run the AXI Bus functional simulation, the cadence AXI BFM PLI library needs to be downloaded and copied to the behavioral directory.

The library files can be downloaded using the following link:
https://secure.xilinx.com/webreg/clickthrough.do?filename=axi_bfm_ug_examples.tar.gz

Please refer to document DS824 for full details about the libraries. The windows library name is libxil_vsim.dll.

Note: The xil_vsim.dll library is compiled for 32 bit systems. Thus the modelsim simulator must be started from a 32bit command shell window.

The recommended way to use start a 32bit windows shell is to use the one provided under the ISE 32 bit command prompt from the accessory directory.

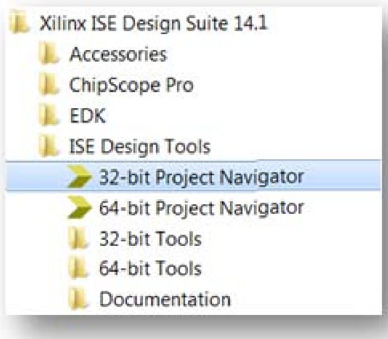
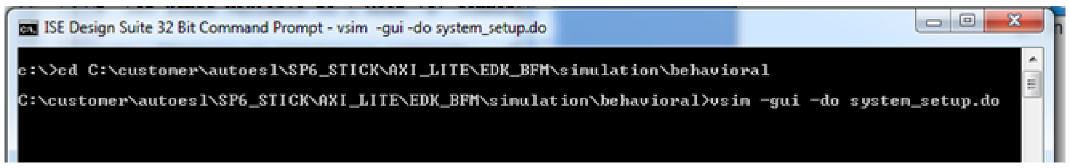


Figure 1-60: **32-bit Project Navigator**

In the shell window do the following:

- Cd into the behavioral directory
- Run modelsim in GUI mode and run system_setup.do by typing
- vsim -gui -do system_setup.do

This will start the Modelsim and run the system_setup.do TCL script.


ISE Design Suite 32 Bit Command Prompt - vsim -gui -do system_setup.do

```
c:\>cd C:\customer\autoesl\SP6_STICK\AXI_LITE\EDK_BFM\simulation\behavioral
C:\customer\autoesl\SP6_STICK\AXI_LITE\EDK_BFM\simulation\behavioral>vsim -gui -do system_setup.do
```

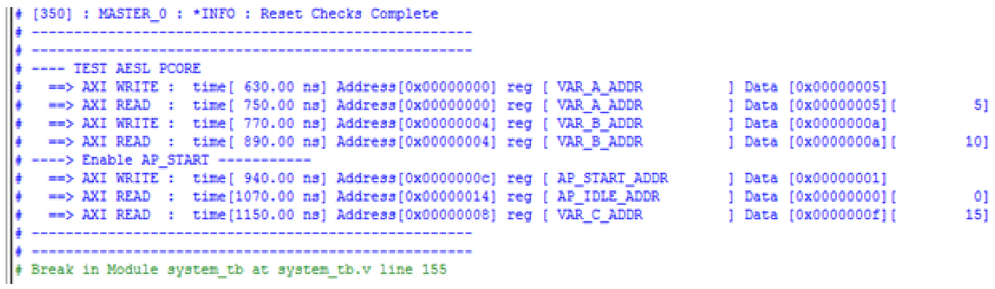
Figure 1-61: **system_setup.do TCL Script**

Override the system_tb.v in the behavioral directory with the one in the simulation directory. (override the template one).

In the modelsim transcript window type the following:

- c: This will compile the design files
- s: This will load the design for simulation
- w: This will open a wave window
- run –all: This will run the test bench

The figure below shows the output results on the transcript window.



```
# [350] : MASTER_0 : *INFO : Reset Checks Complete
# ----------------------------------------------------
# ----------------------------------------------------
# ---- TEST AESL PCORE
#   ==> AXI WRITE :  time[ 630.00 ns] Address[0x00000000] reg [ VAR_A_ADDR         ] Data [0x00000005]
#   ==> AXI READ  :  time[ 750.00 ns] Address[0x00000000] reg [ VAR_A_ADDR         ] Data [0x00000005][         5]
#   ==> AXI WRITE :  time[ 770.00 ns] Address[0x00000004] reg [ VAR_B_ADDR         ] Data [0x0000000a]
#   ==> AXI READ  :  time[ 890.00 ns] Address[0x00000004] reg [ VAR_B_ADDR         ] Data [0x0000000a][        10]
# ----> Enable AP_START -----------
#   ==> AXI WRITE :  time[ 940.00 ns] Address[0x0000000c] reg [ AP_START_ADDR      ] Data [0x00000001]
#   ==> AXI READ  :  time[1070.00 ns] Address[0x00000014] reg [ AP_IDLE_ADDR       ] Data [0x00000000][         0]
#   ==> AXI READ  :  time[1150.00 ns] Address[0x00000008] reg [ VAR_C_ADDR         ] Data [0x0000000f][        15]
# ----------------------------------------------------
# ----------------------------------------------------
# Break in Module system_tb at system_tb.v line 155
```

Figure 1-62: **Transcript Window**

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this guide:

- Avnet Microboard Kit: http://www.em.avnet.com/en-us/design/drc/Pages/Xilinx-Spartan-6-FPGA-LX9-MicroBoard.aspx
- AutoESL High-Level Synthesis Tool: http://www.xilinx.com/tools/autoesl.htm
- Platform Studio and the Embedded Development Kit: http://www.xilinx.com/tools/platform.htm
- Xilinx Software Development Kit (SDK): http://www.xilinx.com/tools/sdk.htm