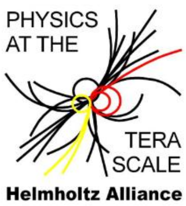
5th Detector Workshop of the Helmholtz Alliance “Physics at the Terascale”
14 – 16 March 2012, Physikalisches Institut Universität Bonn
FPGA School



Embedded System Design Lab Course

T. Hemperek, H. Krüger, M. Lemarenko, M. Schnell
Bonn University

Outline of the Lab Course

Tutorial 1: Creating an embedded system

Tutorial 2: Adding EDK IP

Tutorial 3: Adding custom IP

Tutorial 4: Embedded system simulation

Tutorial 5: Embedded ChipScope debugging

Tutorial 6: MicroBlaze SPI flash bootloader

- Step-by-step walk through tutorials
- All tutorials build upon another
- For every tutorial there is a pre-compiled solution found in the directory **EDK132_Lab<X>_Solution** (X standing for the number of the tutorial)
- The solutions are accumulative: **EDK132_Lab<N>_Solution** includes the solutions of all tutorials M with M<N. For example, if you want to do Tutorial 4 and you haven't done Tutorial 3 you can start from EDK132_Lab3_Solution.
- The project output files from that directory can be used to avoid long compile times (**pre-compiled** core, ask instructors)

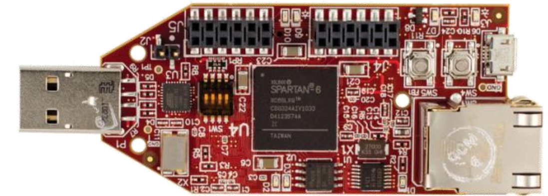
General Remarks

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

- **Verilog** is used as HDL option for this tutorial – with only a few exceptions (VHDL would be supported by the tool chain as well)
- Getting started (**Linux**):
  - user: *fpgaschool* password: *bonn2012*
  - To set up environment type ***xilinx*** in your shell
  - Some examples need a **terminal application**. Use ***cutecom*** and set device to **/dev/ttyUSB0**
- Location of the **tutorial files**
  - user created project files: *~/EDK/EDK_Tutorial*
  - solution files (precompiled): *~/EDK/EDK132_Lab<X>_Solution/EDK_Tutorial*
- **Start scripts** for the Xilinx executables
  - **ISE (Project Navigator):** *ise*
  - **XPS (Xilinx Platform Studio):** *xps*
    should always be called from within **ISE** to load proper project settings
  - **SDK (Software Development Kit):** *xsdk*

Prerequisites

- Hardware
  - Spartan-6 LX9 MicroBoard (http://em.avnet.com/s6microboard)
  - Mini-USB cable
  - Ethernet cable

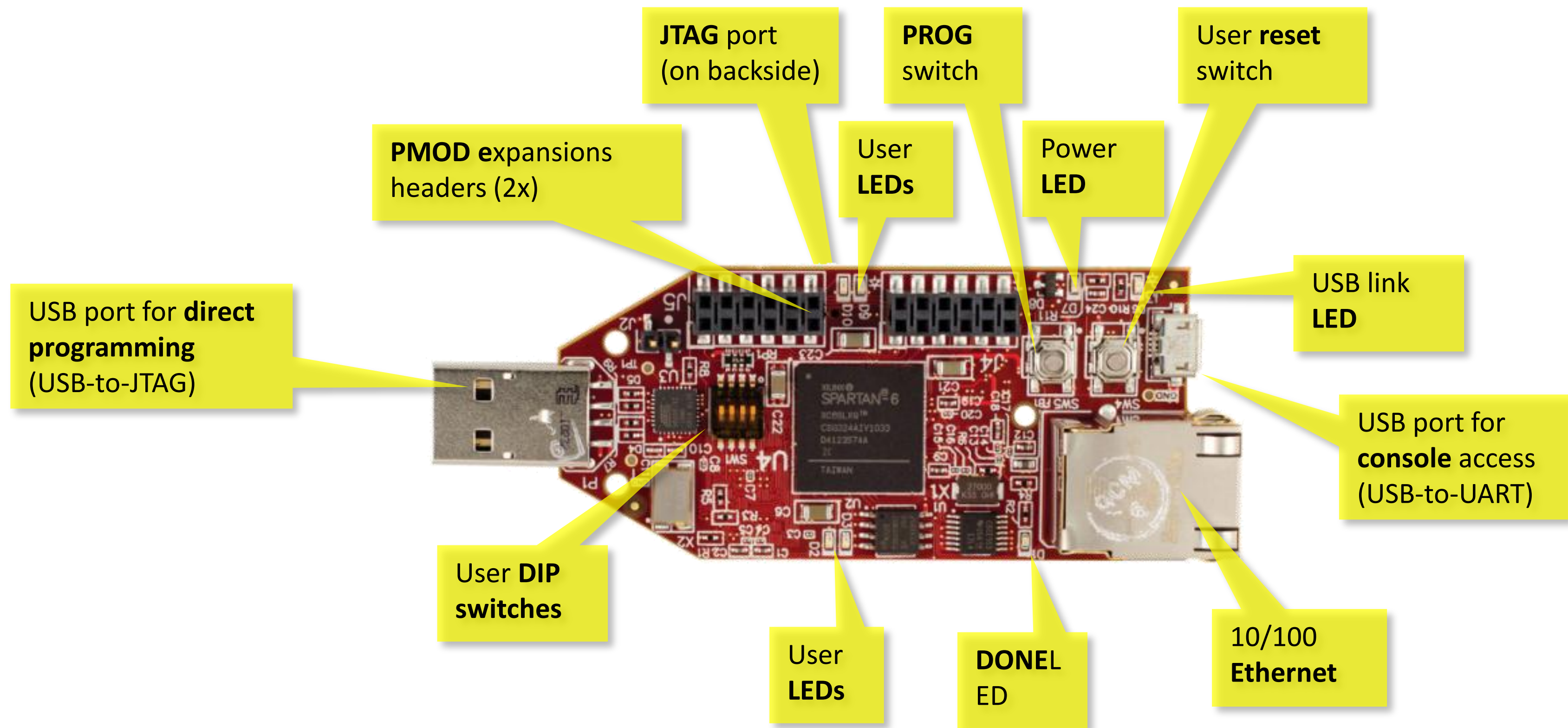


- Software
  - ISE WebPack (http://www.xilinx.com/tools/webpack.htm) with EDK add-on or ISE Embedded Edition version 13.2
  - USB-to-UART driver (Silicon Labs CP210x)
  - USB-to-JTAG driver (Digilent driver)
  - Xilinx board support files for Spartan-6 LX9 MicroBoard XBD (for EDK 13.2) files (available from Avnet)

- Additional reading
  - Support files download (registration required): http://em.avnet.com/s6microboard (also this lab course is based on the tutorials from AVNET)

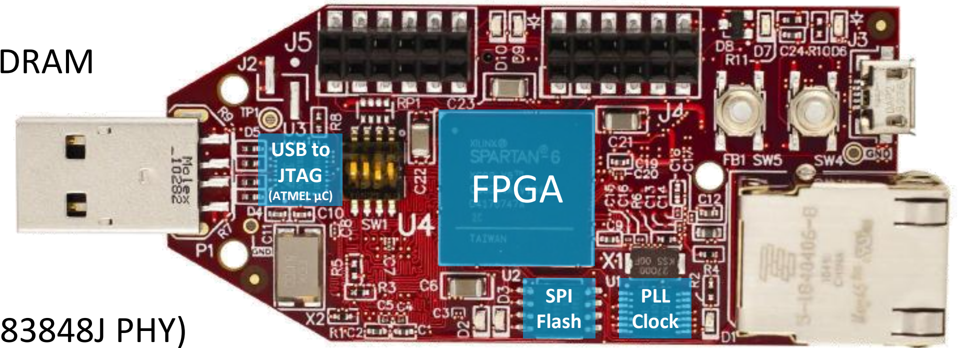
Spartan-6 LX9 Microboard IO Devices

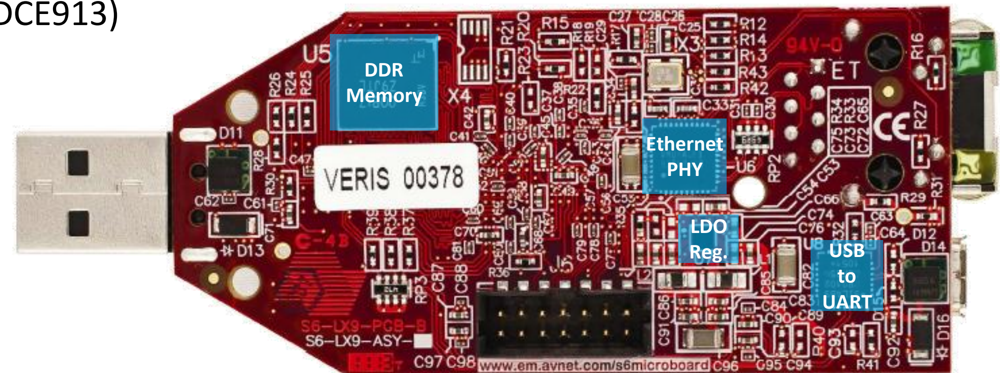
PHYSICS AT THE TERA SCALE
Helmholtz Alliance



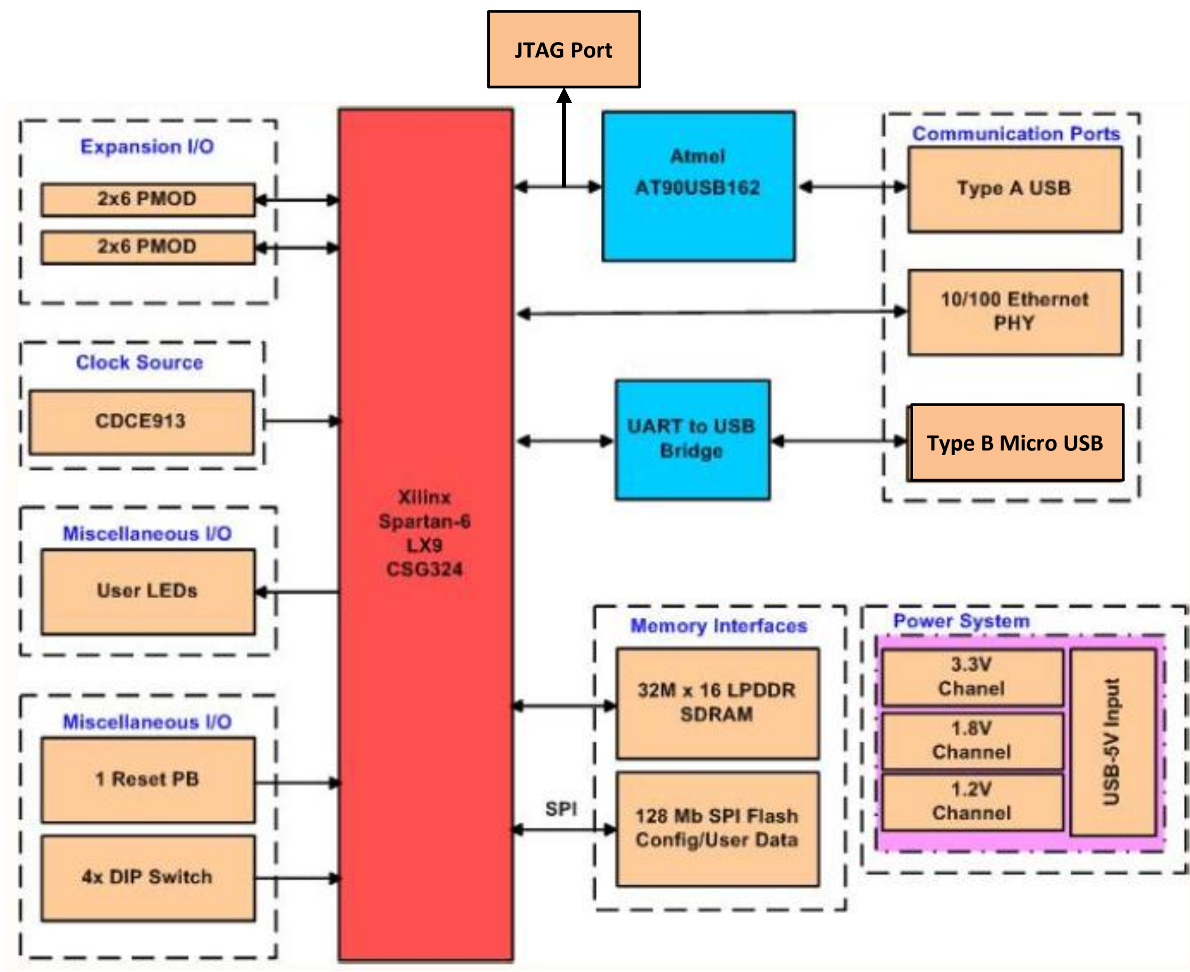
Spartan-6 LX9 Microboard Components

- FPGA
  - Xilinx Spartan-6 XC6SLX9-2CSG324C
- Memory
  - Micron 32 Mb x 16 (512 Mb) LPDDR Mobile SDRAM
  - 128 Mb Micron Multi-I/O SPI Flash
- Communication
  - FS USB-to-UART bridge (Silicon Labs CP2102)
  - FS USB-to- JTAG bridge (Atmel AT90USB162)
  - 10/100 Ethernet (National Semiconductor DP83848J PHY)
- Clocks
  - PLL, triple output, user programmable (TI CDCE913)





Spartan-6 LX9 Microboard Block Diagram



Tutorial 1: Creating an Embedded System

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

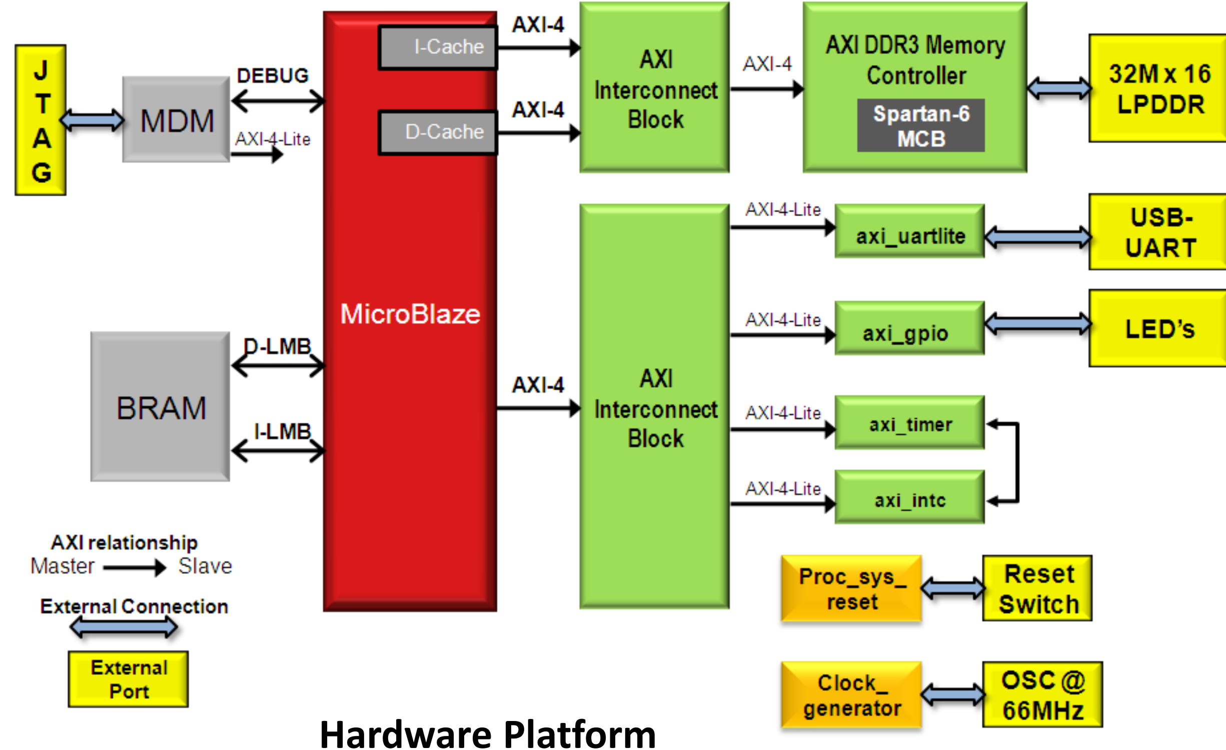
Scope of the tutorial

1.1 Generating the hardware platform

1.2 Understanding what has been created

1.3 Using the SDK to compile a software application

1.4 Testing the system on the FPGA

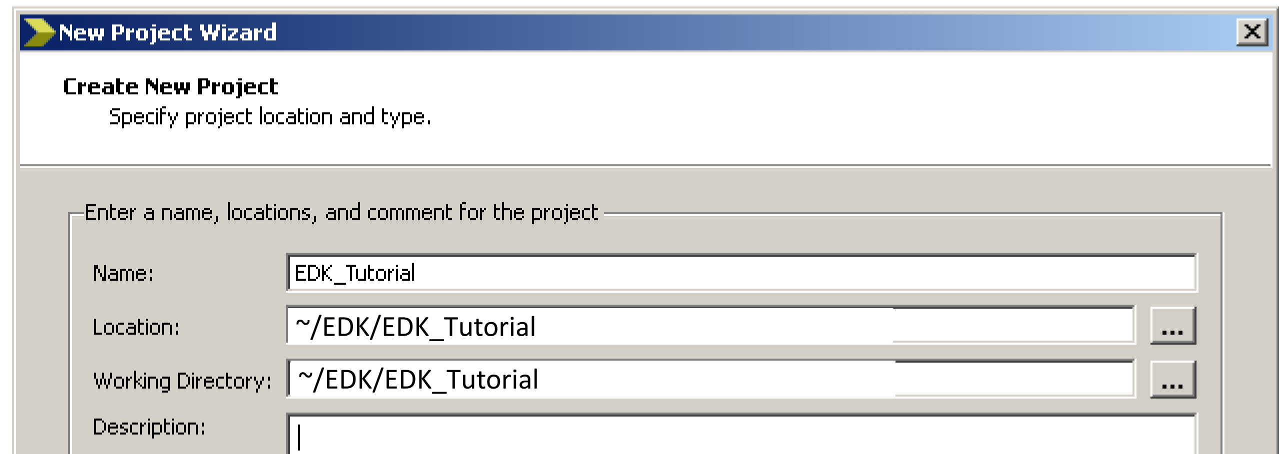


Hardware Platform

1.1 Creating the Hardware Platform

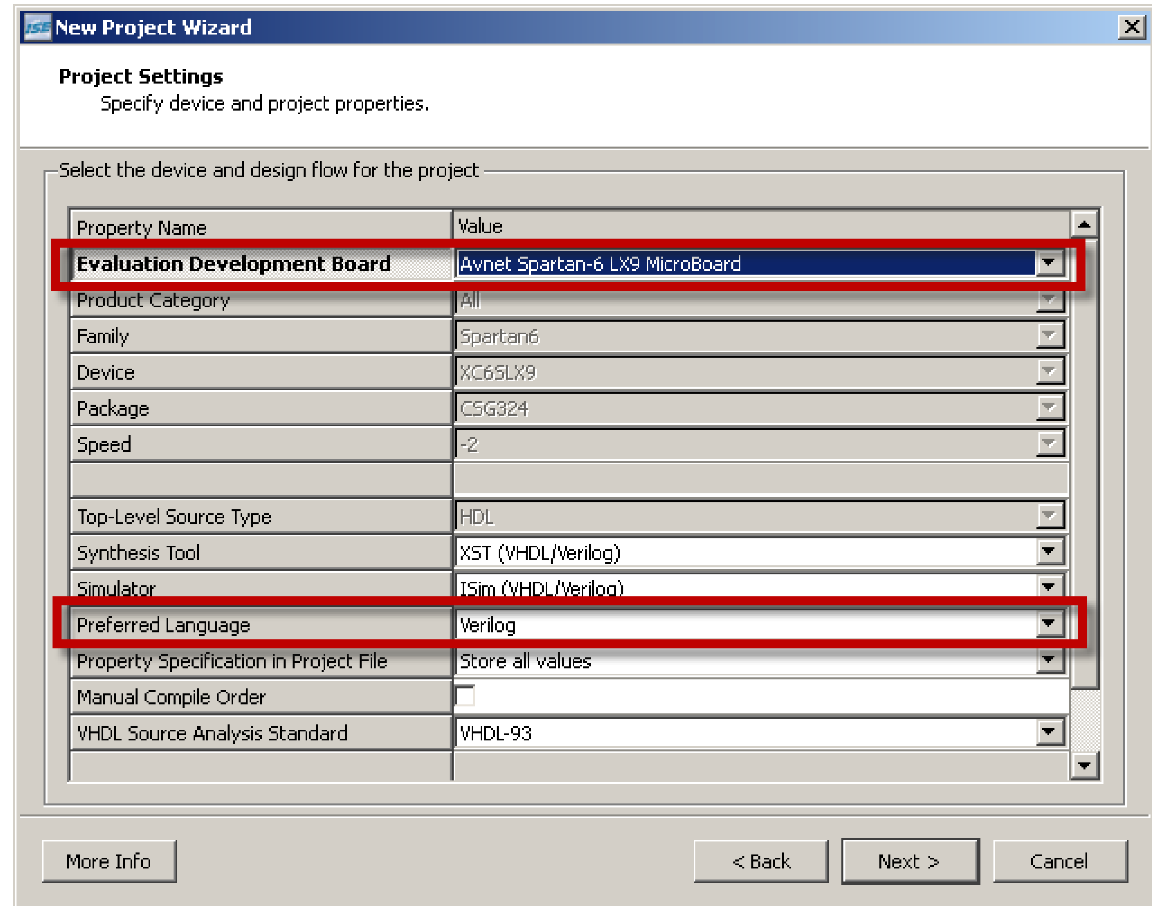
Use the **Base System Builder** to create all project files needed for an embedded microcontroller (MicroBlaze) based system.

1. To start the Xilinx ISE Project Navigator, type ***ise*** and create a new project*:* ***File > New Project…***
2. Set the Project Location to ***~/EDK*** and the Project Name to ***EDK_Tutorial***. Click Next.



1.1 Creating the Hardware Platform

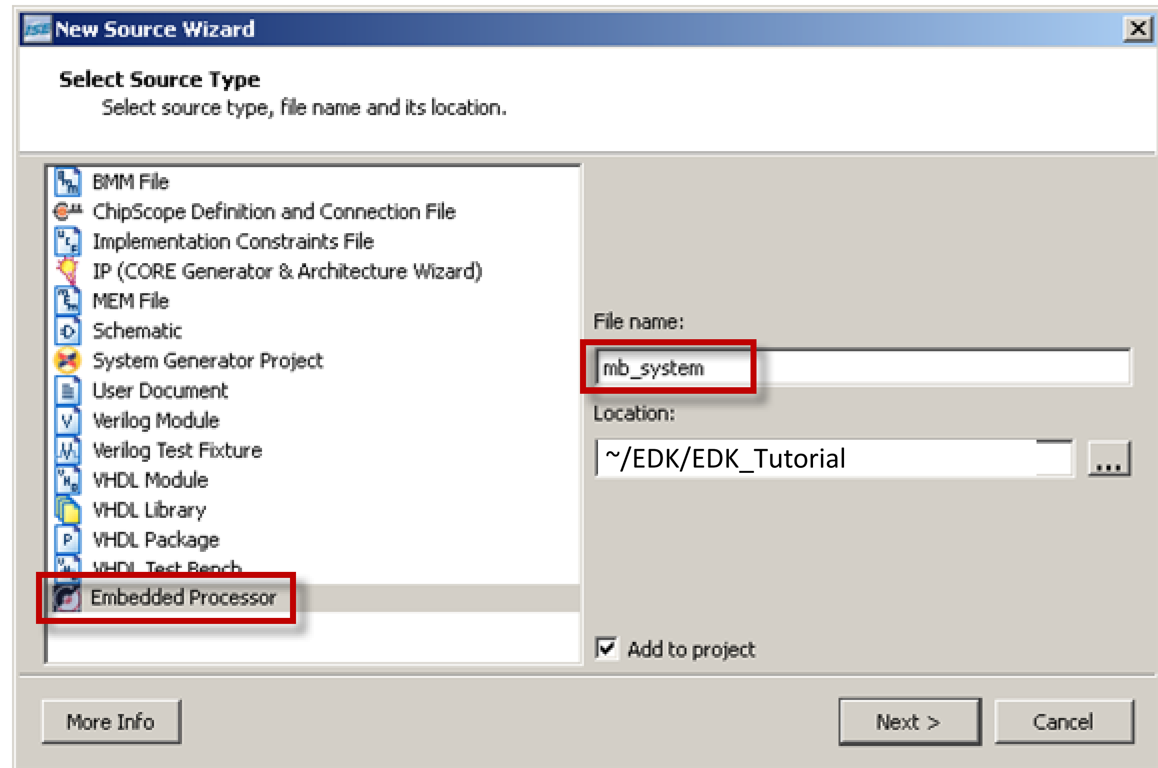
3. Select ***Spartan6, XC6SLX9, CSG324, -2***. Select ***Verilog*** as the Preferred Language. Click ***Next***.
4. Click ***Finish***



1.1 Creating the Hardware Platform

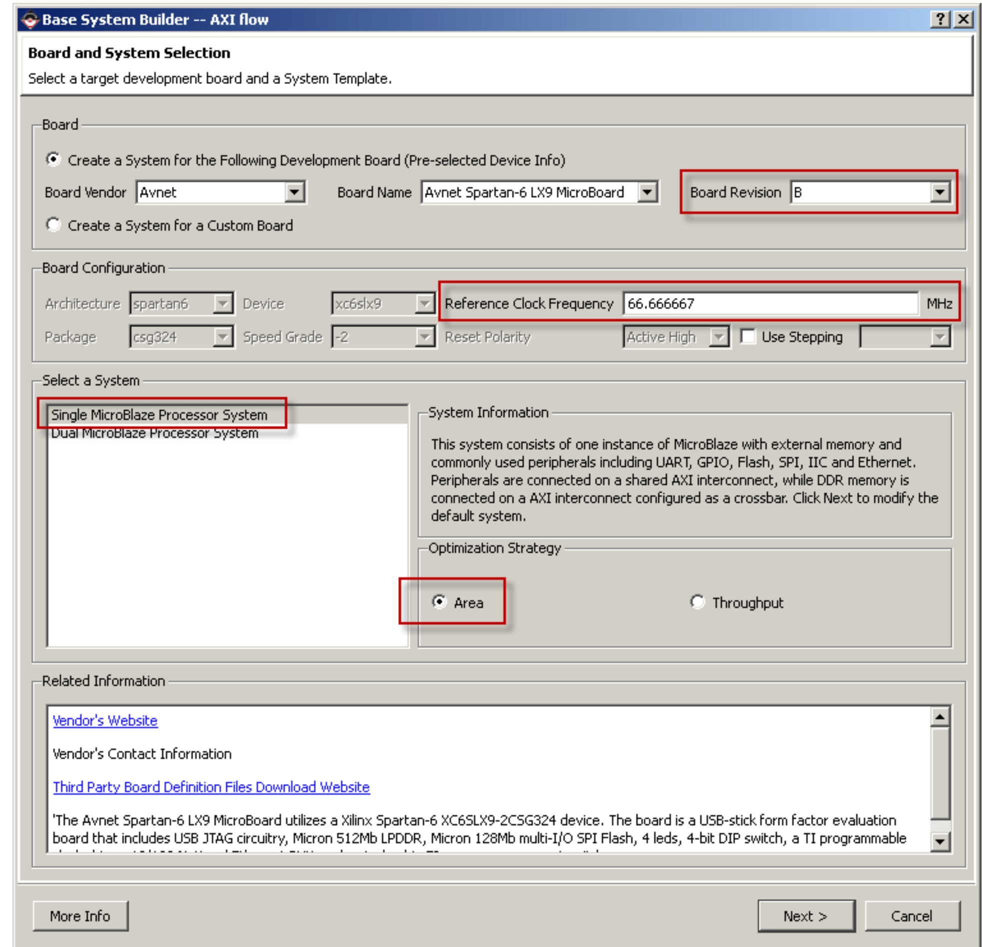
PHYSICS AT THE TERA SCALE
Helmholtz Alliance

5. Go to ***Project > New Source…*** then select ***Embedded Processor***. Type ***mb_system*** for the File name. Click ***Next.*** Click ***Finish.***



1.1 Creating the Hardware Platform

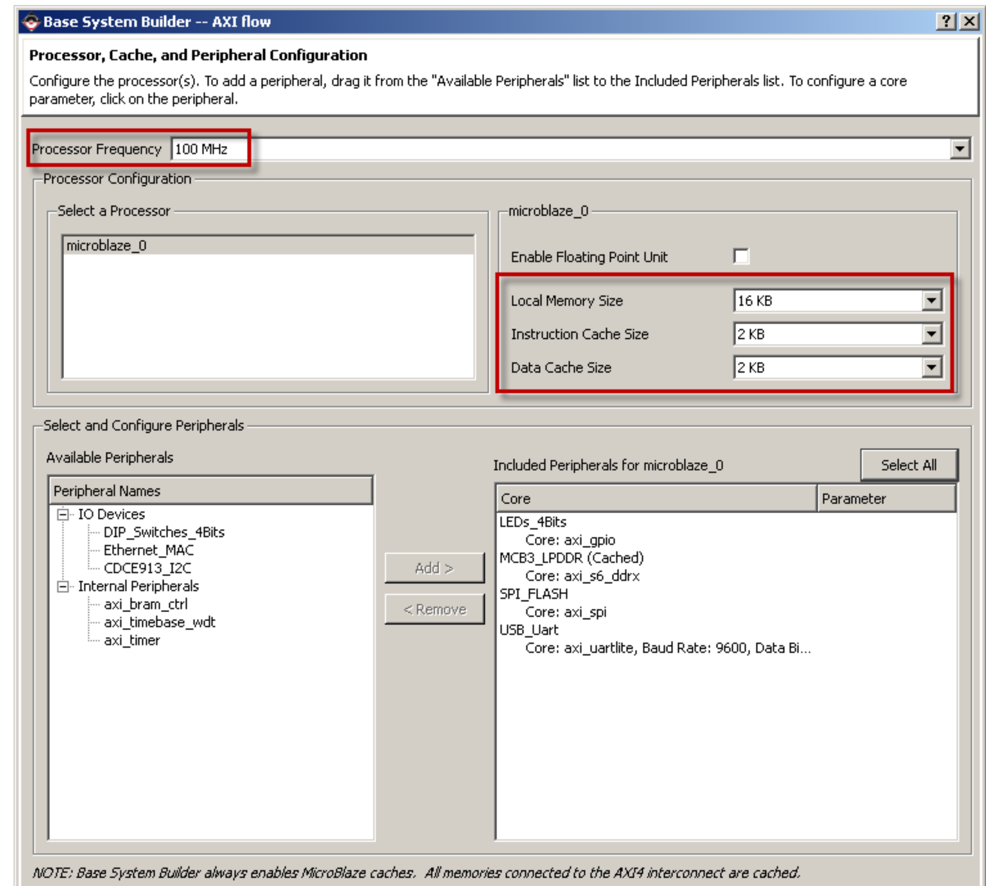
6. A message will appear asking if you want to create a **Base System** using the BSB wizard, click ***Yes***.
7. Select the **AXI system** then click ***OK***.
8. In the **Welcome** window click ***Next*** to create a new design.
9. In the Board and System Selection Window select ***Avnet Spartan-6 LX9 MicroBoard***. Configure the options as shown. Click ***Next.***



PHYSICS AT THE TERA SCALE
Helmholtz Alliance

1.1 Creating the Hardware Platform

11. In the **Processor, Cache and Peripheral Configuration** window configure the following options
    - **100 MHz** Processor Frequency
    - **16KB** Local Memory.
    - **2KB** Instruction Cache Memory
    - **2KB** Data Cache Memory
    - Remove **CDCE913_I2C** core
    - Remove **DIP_Switch_4Bits** core
    - Remove **Ethernet_MAC** core
    - Click ***Finish***.



1.2 Understanding the System

Use the **Xilinx Platform Studio** IDE to generate the embedded hardware platform.

Address tab shows the address map for the system

Bus Interfaces tab shows peripheral connections to the processor

Ports tab lists internal and external connections

System Assembly View shows each peripheral used and the connections between the peripherals when the Bus Interfaces tab is selected.

Project window provides information on the project options used, gives access to the main project files, and log files

- **MHS File**, microprocessor hardware specifcation
- **UCF File**, user constraints file

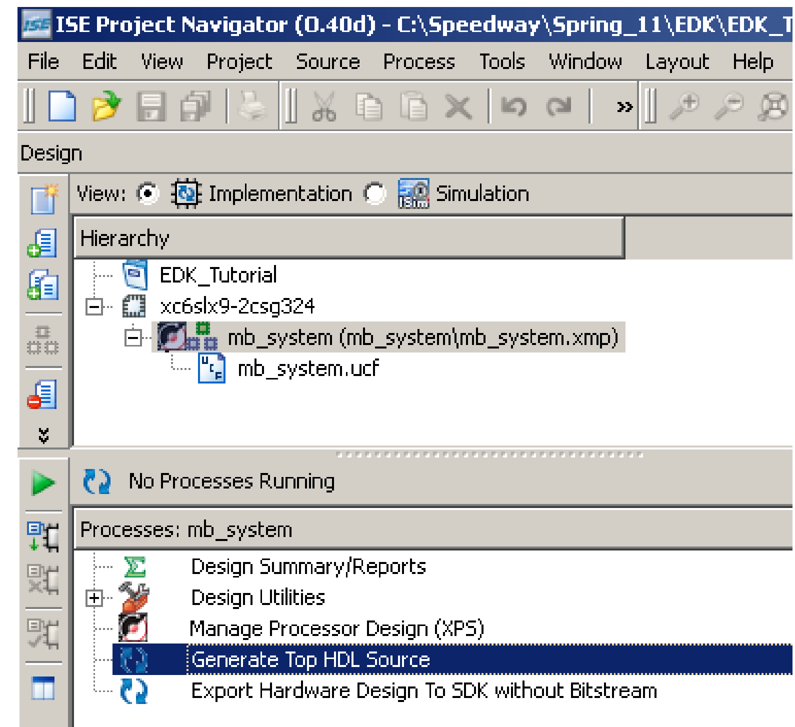
1.2 Understanding the System

PHYSICS AT THE TERA SCALE Helmholtz Alliance

- Select ***Project →Generate Block Diagram Image*** to view the block diagram for the project. The block diagram shows the connections between the different busses and components in the system. A jpeg image of the block diagram gets saved in your project in a folder named *blockdiagram*
- A datasheet of the system can also be generated. Go to ***Project →Generate and View Design Report*** to view the design report. This is an html file that is generated in a *report* subfolder.
- To view the general project options go to ***Project > Project Options***… Click ***Cancel*** to close the window.
- ***Close*** XPS when finished.

1.2 Understanding the System

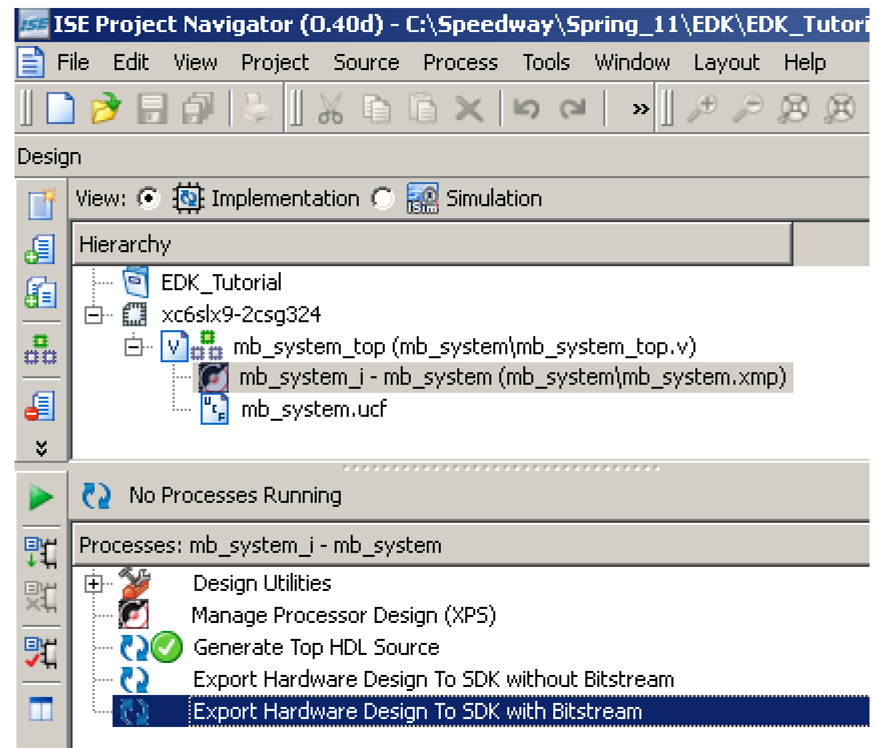
1. Return to Project Navigator and ***Go to Project → Add Copy of Source*** to add the FPGA constraints to the project.
2. Select **mb_system.ucf** in the *mb_system\data* directory. Click **OK**.
3. Select **mb_system** in the hierarchy window.
4. ***Double-click*** on ***Generate Top HDL Source*** in the Processes window**.** This creates a HDL instantiation template for your MicroBlaze processor subsystem and instantiates into a new created Verilog module.



1.3 Compiling a Test Application Using SDK

Xilinx SDK is an Eclipse based software development environment to create and debug software applications. Features include project management, multiple build configurations, C/C++ code editor, error navigation, a debugging and profiling environment, and source code version control.

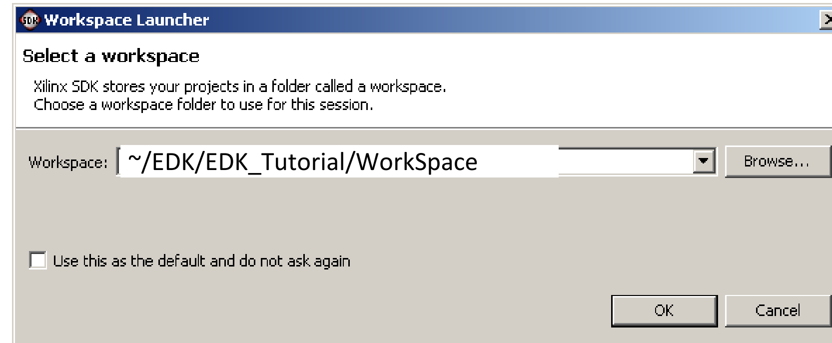
1. Select ***mb_system_i*** in the Hierarchy window. ***Double-click*** on ***Export Hardware To SDK with Bitstream*** in the Processes window. This may take several minutes to complete.



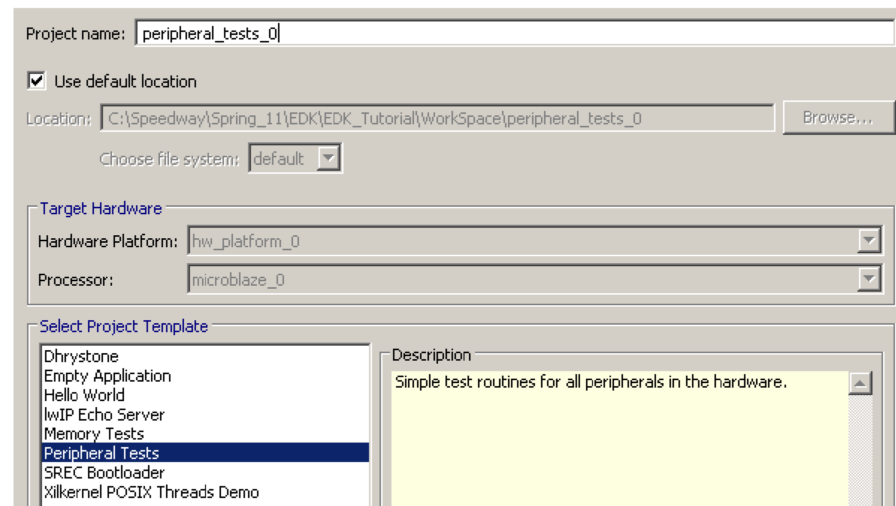
1.3 Compiling a Test Application Using SDK

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

2. Create a Workspace named **WorkSpace** in the *EDK_Tutorial* directory. Click ***OK***.



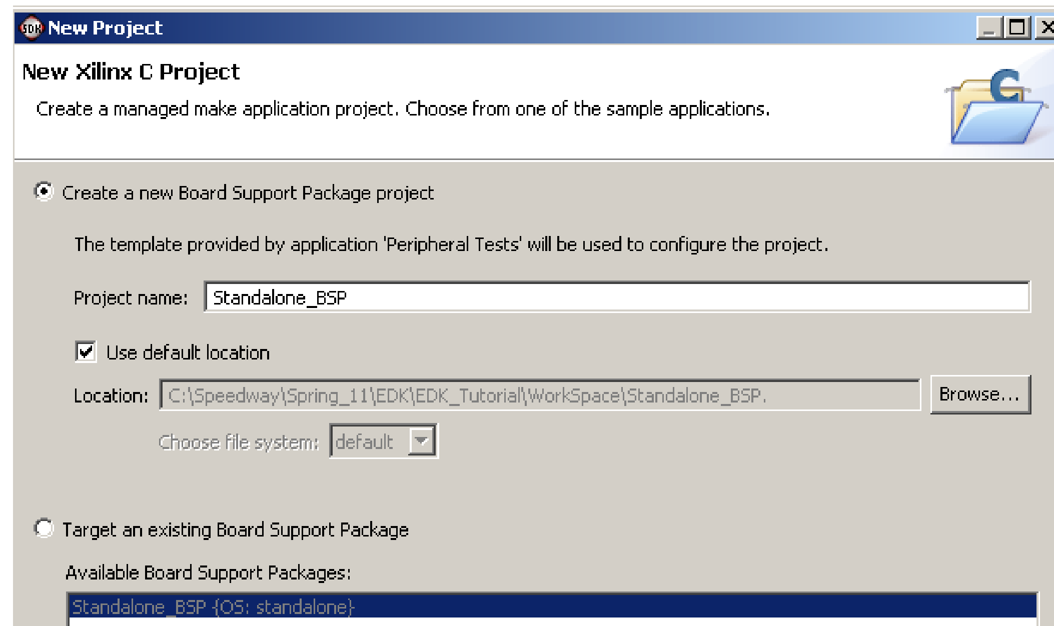
3. You can close the Welcome window on the right side of the screen.
4. Go to ***File > New > Xilinx C Project*** to create a new C project.
5. Select **Peripheral Tests** application from the project templates then click ***Next***.



1.3 Compiling a Test Application Using SDK

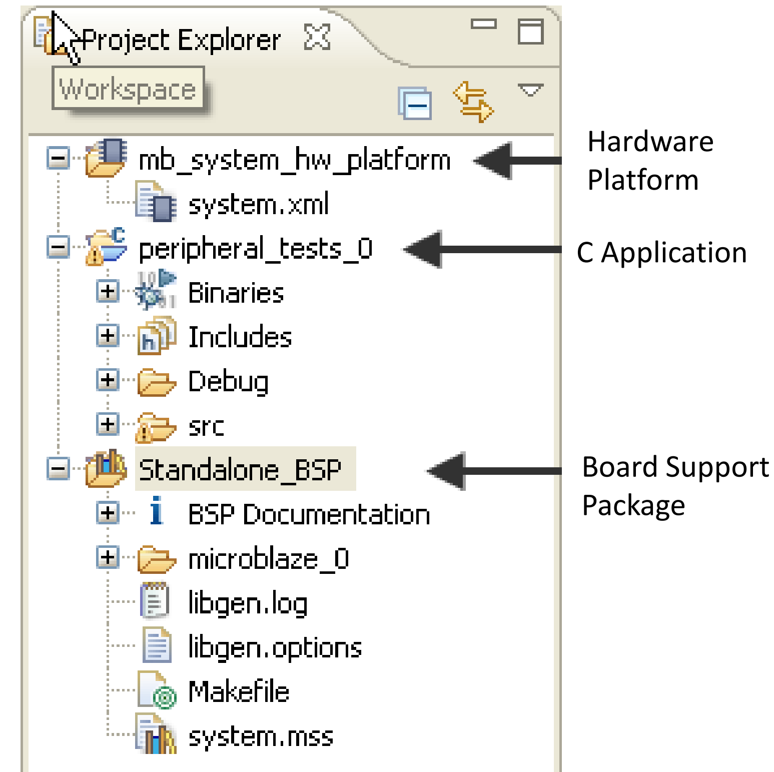
PHYSICS AT THE TERA SCALE
Helmholtz Alliance

6. Change the Board Support Package project name to **Standalone_BSP**.
7. Click ***Finish***. The application will start building and create an ELF file, which is the compiled application.



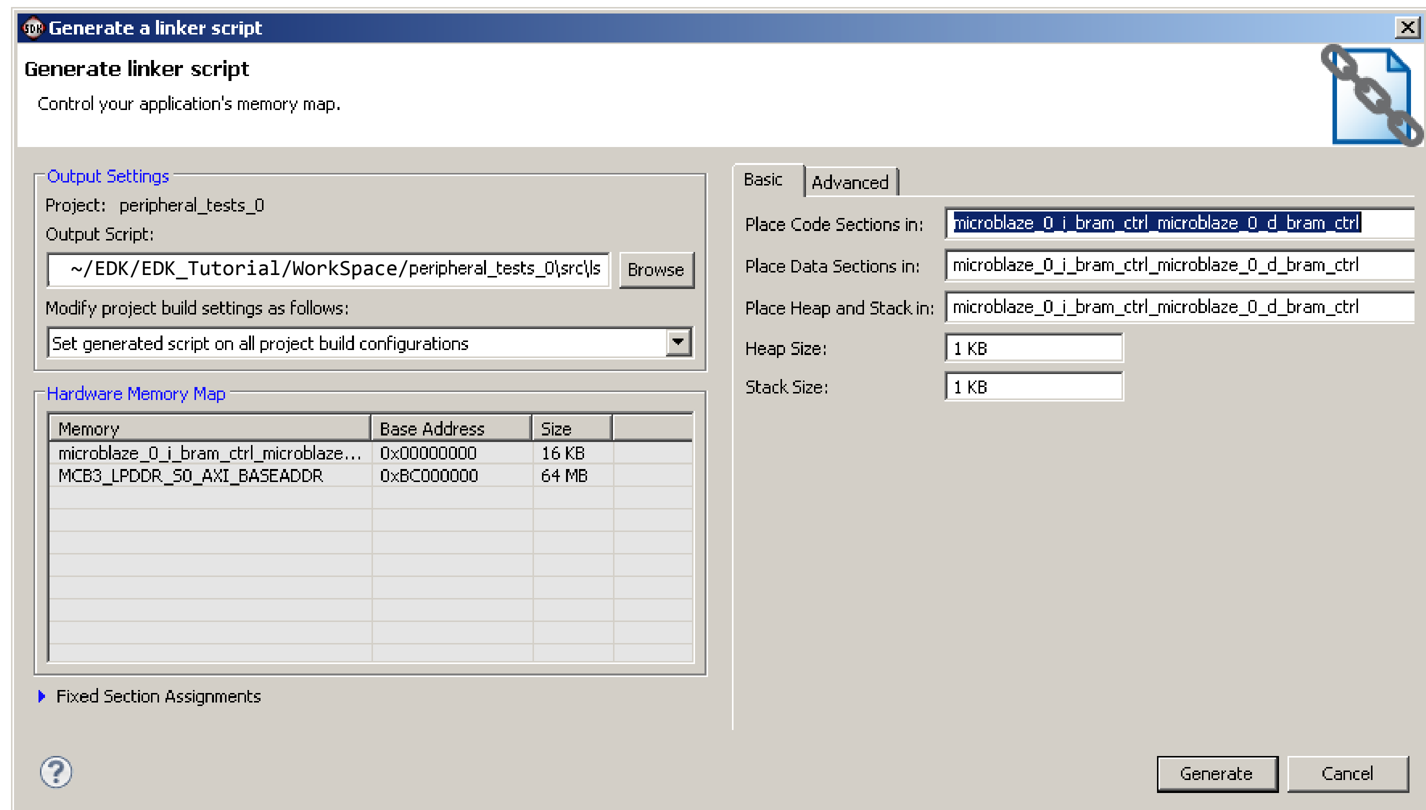
1.3 Compiling a Test Application Using SDK

8. The Project Explorer View in SDK contains 3 projects: the **hardware platform**, the **Board Support Package**, and **the C application**
9. The Board Support Package lists all the libraries and include files associated with the hardware project. The *microblaze_0\include* folder contains all the header files applicable for the current project.
10. The **peripheral_tests_0** project contains C source files to test each peripheral. Expand the project ***src*** folder to view the sources. The project is compiled automatically after being created.
11. Double click on the **testperiph.c** file to view the main application.
12. To view the project properties right click on the **peripheral_tests_0** project and select ***Properties***. **Expand C/C++ Build** and select ***Settings*** to view all the build options. Click ***Cancel*** to exit.



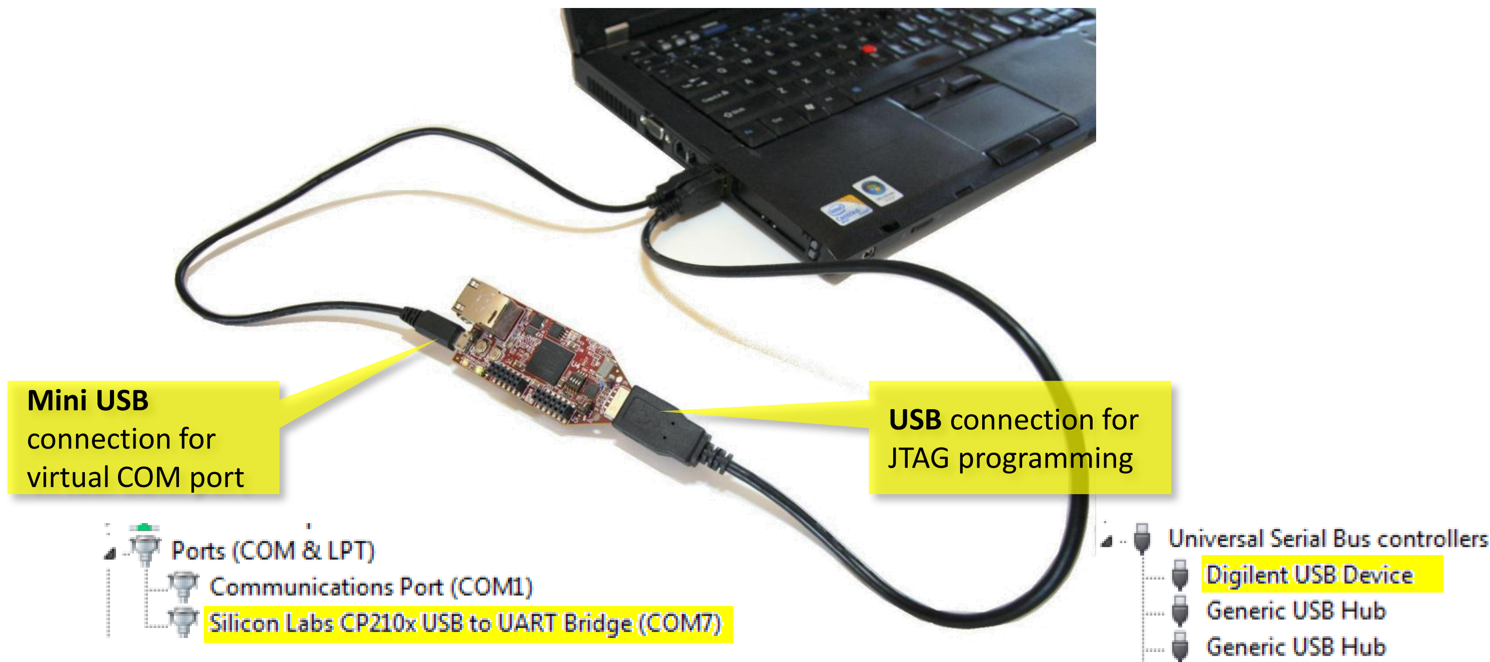
1.3 Compiling a Test Application Using SDK

13. The system contains internal BRAM memory as well as external DDR memory. We can select where the code will be physically located through a linker script.
14. Use the drop-down list to select the internal BRAM memory, ***microblaze_0_i_bram_ctrl_microblaze_0_d_bram_ctrl***, for all the code sections. Click Generate then ***Yes***.



1.4 Testing the Generated System

1. The LX9 MicroBoard uses two different USB interfaces for connecting to a PC for debugging and programming the device. Connect **both** USB cables:
   – **USB-to-UART** bridge for debug output (virtual COM port). The COM port was specified when installing the drivers for the Silicon Labs USB-to-UART Bridge (**Windows**: check Device Manager for COM port number, **Linux**: set correct **TTY**)
   – **USB-to-JTAG** bridge for programming the board (*.bit, *.elf file download, Flash programming)

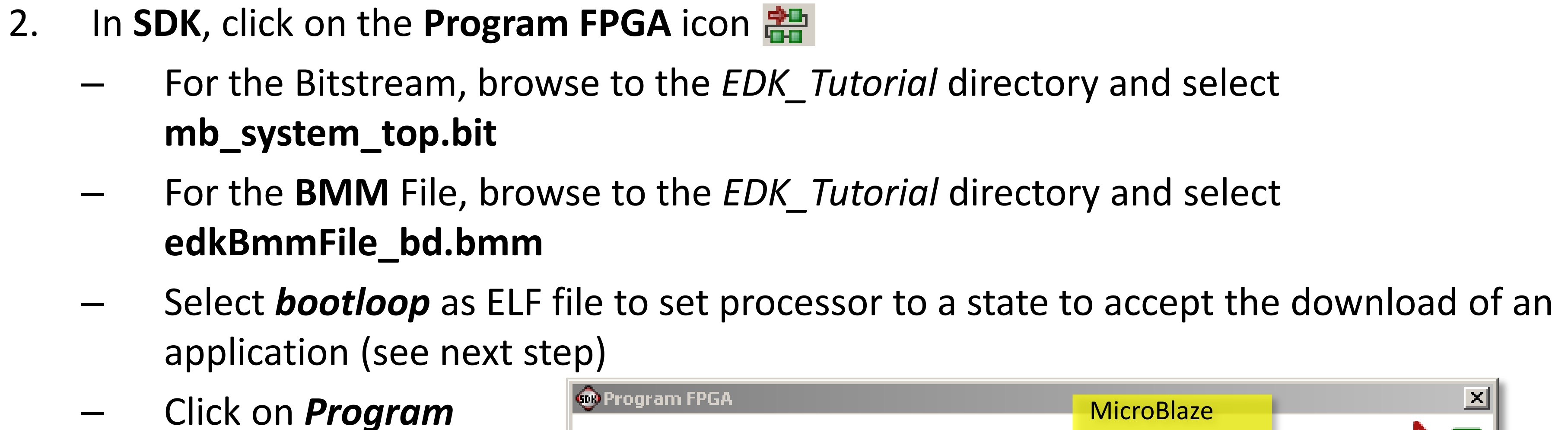


PHYSICS AT THE TERA SCALE Helmholtz Alliance

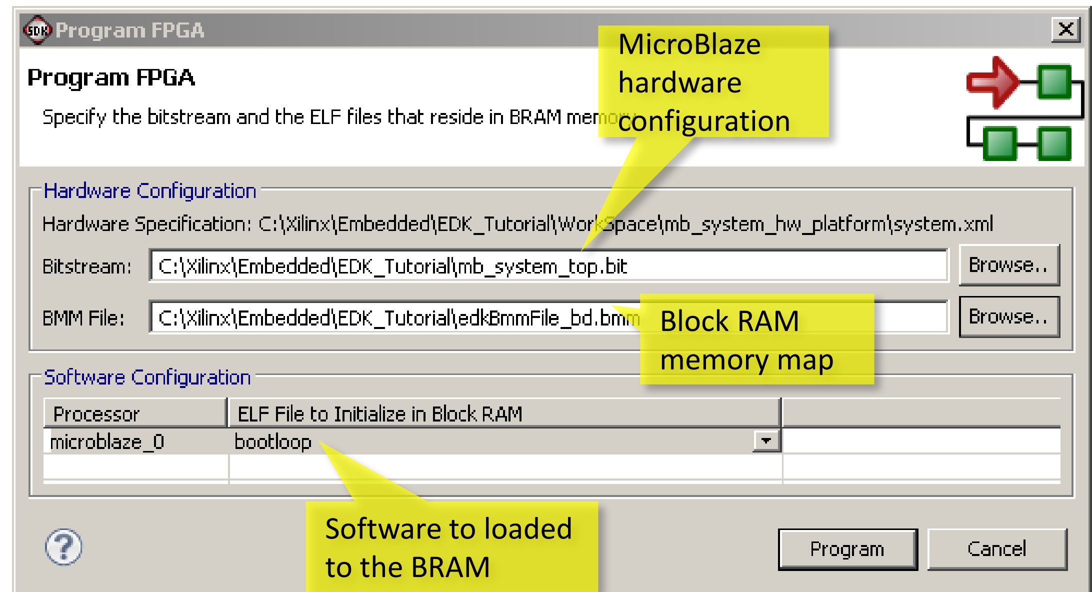
1.4 Testing the Generated System

At first, the hardware configuration is downloaded, and the processor reset to a state which allows the download of an application (bootloop).

2. In **SDK**, click on the **Program FPGA** icon



   – For the Bitstream, browse to the *EDK_Tutorial* directory and select **mb_system_top.bit**
   – For the **BMM** File, browse to the *EDK_Tutorial* directory and select **edkBmmFile_bd.bmm**
   – Select ***bootloop*** as ELF file to set processor to a state to accept the download of an application (see next step)
   – Click on ***Program***



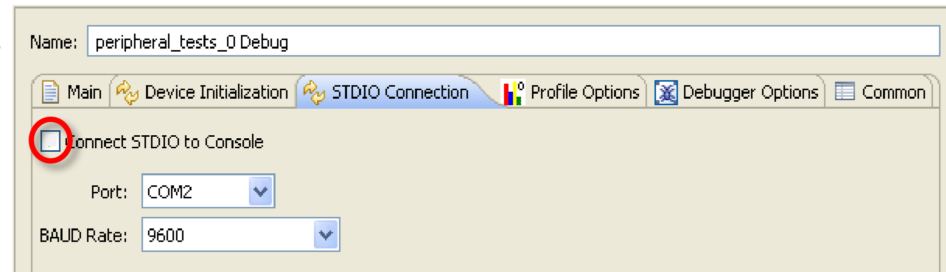
1.4 Testing the Generated System

Now the file *peripheral_test_0.elf* is selected to download and launch the application.

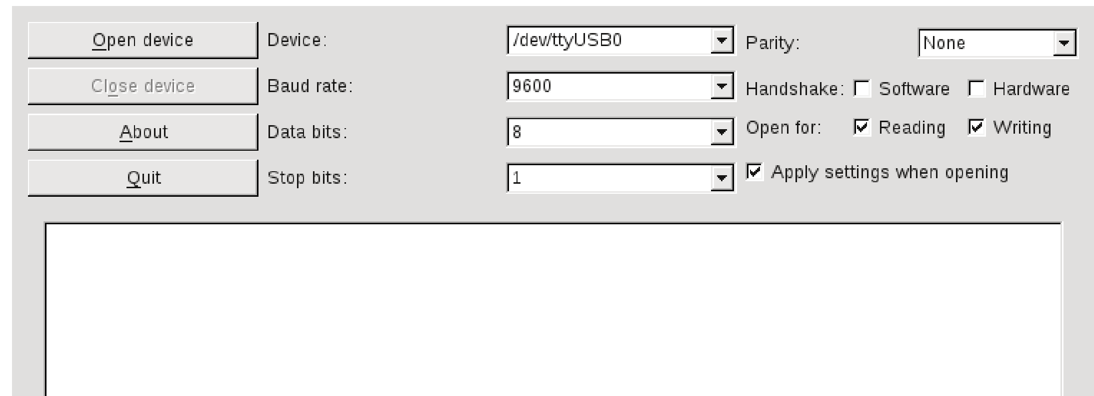
3. In the **SDK Project Explorer View**, right-click on the **peripheral_tests_0** project and select ***Run As -> Run Configurations***.
4. Select **Xilinx C/C++ ELF** and click on the ***New Launch Configuration*** icon.

1.4 Testing the Generated System

The **Run Configuration** contains settings to run the application on the target board. We will change the settings to disable STDIO inputs and outputs on the SDK console (stability issues). Instead we will be using **cutecom** as a terminal application.

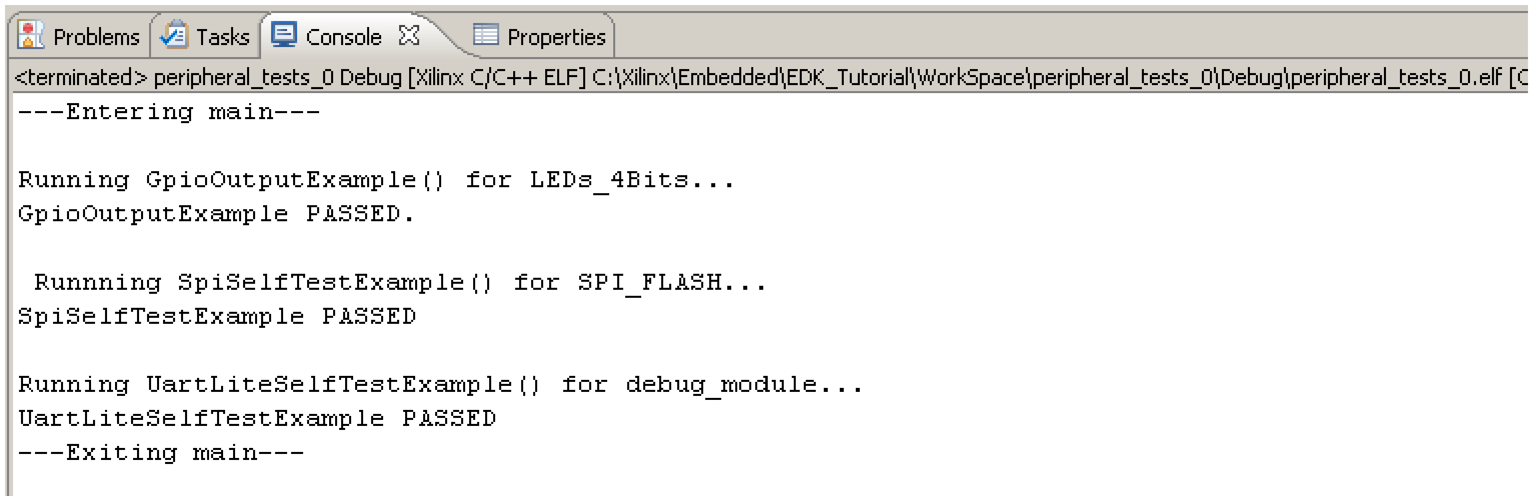


5. In the **SDK Run Configurations** window, select the **STDIO Connection tab**.
6. Uncheck the **Connect STDIO to Console** box.
7. Start **cutecom** with settings
   - Device: **/dev/ttyUSB0**
   - BAUD rate: 9600
   - Data bits: 8
   - Stop bits: 1
   - no handshake
8. Click ***Run*** in the **SDK** window.



1.4 Testing the Generated System

9. Verify that the different peripheral print statements appear on the **Console** output window and that all tests pass. Alternatively one could use any another terminal program to connect to the COM port.



10. Close the **SDK**. This is the end of tutorial 1.

Tutorial 2: Adding EDK IP

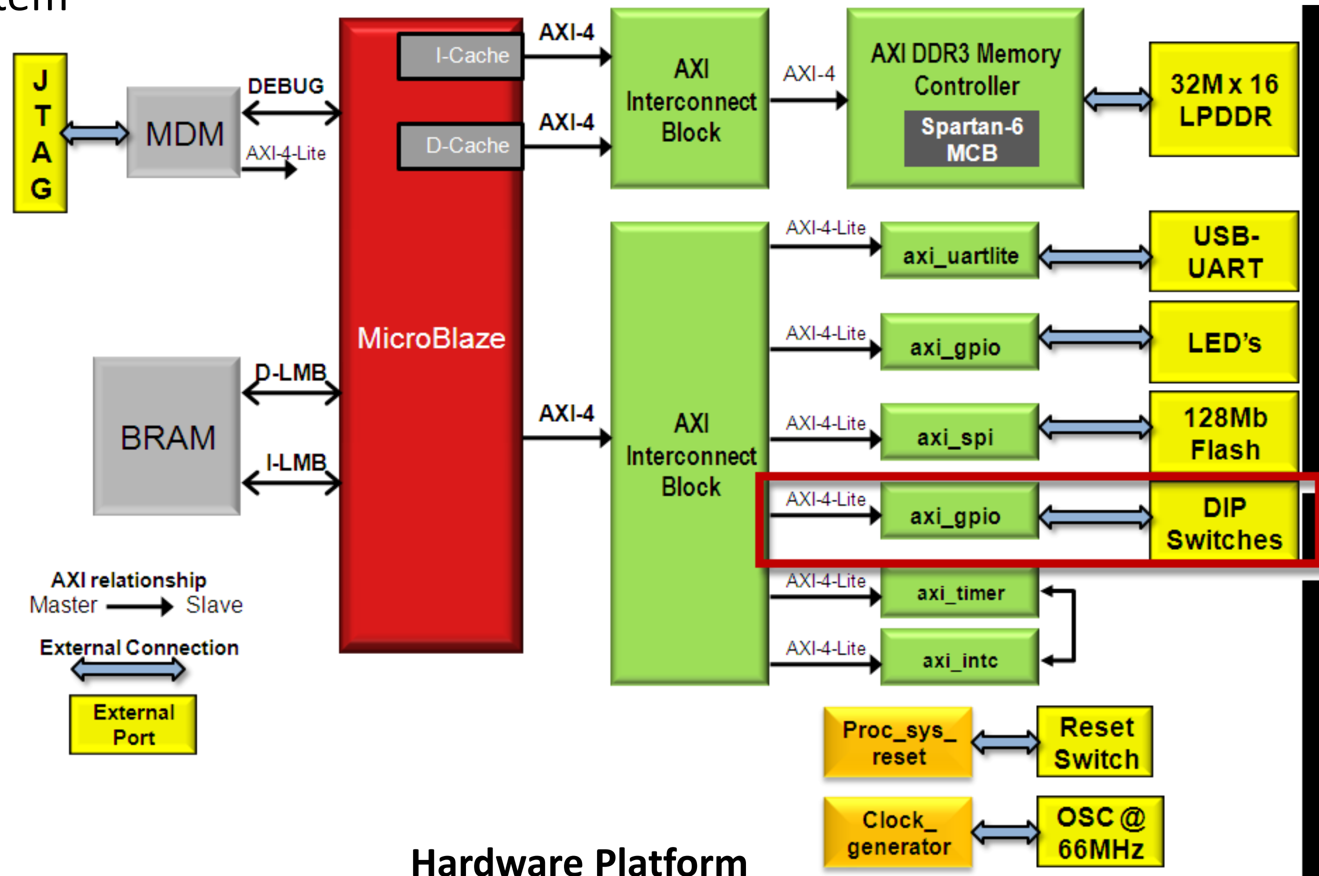
PHYSICS AT THE TERA SCALE
Helmholtz Alliance

Scope of the tutorial

2.1 Adding a new peripheral (➔ DIP switches)

2.2 Writing code for the peripheral

2.3 Testing the system



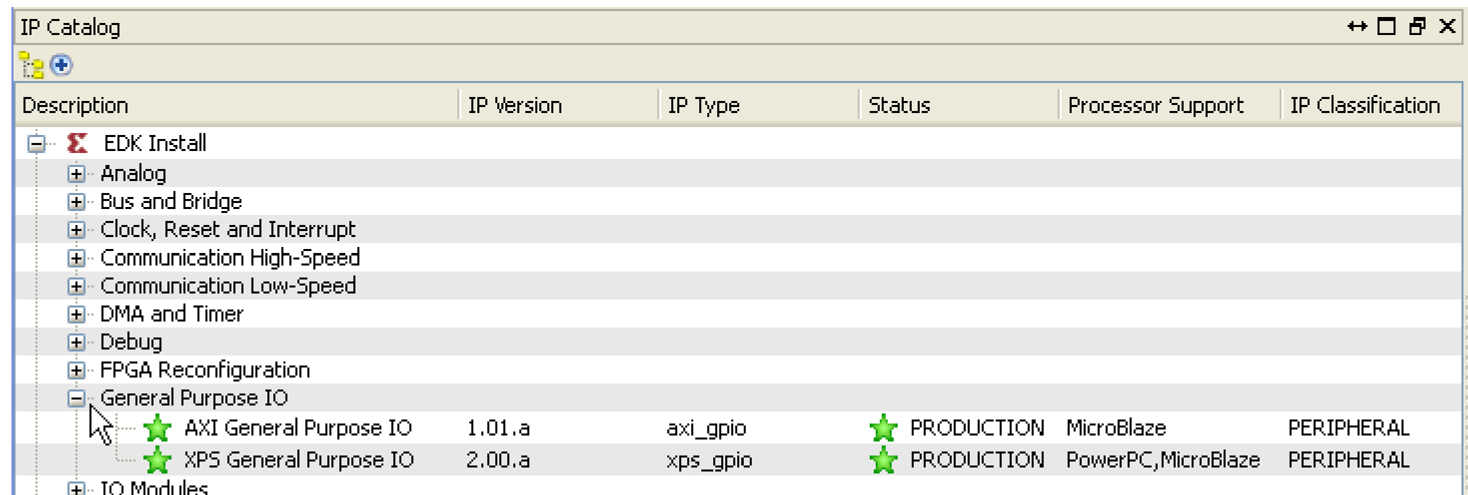
Hardware Platform

2.1 Adding a New Peripheral

PHYSICS AT THE TERA SCALE Helmholtz Alliance

We will use Xilinx Platform Studio (XPS) to add and connect a new peripheral to the existing system.

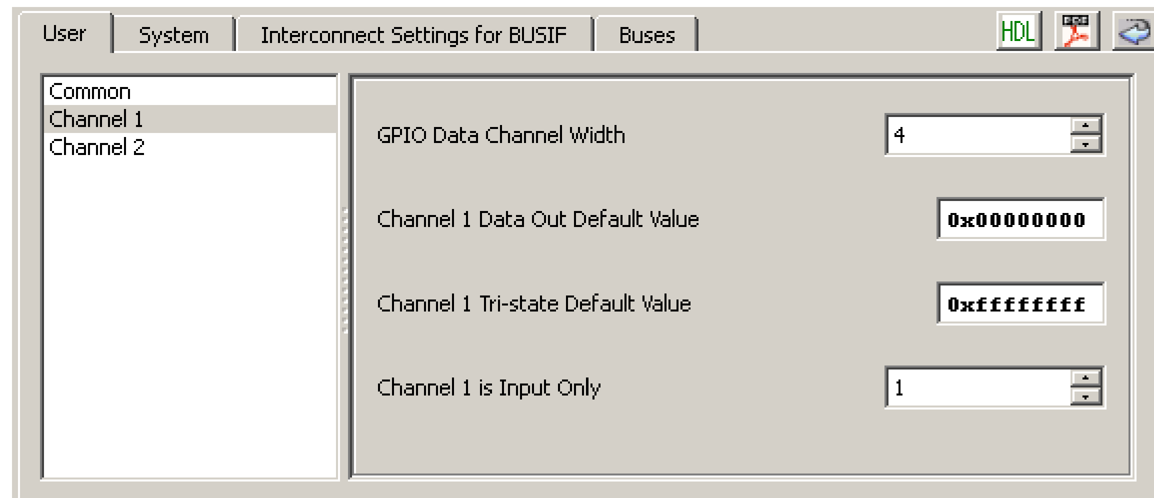
1. Start Project Navigator and open the **EDK_Tutorial** project from the first tutorial. If this tutorial is your starting point, you can start with the project found in **EDK132_Lab1_Solution**
2. Double-click on the **mb_system.xmp** module to open the system in XPS.
3. Select the **IP Catalog** tab in the project window. The IP catalog lists all the processor peripherals available with extended information. Expand the **General Purpose IO** option. The peripherals can be sorted by column field. Right click on the peripheral to view its datasheet or change log information.



2.1 Adding a New Peripheral

PHYSICS AT THE TERA SCALE Helmholtz Alliance

4. Select **axi_gpio** version 1.01.a on the list then drag and drop it to the **System Assembly View** window.
5. The peripheral configuration window will open to configure the peripheral. Select **Channel 1** and change the **GPIO Data Channel Width** from **32 to 4**. Change **Channel 1 is Input Only** from **0** to **1**.
6. Click **OK**.
7. When IP is added, a pop-up window will appear asking to automatically connect your new IP to the MicroBlaze core. Click **OK**.



2.1 Adding a New Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

8. Click on **axi_gpio_0** in the ***Name*** column of the **System Assembly** window. Rename the instance to **DIP_Switches**.

Bus Interfaces | Ports | Addresses

Name	Bus Name	IP Type	IP Version
axi4_0		axi_intercon...	1.02.a
axi4lite_0		axi_intercon...	1.02.a
microblaze_0_dlmb		lmb_v10	2.00.a
microblaze_0_ilmb		lmb_v10	2.00.a
microblaze_0		microblaze	8.10.a
microblaze_0_bram_block		bram_block	1.00.a
microblaze_0_d_bram_ctrl		lmb_bram_if...	3.00.a
microblaze_0_i_bram_ctrl		lmb_bram_if...	3.00.a
MCB3_LPDDR		axi_s6_ddrx	1.02.a
debug_module		mdm	2.00.b
microblaze_0_intc		axi_intc	1.01.a
DIP_Switches		axi_gpio	1.01.a
S_AXI	axi4lite_0		
LEDs_4Bits		axi_gpio	1.01.a

9. Click on the **Addresses** tab. The addresses view shows the address space for all the peripherals. The **Lock** box prevents the address for that peripheral from being changed when generating new addresses.

Change generated address parameters if necessary

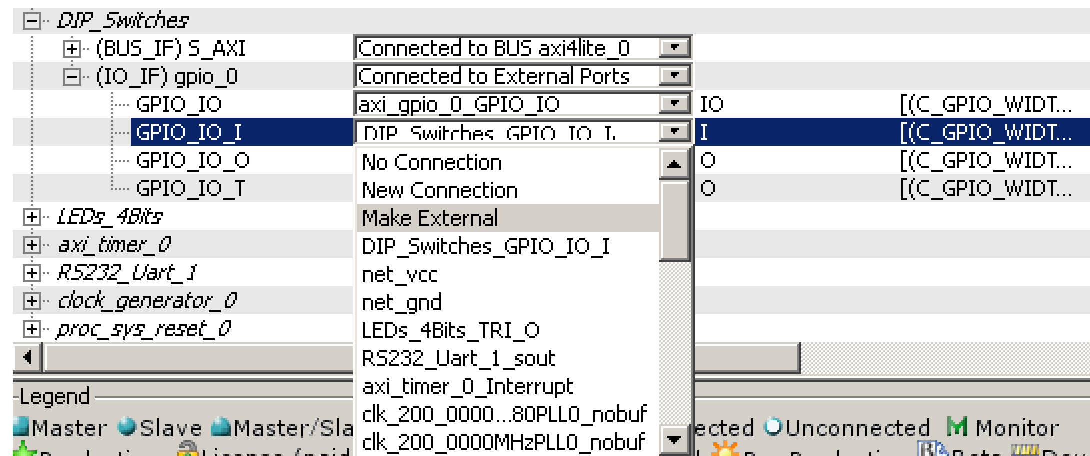
Alternatively: re-generate non-overlapping addresses for all peripherals

Bus Interfaces | Ports | Addresses — **Generate Address Button**

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_dlmb	☐
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	microblaze_0_ilmb	☐
LEDs_4Bits	C_BASEADDR	0x40000000	0x4000FFFF	64K	S_AXI	axi4lite_0	☐
DIP_Switches	C_BASEADDR	0x40020000	0x4002FFFF	64K	S_AXI	axi4lite_0	☐
RS232_Uart_1	C_BASEADDR	0x40600000	0x4060FFFF	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	☐
MCB3_LPDDR	C_S0_AXI_BASE...	0xBC000000	0xBFFFFFFF	64M	S0_AXI	axi4_0	☐

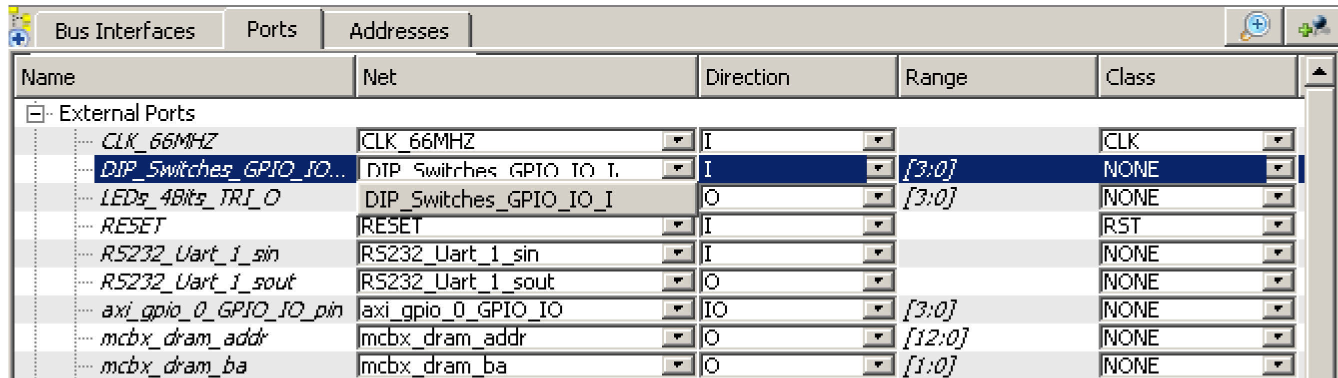
2.1 Adding a New Peripheral

10. Click on the **Ports** tab. The Ports view shows the internal connections between the peripherals as well as the external ports connections
11. Expand ***DIP_Switches*** from the list. It will show the connections available for the peripheral.
12. Expand the **(IO_IF) gpio_0** selection. Look at the GPIO datasheet for a description of each port. The datasheet can be found by right-clicking on ***DIP_Switches***.
13. Select **GPIO_IO_I** since the DIP switches are only inputs. Click on **No Connection** drop-down list in the **Net** column. Select **Make External** to add it the external ports list.
14. Select **GPIO_IO** net and set to **No Connection** as we are only using this port as an input.



2.1 Adding a New Peripheral

15. Expand the **External Ports** to view the new connection. The name of the new external port is **DIP_Switches_GPIO_IO_I_pin** with a range of **[3:0]**.



Bus Interfaces | Ports | Addresses

Name	Net	Direction	Range	Class
External Ports				
CLK_66MHZ	CLK_66MHZ	I		CLK
DIP_Switches_GPIO_IO...	DIP_Switches_GPIO_IO_I...	I	[3:0]	NONE
LEDs_4Bits_TRI_O	DIP_Switches_GPIO_IO_I	O	[3:0]	NONE
RESET	RESET	I		RST
RS232_Uart_1_sin	RS232_Uart_1_sin	I		NONE
RS232_Uart_1_sout	RS232_Uart_1_sout	O		NONE
axi_gpio_0_GPIO_IO_pin	axi_gpio_0_GPIO_IO	IO	[3:0]	NONE
mcbx_dram_addr	mcbx_dram_addr	O	[12:0]	NONE
mcbx_dram_ba	mcbx_dram_ba	O	[1:0]	NONE

We need to update the design information for SDK. In Tutorial 1, we exported our design to SDK from XPS, but since we are managing this embedded project from within Project Navigator, it will create the XML file and export the design to SDK.

16. Close **XPS**.

2.1 Adding a New Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

Since we've added a new port to MicroBlaze, it needs to be added to the HDL source. Additionally, we'll need to update the constraint file to add the pinout information for the DIP switches.

17. In Project Navigator, Open the top-level HDL file, **mb_system_top**, by double-clicking it. That will open the HDL source in editor.

18. Since the port goes externally to the FPGA it needs to be added to the **entity/module declaration**, add the following line as shown (add the highlighted, bold lines):

```
module mb_system_top
 (
  rzq,
   .
   .
   .
  SPI_FLASH_HOLDn,
  RESET,
  LEDs_4Bits_TRI_O,
  CLK_66MHZ,
  DIP_Switches_GPIO_IO_I_pin
 );
```

19. The same line needs to be added to the **component/signal declaration** of the mb_system. Add the following line as shown:

```
inout rzq;
   .
   .
   .
output SPI_FLASH_HOLDn;
input RESET;
output [3:0] LEDs_4Bits_TRI_O;
input CLK_66MHZ;
input [3:0] DIP_Switches_GPIO_IO_I_pin;
```

2.1 Adding a New Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

20. Finally, the instantiation of the **mb_system** needs to be updated as well. Add the following line as shown:

```
mb_system
  mb_system_i (
    .rzq ( rzq ),
    .mcbx_dram_we_n ( mcbx_dram_we_n ),
    .
    .
    .
    .SPI_FLASH_HOLDn ( SPI_FLASH_HOLDn ),
    .RESET ( RESET ),
    .LEDs_4Bits_TRI_O ( LEDs_4Bits_TRI_O ),
    .CLK_66MHZ ( CLK_66MHZ ),
    .DIP_Switches_GPIO_IO_I_pin (
DIP_Switches_GPIO_IO_I_pin)
  );
```

21. Save and close **mb_system_top.**
22. Select the **mb_system.ucf** file, expand **User Constraints** in the Processes window and double-click on **Edit Constraints (Text)**. In the UCF file add the lines:

```
NET DIP_Switches_GPIO_IO_I_pin[0] LOC = "B3" | IOSTANDARD = "LVCMOS33" | PULLDOWN;
NET DIP_Switches_GPIO_IO_I_pin[1] LOC = "A3" | IOSTANDARD = "LVCMOS33" | PULLDOWN;
NET DIP_Switches_GPIO_IO_I_pin[2] LOC = "B4" | IOSTANDARD = "LVCMOS33" | PULLDOWN;
NET DIP_Switches_GPIO_IO_I_pin[3] LOC = "A4" | IOSTANDARD = "LVCMOS33" | PULLDOWN;
```

23. Save and close the **UCF file**.
24. Expand the **mb_system_top** module in the **Hierarchy** window. Then select the embedded processor, **mb_system_i – mb_system(mb_system.xmp).**
25. Double-Click on **Export Hardware Design to SDK with Bitstream** to update the bit file with the new peripheral. This will take several minutes.

2.2 Writing Code for the New Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

To test the new peripheral we will create a new software application in **Platform Studio SDK** and use the GPIO device drivers.

1. When SDK opens select the **Workspace** from *EDK_Tutorial*.
2. The peripheral datasheets and address map can be found under the hardware platform. Expand the **mb_system_hw_platform** project and double-click on the **system.xml** file.
3. Open the **axi_gpio** datasheet to view the GPIO register map. The **GPIO_Data** Register is located at the base address of the peripheral, which is 0x40020000 for the DIP Switches. **NOTE**: You can open the datasheet by clicking on the hyperlink for the **axi_gpio** peripheral, however if no hyperlink is available, you can open the open the datasheet by expanding Standalone_BSP, then expanding BSP_Documentation and double-clicking on **gpio_v3_00_a.**

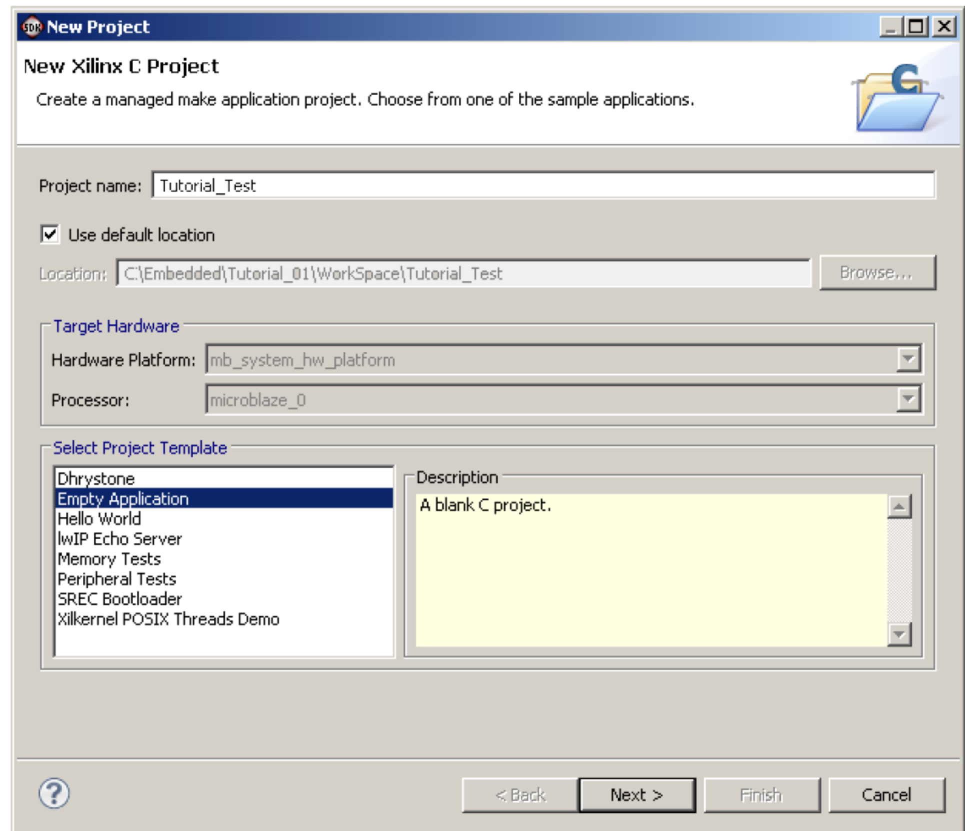
Table 4: **Registers**

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
C_BASEADDR + 0x00	GPIO_DATA	Read/Write	0x0	Channel 1 AXI GPIO Data Register.
C_BASEADDR + 0x04	GPIO_TRI	Read/Write	0x0	Channel 1 AXI GPIO Three-state Register.
C_BASEADDR + 0x08	GPIO2_DATA	Read/Write	0x0	Channel 2 AXI GPIO Data Register.
C_BASEADDR + 0x0C	GPIO2_TRI	Read/Write	0x0	Channel 2 AXI GPIO Three-state Register.

2.2 Writing Code for the New Peripheral

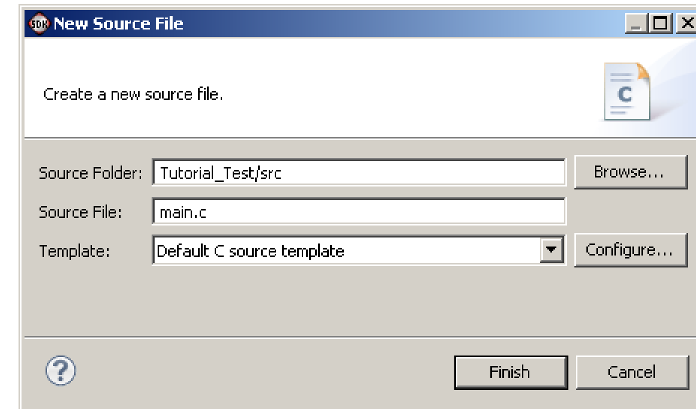
PHYSICS AT THE TERA SCALE Helmholtz Alliance

4. Go to **File > New > Xilinx C Project**.
5. Name the project **Tutorial_Test** and select Empty Application from the project templates. Click **Next**.
6. Select **Target an Existing Board Support Package** then click **Finish**.



2.2 Writing Code for the New Peripheral

7. We need to add a source file for the new empty C project. Select the ***Tutorial_Test\src*** folder and go to **File > New > Source File**. Enter ***main.c*** for the file name. Click **Finish**.



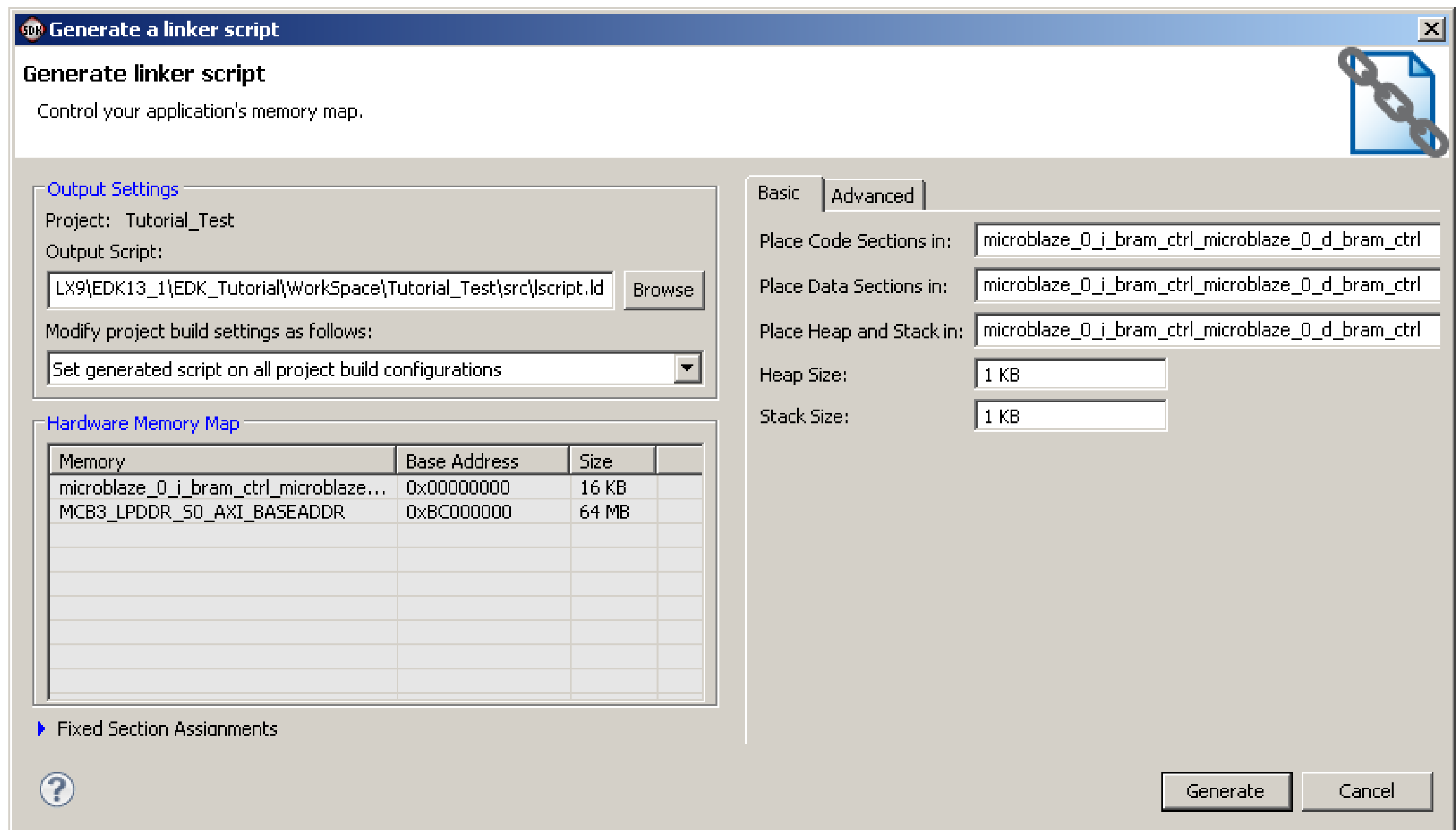
8. Inside ***main.c***, after the comments, add:

```
#include "xparameters.h"
#include "stdio.h"
#include "xbasic_types.h"
//==================================
int main (void) {
 print("-- Entering main() --\r\n");
return 0;
}
```

9. Save the ***main.c*** file. The application will be compiled when saved. The **Project** menu gives options to change the behavior for building the application.

2.2 Writing Code for the New Peripheral

10. Create a Linker Script for the new application. Right-click on the **Tutorial_Test** project and select **Generate Linker Script.** Select **microbalze_0_i_bram_ctlr_microblaze_0_d_bram_ctlr** for all the code sections to place them in the internal BRAMs. Click on **Generate**. Click **Yes** to overwrite the existing linker script.



2.2 Writing Code for the New Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

We will add code after the print statement to turn-on the LEDs when the DIP switches are asserted.

11. On the left side expand the **Standalone_BSP** project. The **BSP Documentation** section contains the documentation for the device drivers. The **microblaze_0** folder contains the header files, compiled libraries, and sources for the Board Support Package.
12. Expand **microblaze_0** then expand the **include** directory. Double click on the **xparameters.h** file to view the hardware parameters for the system. Using the macros will isolate the software from the actual hardware.
13. Inside the expanded **include** directory for **microblaze_0** are all the driver header files for the different peripherals. The **_l.h** denotes a low level driver. Double-click on **xgpio_l.h** to view the GPIO low level functions.
14. Click on **XGpio_ReadReg** in the **Outline** window to view the format to read the GPIO registers. We will also use the function **XGpio_WriteReg** to write to the LEDs. The Base Address for the device can be found in the **xparameters.h** file. The Data Register has an offset of 0x00.

2.2 Writing Code for the New Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

peripheral_test_0
code from tutorial 1

Standalone_BSP
Board support package documentation

microblaze_0
header files, compiled libraries, BSP sources

microblaze_0 -> include
- *xparameters.h*: macros for HW / SW isolation
- *xgpio_l.h*: low level functions (driver files *_l.h)

Tutorial_Test
code from this tutorial

C/C++ - Tutorial_Test/main.c - Xilinx SDK

File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help

Project Explorer

- mb_system_hw_platform
- peripheral_tests_0
- Standalone_BSP
  - BSP Documentation
    - bram_v3_00_a
    - gpio_v3_00_a
    - s6_ddrx_v1_00_a
    - spi_v3_02_a
    - standalone_v3_01_a
    - uartlite_v2_00_a
  - microblaze_0
    - code
    - include
    - lib
    - libsrc
  - libgen.log
  - libgen.options
  - Makefile
  - system.mss
- Tutorial_Test
  - Binaries
  - Includes
  - Debug
  - src
  - main.c

system.xml | system.mss | main.c

```c
/*
 * main.c
 *
 *  Created on: Sep 22, 2011
 *      Author: 028700
 */

#include "xparameters.h"
#include "stdio.h"
#include "xbasic_types.h"
#include "xgpio_l.h"
//====================================================

u32 DIP_Read;
int main (void) {

    print("-- Entering main() --\r\n");

    while (1) {
    DIP_Read = XGpio_ReadReg(XPAR_DIP_SWITCHES_BASEADDR, 0);
    XGpio_WriteReg(XPAR_LEDS_4BITS_BASEADDR, 0, DIP_Read);
    }

    return 0;
}
```

Problems | Tasks | Console | Properties

C-Build [Tutorial_Test]

```
Invoking: Xilinx ELF Check
elfcheck Tutorial_Test.elf -hw ../../mb_system_hw_platform/system.xml -pe
microblaze_0  |tee "Tutorial_Test.elf.elfcheck"
elfcheck
Xilinx EDK 13.3 Build EDK_O.76xd
Copyright (c) 1995-2011 Xilinx, Inc.  All rights reserved.

Command Line: elfcheck -hw ../../mb_system_hw_platform/system.xml -pe
microblaze_0 Tutorial_Test.elf
```

/Tutorial_Test/main.c

2.2 Writing Code for the New Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

15. Add code to ***main.c*** to read DIP switches and write the settings to the LED output

system.xml | system.mss | main.c

```c
/*
 * main.c
 *
 *  Created on: Sep 22, 2011
 *      Author: 028700
 */

#include "xparameters.h"
#include "stdio.h"
#include "xbasic_types.h"
#include "xgpio_l.h"
//====================================================

u32 DIP_Read;
int main (void) {

    print("-- Entering main() --\r\n");

    while (1) {
    DIP_Read = XGpio_ReadReg(XPAR_DIP_SWITCHES_BASEADDR, 0);
    XGpio_WriteReg(XPAR_LEDS_4BITS_BASEADDR, 0, DIP_Read);
    }

    return 0;
}
```

include GPIO low level driver *xgpio_l.h*

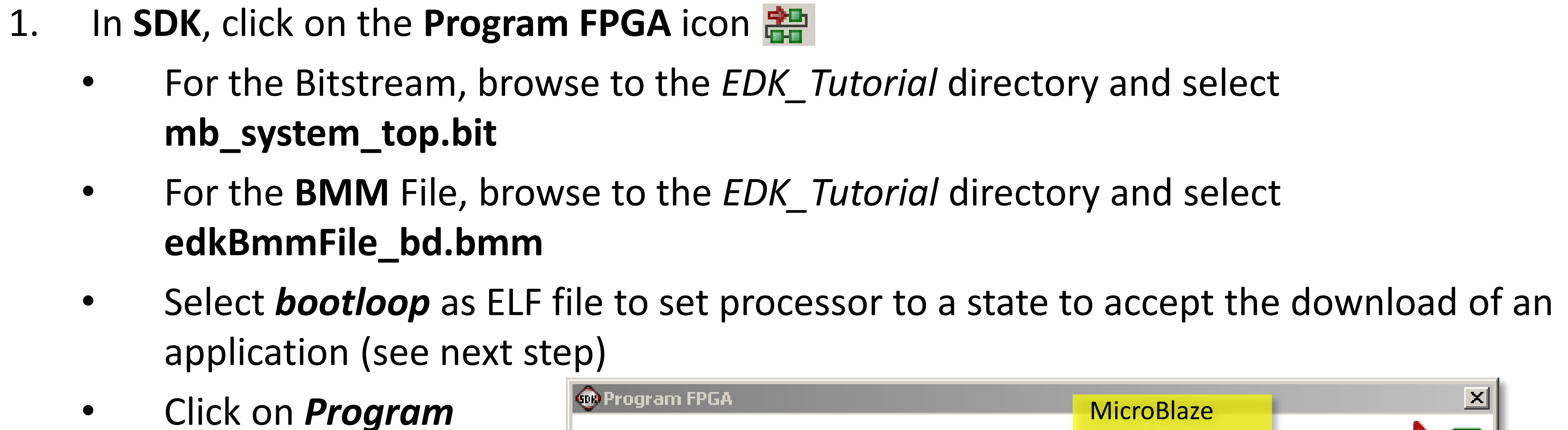
declare global variable *DIP_Read*

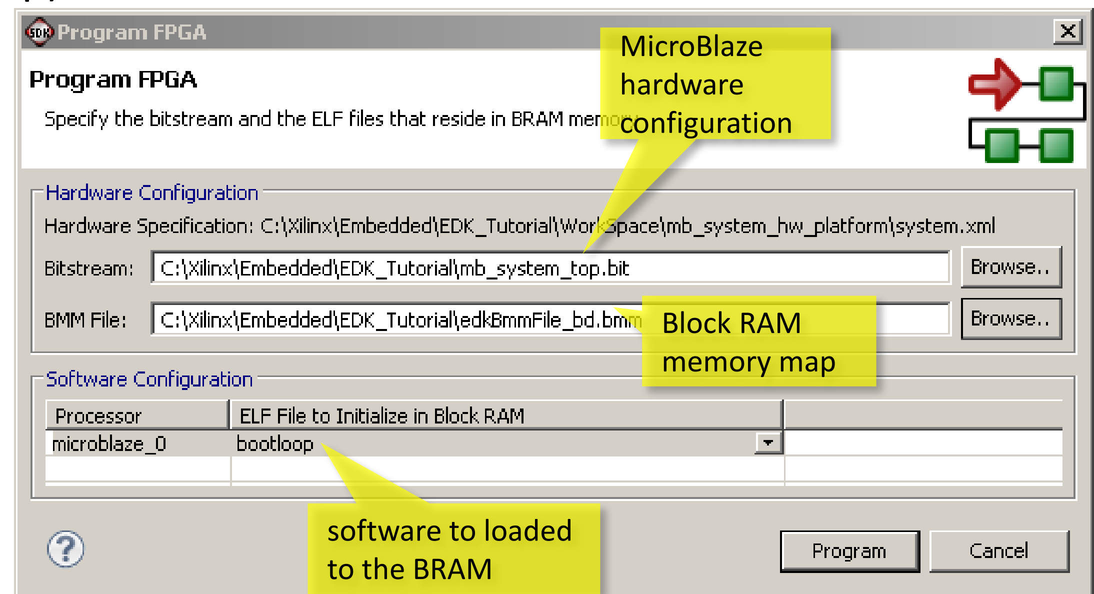
add code to read DIP switches and set LED accordingly

16. Save and close the file. The application will be compiled automatically.
17. To view the changes made to ***main.c***, right-click on ***main.c*** in the **Project Explorer** window and select **Compare > With Local History…** Click **OK** when finished.

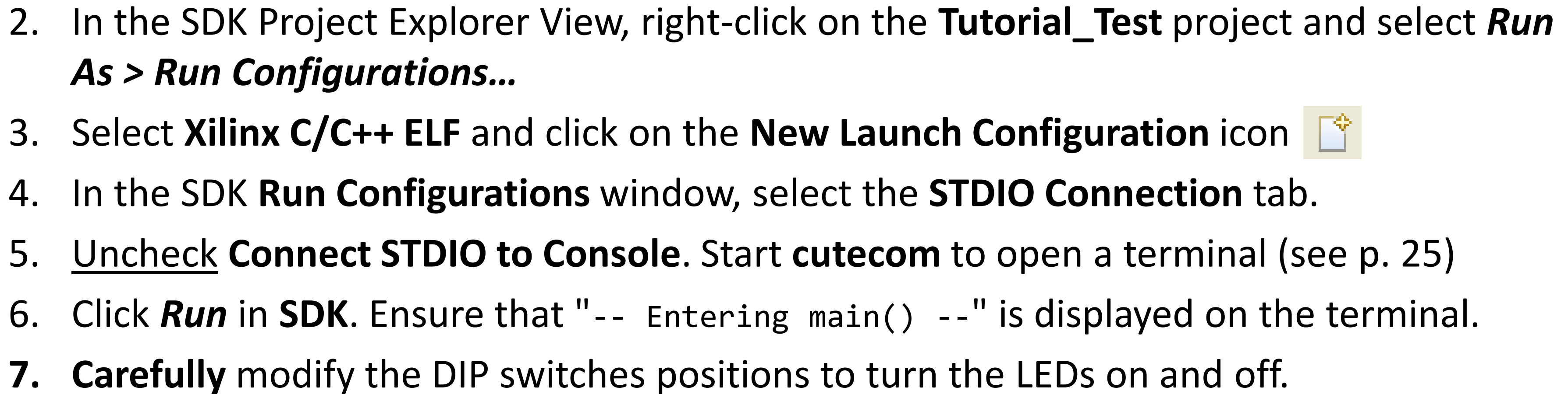
2.3 Testing the Generated System

At first, the hardware configuration is downloaded, and the processor is reset to a state which allows the download of an application (bootloop).

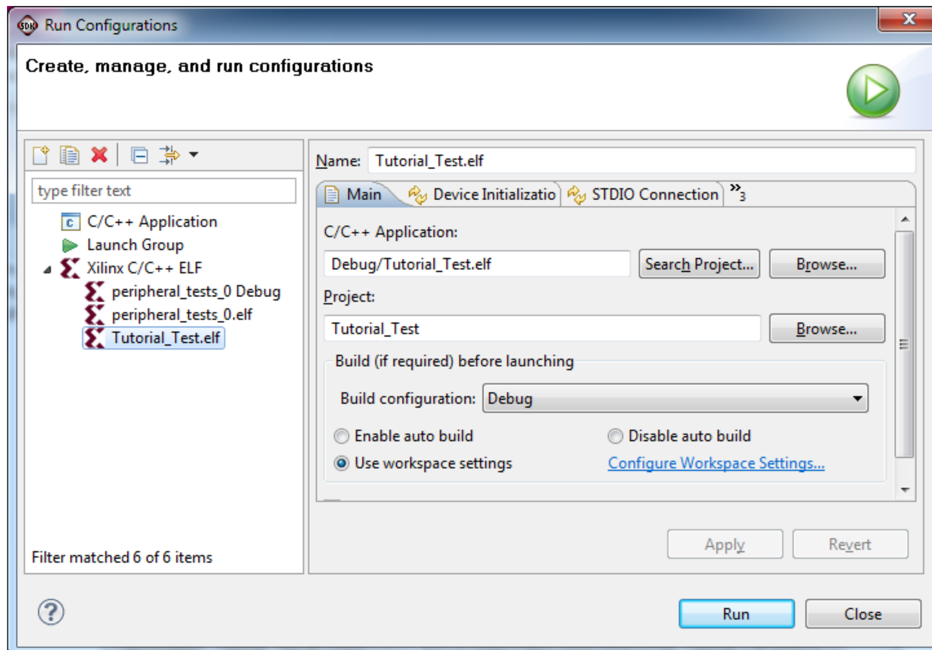
1. In **SDK**, click on the **Program FPGA** icon 
   - For the Bitstream, browse to the *EDK_Tutorial* directory and select **mb_system_top.bit**
   - For the **BMM** File, browse to the *EDK_Tutorial* directory and select **edkBmmFile_bd.bmm**
   - Select ***bootloop*** as ELF file to set processor to a state to accept the download of an application (see next step)
   - Click on ***Program***

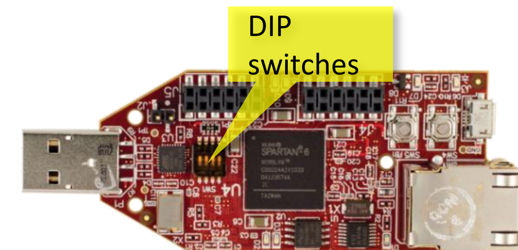


2.3 Testing the Generated System



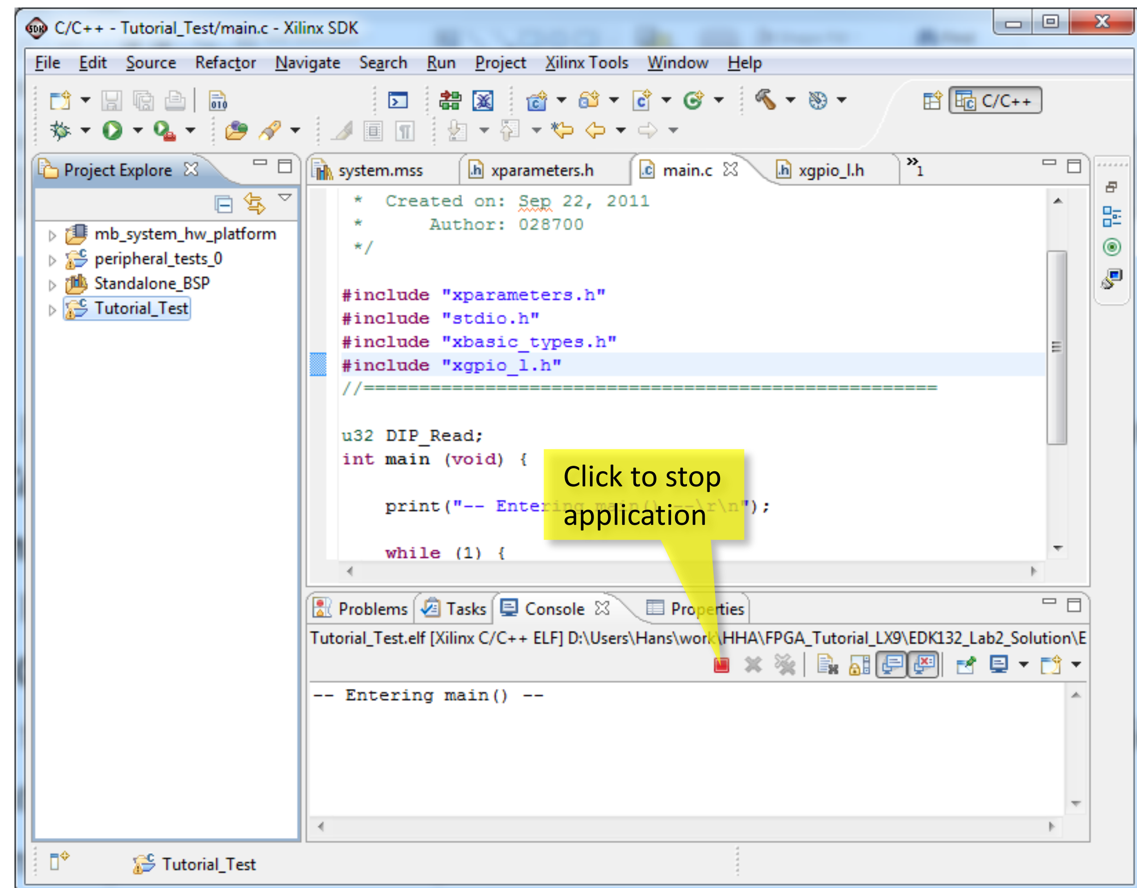
2. In the SDK Project Explorer View, right-click on the **Tutorial_Test** project and select ***Run As > Run Configurations…***
3. Select **Xilinx C/C++ ELF** and click on the **New Launch Configuration** icon
4. In the SDK **Run Configurations** window, select the **STDIO Connection** tab.
5. Uncheck **Connect STDIO to Console**. Start **cutecom** to open a terminal (see p. 25)
6. Click ***Run*** in **SDK**. Ensure that "`-- Entering main() --`" is displayed on the terminal.
7. **Carefully** modify the DIP switches positions to turn the LEDs on and off.





2.3 Testing the Generated System

8. When finished **Stop/Terminate** the application by pressing the red Stop/Terminate button in the Console window
9. Close **SDK**. This is the end of Tutorial 2.



Tutorial 3: Adding Custom IP

Scope of the tutorial

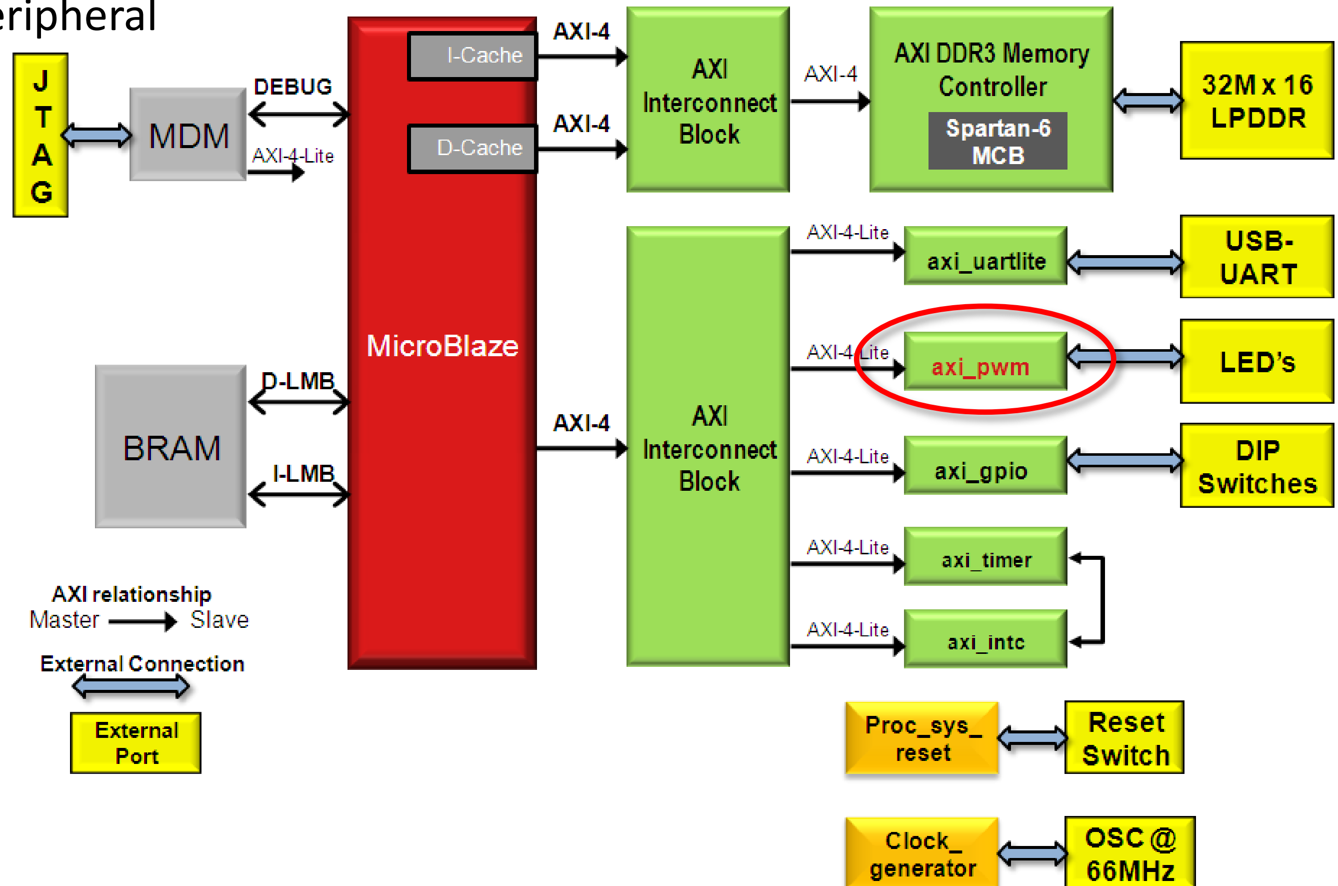
3.1 Creating a custom core (CIP wizard)

3.2 Customizing the peripheral

3.3 Adding the IP

3.4 Writing code

3.5 Testing the system



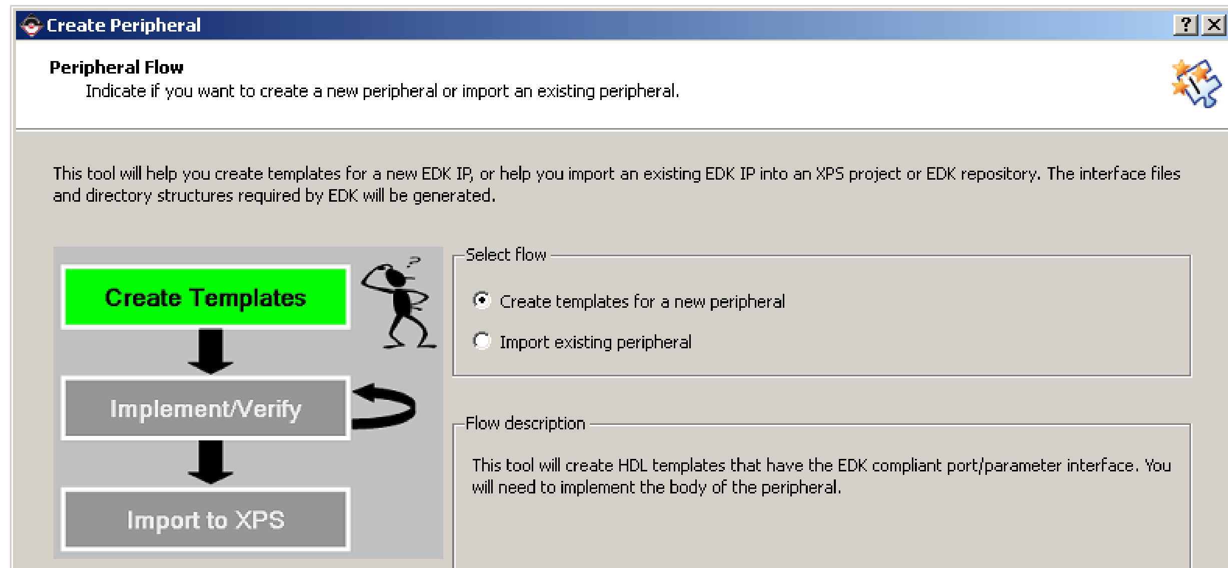
Hardware Platform

3.1 Creating a Custom IP

We will use the **Create/Import Peripheral (CIP)** wizard in **XPS** to create a new custom IP for the existing system. The custom IP will consist of a Pulse Width Modulator (PWM) controlled using a software mapped register.

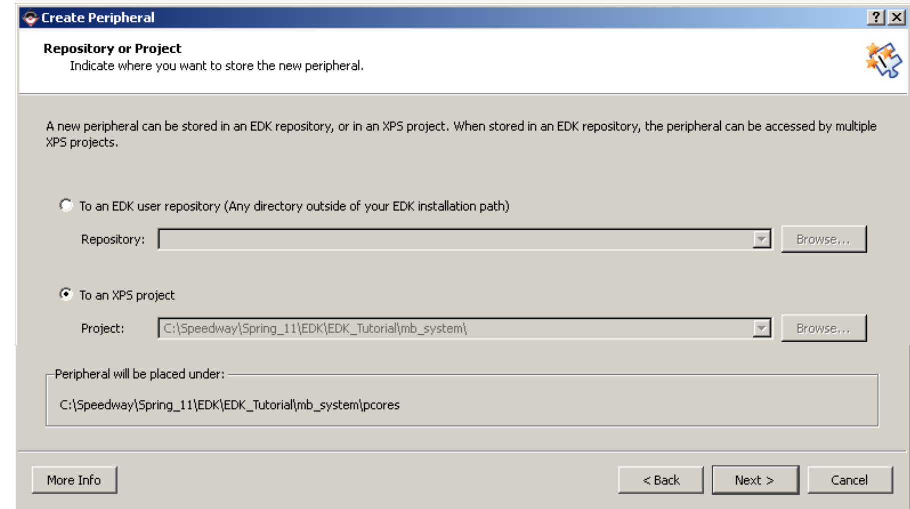
If this tutorial is your starting point, you can use the **EDK132_Lab2_Solution** to start with

1. Start **ISE Project Navigator** and open the **EDK_Tutorial** project.
2. Double-click on the **mb_system.xmp** module to open the system in XPS.
3. Go to **Hardware > Create or Import Peripheral…** Click on **Next**.
4. Make sure that **Create templates for a new peripheral** is selected then click **Next**.

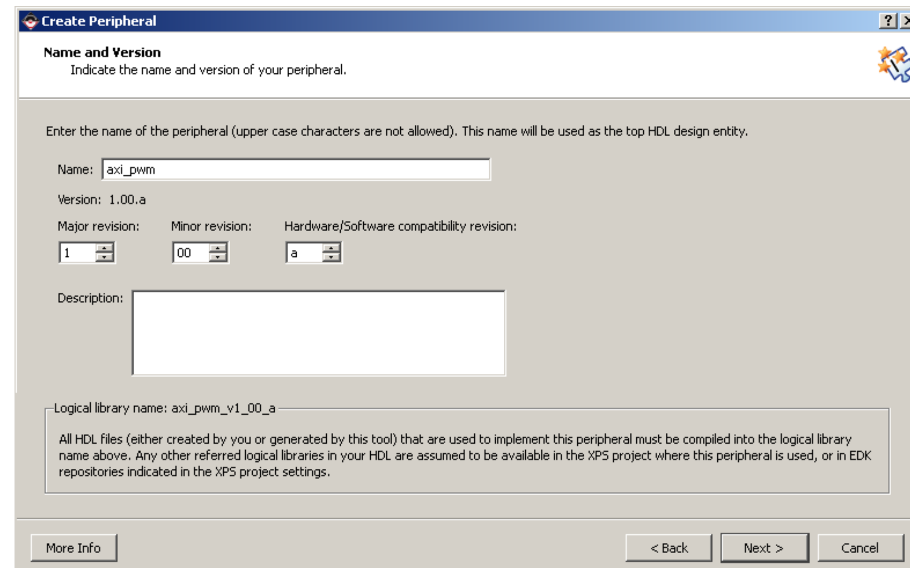


3.1 Creating a Custom IP

5. Select **To an XPS project**.
   Click on **Next**.



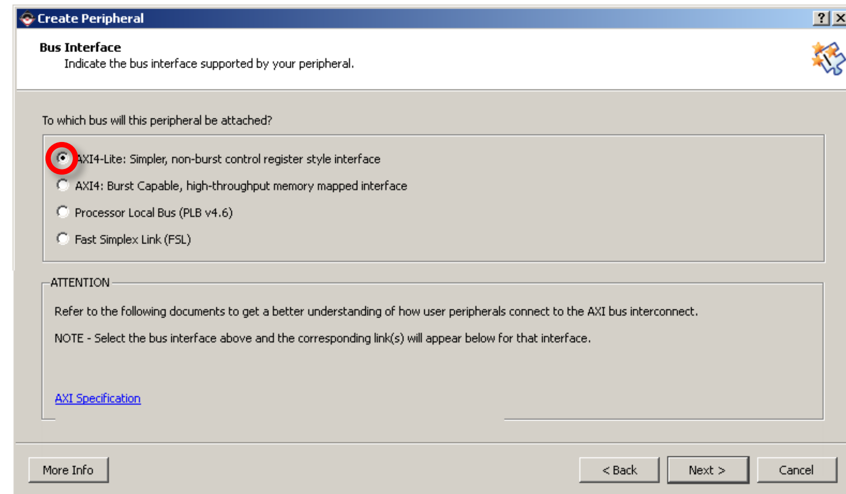
6. Enter the name for the new peripheral, **axi_pwm**, and then click **Next**.



PHYSICS AT THE TERA SCALE
Helmholtz Alliance

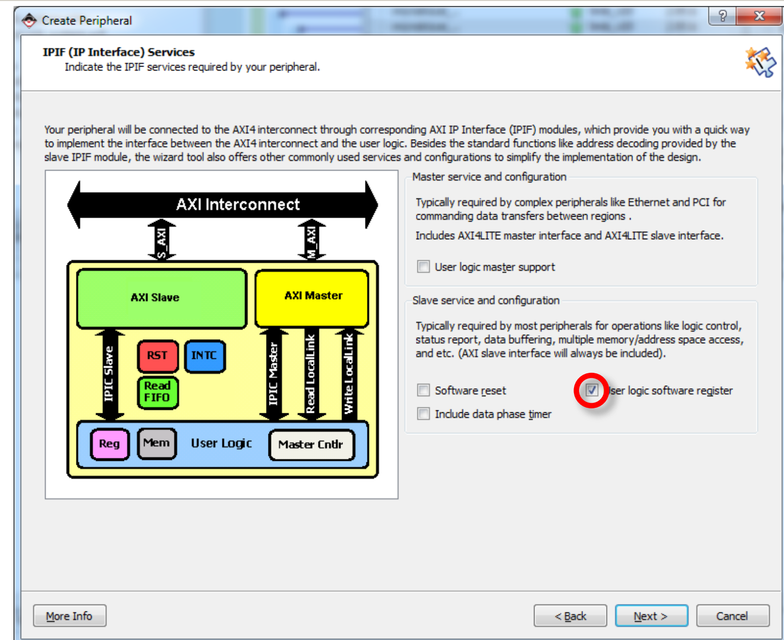
3.1 Creating a Custom IP

7. Select **AXI4-Lite**: Simpler, non-burst control register style interface. Click **Next**.



The IPIF is a module isolating the user interface from the bus. In addition to facilitating bus attachment, the IPIF provides additional optional services. The services include software registers, user address ranges, FIFOs, software reset, interrupt support and bus-master access. You can click on the **More Info** button, then select **AXI Bus Interface > IPIF Features for AXI** to see a description of each feature.

8. We will be using software registers to control the peripheral. Select **User logic software register**. We'll choose how many registers in the next screen. **Unselect** all other choices. Click **Next**.



3.1 Creating a Custom IP

9. We will use one 32-bit wide register to communicate with the PWM hardware. Though only twelve bits will be used to select the pulse duty cycle. Select **1** for the number of registers. Click on **Next**.

Create Peripheral

User S/W Register
Configure the software accessible registers in your peripheral.

The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on your design. An example logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference.

Bus2IP_RdReq
Bus2IP_WrReq
Bus2IP_RdCE
Bus2IP_WrCE
Bus2IP_Data
IP2Bus_Data
IP2Bus_RdAck
IP2Bus_WrAck
IP2Bus_Error
Reg 0
Reg 1
Reg 2
Reg 3
Reg n
User Logic

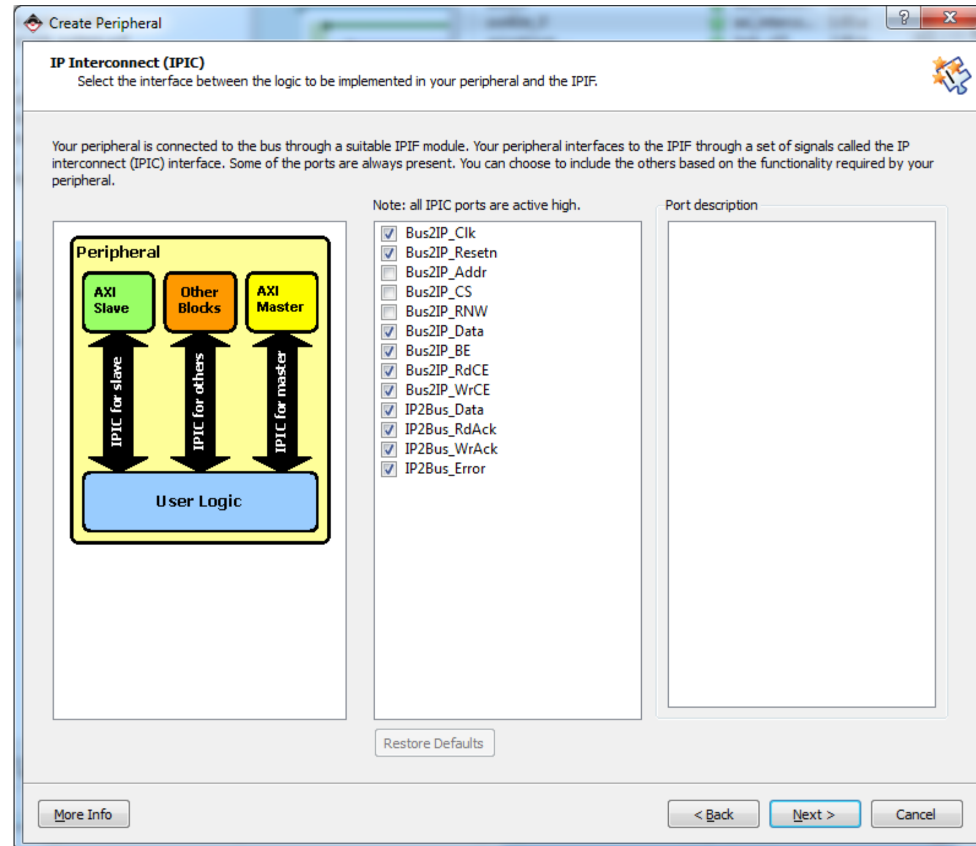
User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIC slave signals to read/write the registers.

Number of software accessible registers: 1 (1 to 32)

More Info | < Back | Next > | Cancel

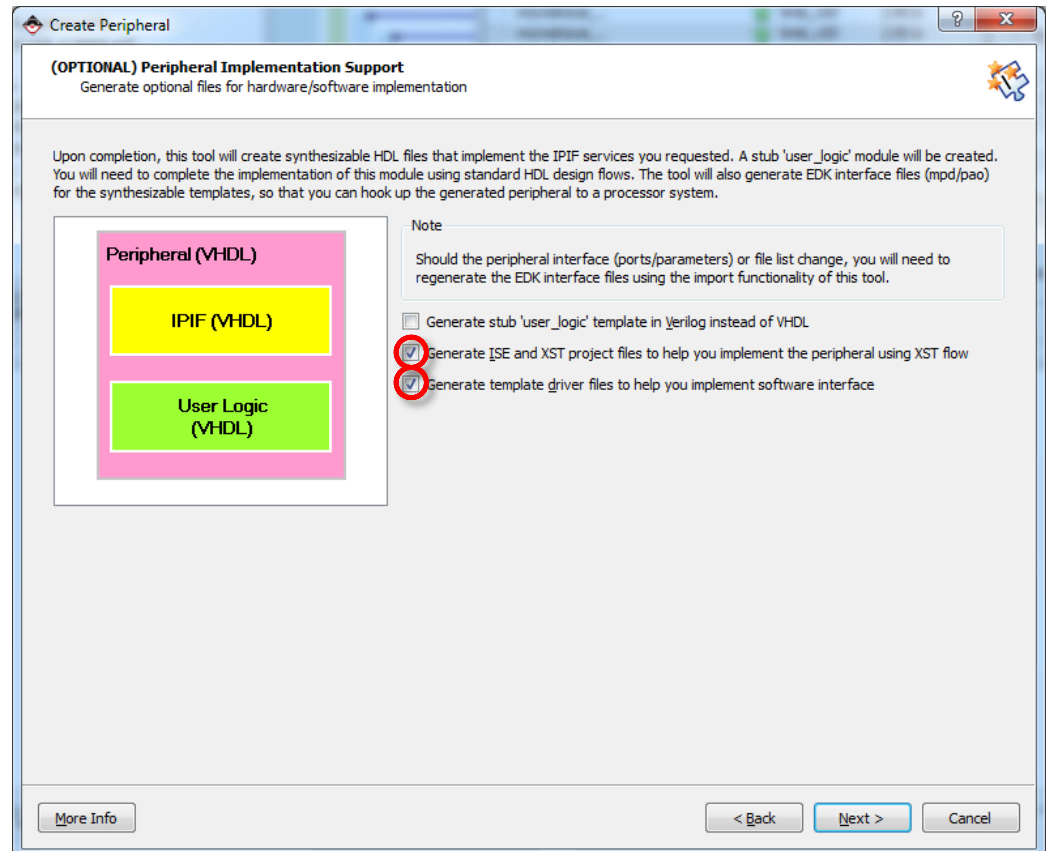
3.1 Creating a Custom IP

10. The IP Interconnect (IPIC) uses a set of signals between the user logic and the AXI bus. We will use the default signals already selected. Click on **Next**.



3.1 Creating a Custom IP

11. Bus Functional Models can be generated to accelerate the IP verification. This tutorial does not cover the BFM simulation of the peripheral. Click on **Next**.
12. The wizard can also generate custom drivers for the peripheral and an ISE project. We will be using **XPS** and writing our own code to test the peripheral. This is also where you can select a Verilog template versus the default, VHDL. Leave it as VHDL for this tutorial. Click on **Next**.
13. Click on **Finish** to create the peripheral.

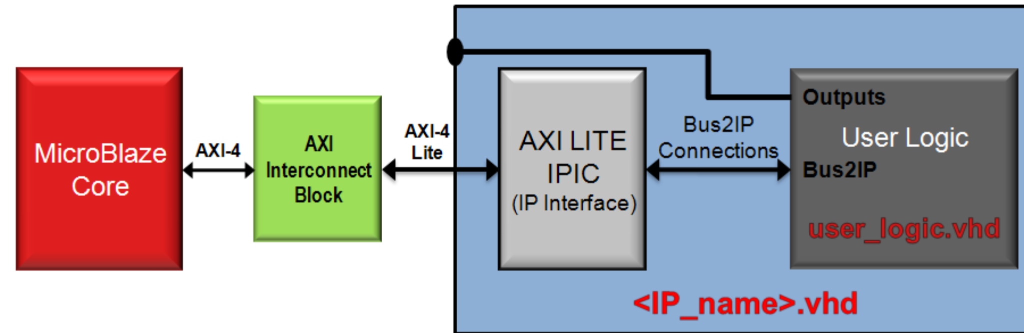


3.2 Customizing the New Custom Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

The new peripheral has been created in */mb_system/pcores/axi_pwm_V1_00_a.* In the folder */hdl/vhdl* two files have been generated:

- *axi_pwm.vhdl*
- *user_logic.vhdl*



Both files will be edited now to implement the PWM functionality. The top level entity *axi_pwm.vhdl* instantiates the user logic (*USER_LOGIC*) and the proxy to the AXI interface (*AXI_LITE_IPIF*) which has been created according to the **CIP wizard** dialog.

1. Within **ISE** open */mb_system/pcores/axi_pwm_V1_00_a/hdl/vhdl/axi_pwm.vhdl* and add the code lines as described on the next slide. Alternatively you can browse to **EDK132_Lab3_Solution** and copy the VHDL file to the appropriate path in your EDK_Tutorial tree.

3.2 Customizing the New Custom Peripheral

PHYSICS AT THE TERA SCALE Helmholtz Alliance

2. Add the external port to the ports declaration of *axi_pwm.vhdl:*

```
-- ADD USER PORTS BELOW THIS LINE ------------------
PWM_Out : out std_logic;
-- ADD USER PORTS ABOVE THIS LINE ------------------
```

3. Add the port to the USER_LOGIC_I component instantiation

```
-- MAP USER PORTS BELOW THIS LINE ------------------
PWM_Out => PWM_Out,
-- MAP USER PORTS ABOVE THIS LINE ------------------
```

4. Save and close the file

5. Open the *user_logic.vhdl* file and implement the PWM output which will be controlled by the software register.

6. Add the ports declaration

```
-- ADD USER PORTS BELOW THIS LINE ------------------
PWM_Out : out std_logic;
-- ADD USER PORTS ABOVE THIS LINE ------------------
```

7. Add user signals declaration

```
-- USER signal declarations added here, as needed for user logic
signal duty_cycle : std_logic_vector (11 downto 0);
signal fcount : std_logic_vector (11 downto 0);
```

3.2 Customizing the New Custom Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

8. Add the user HDL code after the begin statement. The *user_logic* template already contains code to read and write the register.

```
-- USER logic implementation added here
-- Duty cycle is controlled by the software controlled register
duty_cycle <= slv_reg0(11 downto 0);
-- 12-bit rollover counter
counter : process (Bus2IP_Clk)
begin
     if (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
          if Bus2IP_Resetn = '0' then
               fcount <= (others => '0');
          else fcount <= fcount + 1;
          end if;
     end if;
end process counter;
-- Enable the output for the duty cycle selected
PWM_Out <= '1' when (fcount < duty_cycle) else '0';
```

9. Save and close the file

3.2 Customizing the New Custom Peripheral

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

The new external port needs to be added to the definition file for the peripheral in order to be used in **XPS**.

10. Open the *axi_pwm_v1_00_a\data* directory and open the file *axi_pwm_v2_1_0.mpd*.
11. Add the **PWM_Out** port

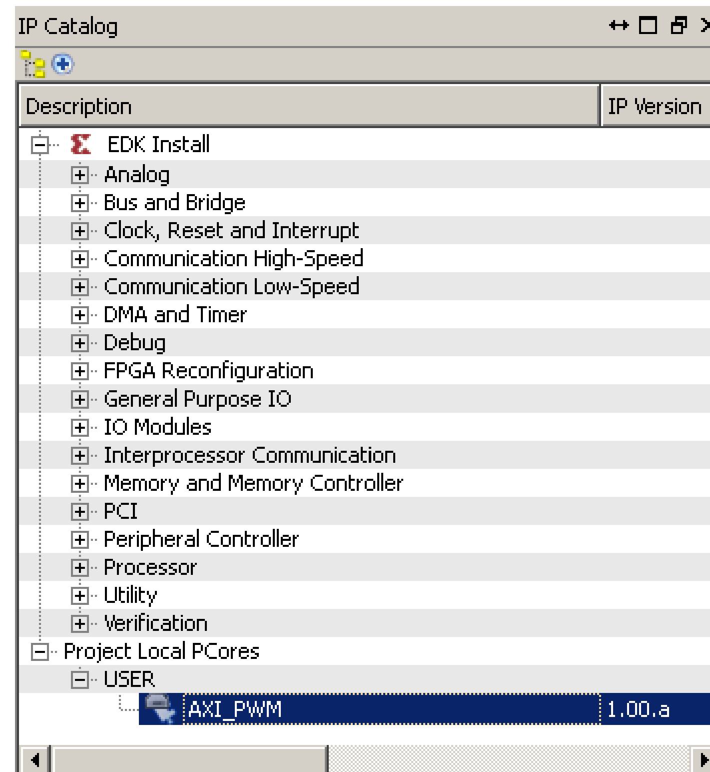
```
## Ports
PORT PWM_Out = "", DIR = O
```

12. Save and close the file.
13. In **XPS**, rescan the user IP directories. **Project > Rescan User Repositories.**
14. The new core will now be available.

3.3 Adding the Custom IP to the System

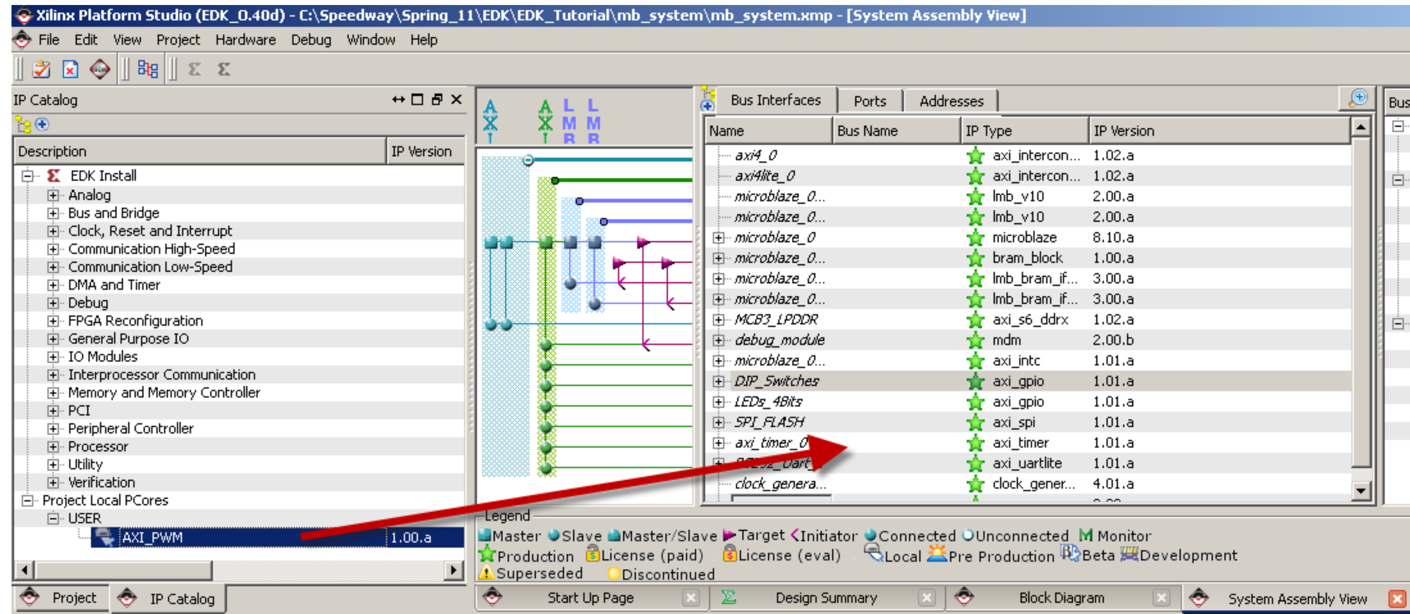
We will add and connect the new custom IP to the existing system following the same instructions as in the previous lab. We will remove the GPIO peripheral for the LEDs and connect the PWM peripheral to the LEDs.

1. In **XPS**, click on the **IP Catalog** tab in the **Project Information Area**.
2. Expand the **Project Local Pcores/USER** list to view the custom IP.



3.3 Adding the Custom IP to the System

3. Select **AXI_PWM** then drag and drop it to the **System Assembly View** window. Click OK.



4. Click OK to connect this IP to MicroBlaze

5. Click on the **Addresses** tab to view the address range for the new IP

Bus Interfaces | Ports | Addresses

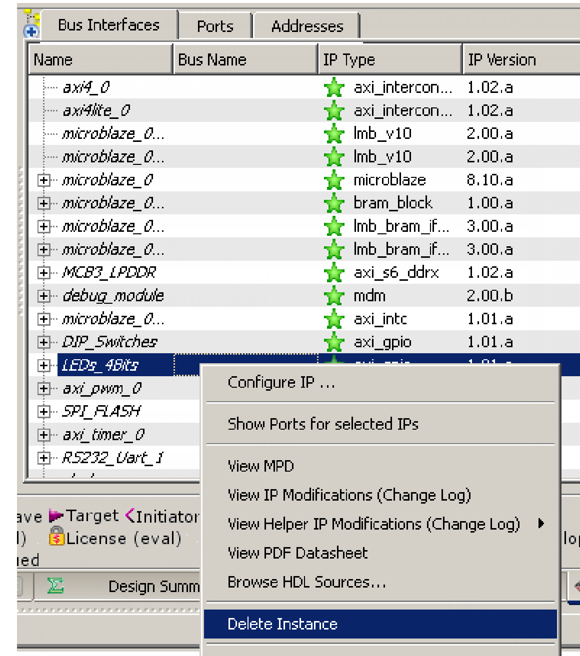
Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	microblaze_0_dlmb	☐
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	microblaze_0_ilmb	☐
LEDs_4Bits	C_BASEADDR	0x40000000	0x4000FFFF	64K	S_AXI	axi4lite_0	☐
DIP_Switches	C_BASEADDR	0x40020000	0x4002FFFF	64K	S_AXI	axi4lite_0	☐
RS232_Uart_1	C_BASEADDR	0x40600000	0x4060FFFF	64K	S_AXI	axi4lite_0	☐
SPI_FLASH	C_BASEADDR	0x40A00000	0x40A0FFFF	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	☐
axi_pwm_0	C_BASEADDR	0x7EE00000	0x7EE0FFFF	64K	S_AXI	axi4lite_0	☐
MCB3_LPDDR	C_S0_AXI_BASE...	0xC0000000	0xC3FFFFFF	64M	S0_AXI	axi4_0	☐

IP Type : mdm

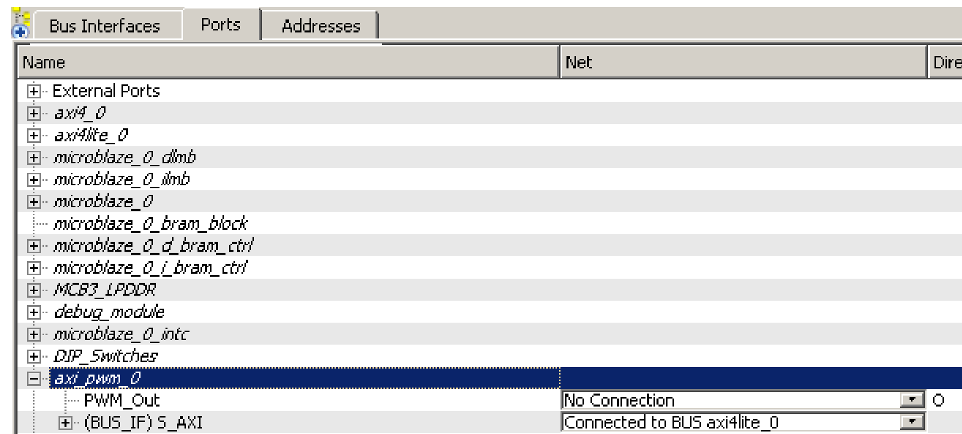
3.3 Adding the Custom IP to the System

6. Delete the GPIO peripheral instance for the LEDs. In the **System Assemble View**, right-click on the **LEDS_4Bit** instance and select **Delete Instance**. In the pop-up select **Delete instance but do not remove the nets**. Click OK.

That is very important! If you delete the nets you cannot connect your new peripheral block.

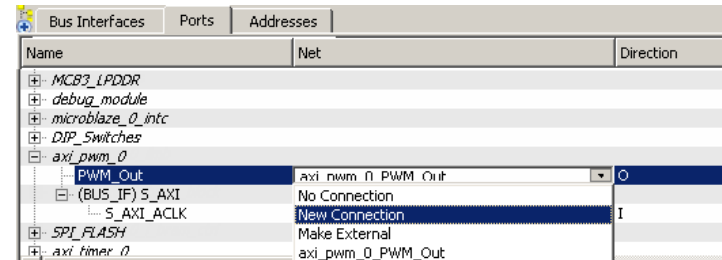


7. Click on the **Ports** tab. Expand **axi_pwm_0** from the list. It will show the connections available for the peripheral.



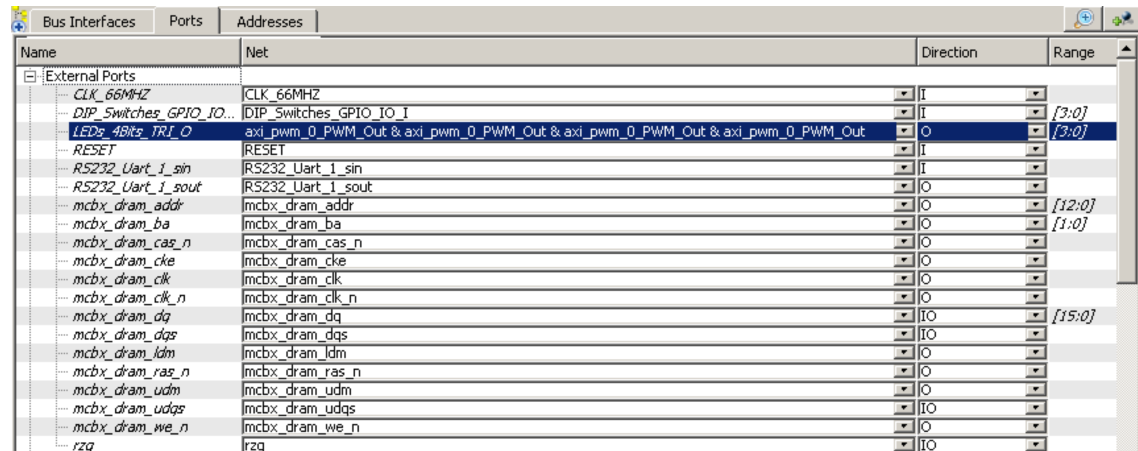
3.3 Adding the Custom IP to the System

8. For **PWM_Out** click on the **Net** column and select **New Connection** from the drop-down list. The new net name will be **axi_pwm_0_PWM_Out**. If it does not appear, select it from the pull down.



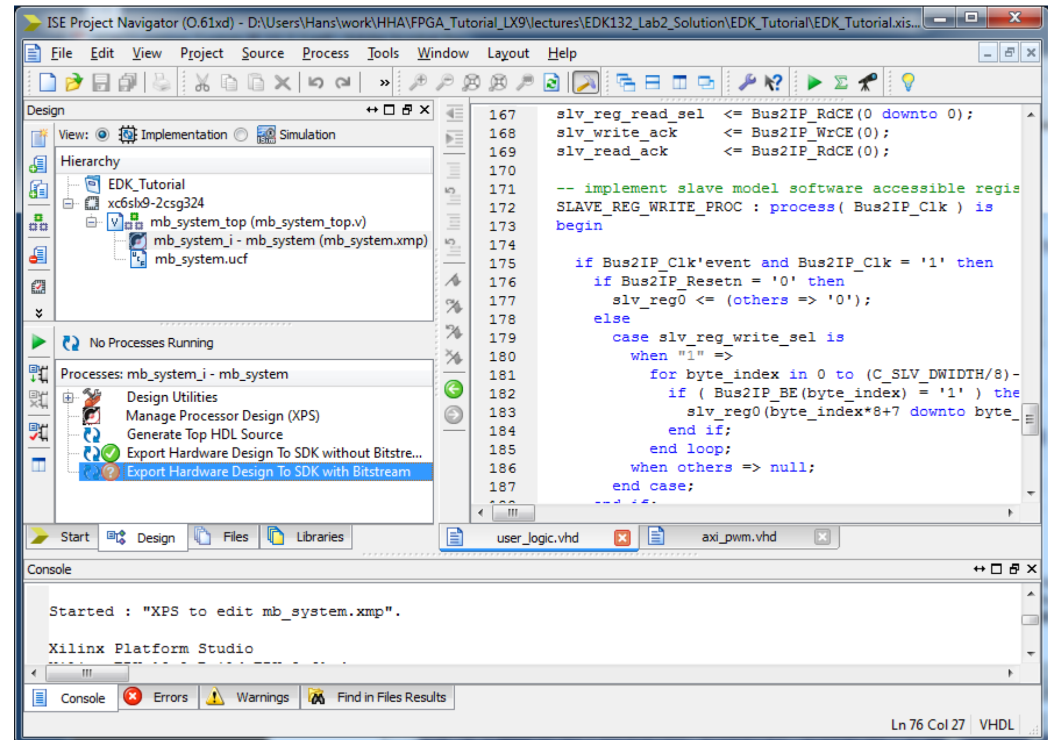
9. Expand the **External Ports** connections. For the **LEDs_4Bits_TRI_O**, replace the current **Net** entry with **axi_pwm_0_PWM_Out** & **axi_pwm_0_PWM_Out** & **axi_pwm_0_PWM_Out** & **axi_pwm_0_PWM_Out**. This concatenation will drive all 4 LED's with the same brightness.

10. Close **XPS**.



3.3 Adding the Custom IP to the System

11. In **Project Navigator**, expand the **mb_system_top** module in the **Hierarchy** window. Then select the embedded processor, **mb_system_i – mb_system(mb_system.xmp)**.



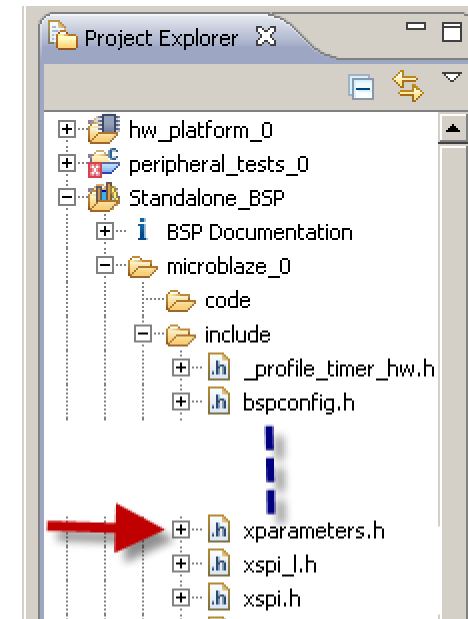
12. Double-Click on **Export Hardware Design to SDK with Bitstream** to update the bit file with the new peripheral. This may take several minutes.

3.4 Writing Code for the Custom Peripheral

To test the new peripheral we will add code to the **Tutorial_Test** project created with Platform Studio SDK. We will make use of the DIP switches to control the pulse duty cycle. The 4 DIP switches will be used to select the 4 most significant bits of the duty cycle. We will lose some precision but still be able to test the peripheral.

1. Start Xilinx **SDK** and select the Workspace from *EDK_Tutorial.*
   NOTE: The applications will not compile since we removed the GPIO peripheral which was used for the LEDs. The duty cycle is controlled with a software register. From the address map defined in the *user_logic* code, the register is located at BaseAddress + 0x0.
2. Expand the **Standalone_BSP** project then **microblaze_0** in the **Project Explorer** window. Expand the include directory. Double click on the *xparameters.h* file to view the driver parameters for the custom peripheral:
3. Scroll down to view the address for custom peripheral.

```
/* Definitions for peripheral AXI_PWM_0 */
#define XPAR_AXI_PWM_0_BASEADDR 0x7EE00000
#define XPAR_AXI_PWM_0_HIGHADDR 0x7EE0FFFF
```



3.4 Writing Code for the Custom Peripheral

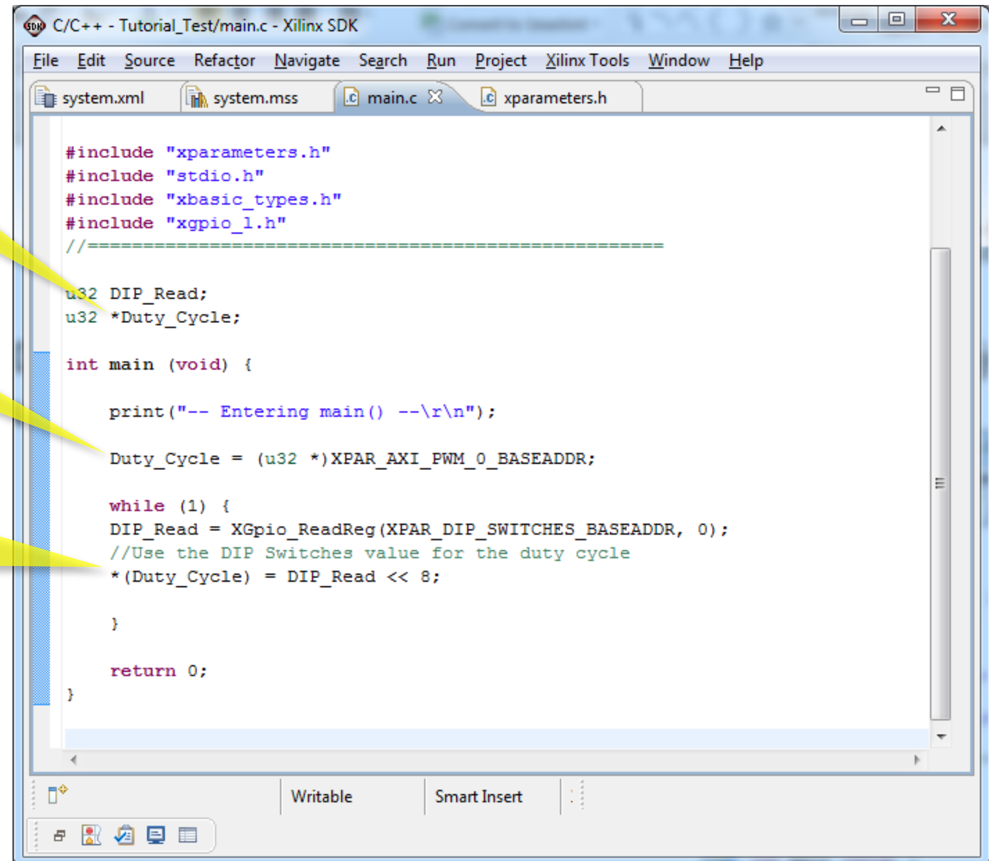
PHYSICS AT THE TERA SCALE
Helmholtz Alliance

4. Double click on the *main.c* file in the **Tutorial_Test** project. We will modify the code to use the custom peripheral instead of the GPIO.

Add new global pointer u32 **Duty_Cycle*

Define base address for the register (see *xparameters.h*)

Replace the GPIO write with code to shift the DIP switch reading to the MSB of 12-bit duty cycle register.

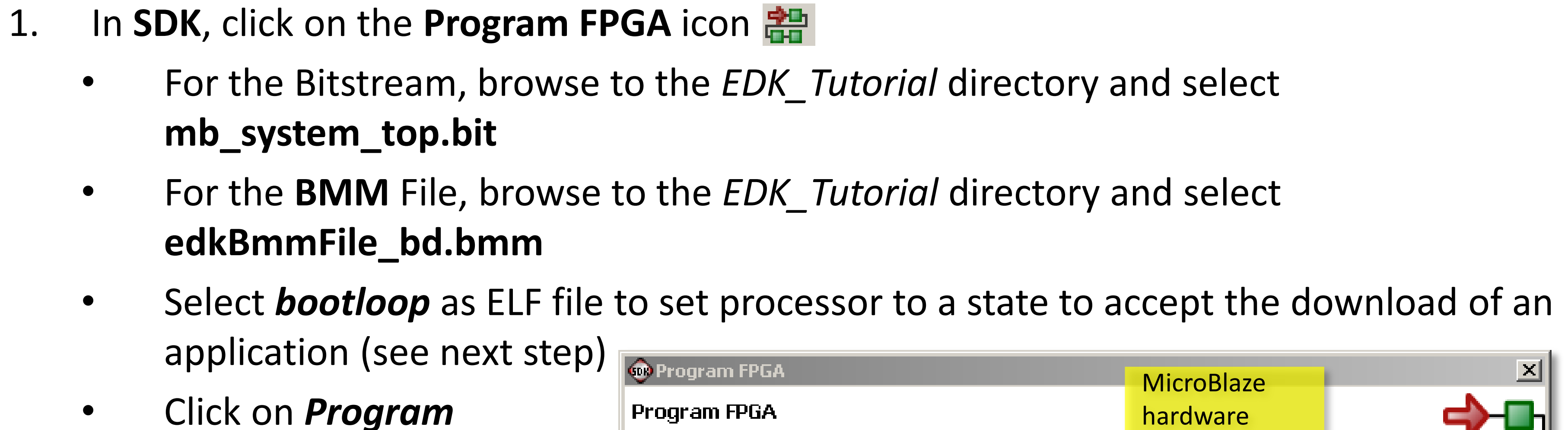


5. Save and close the file. Verify that the code compiled without errors.
6. The system is ready to be downloaded to the board.

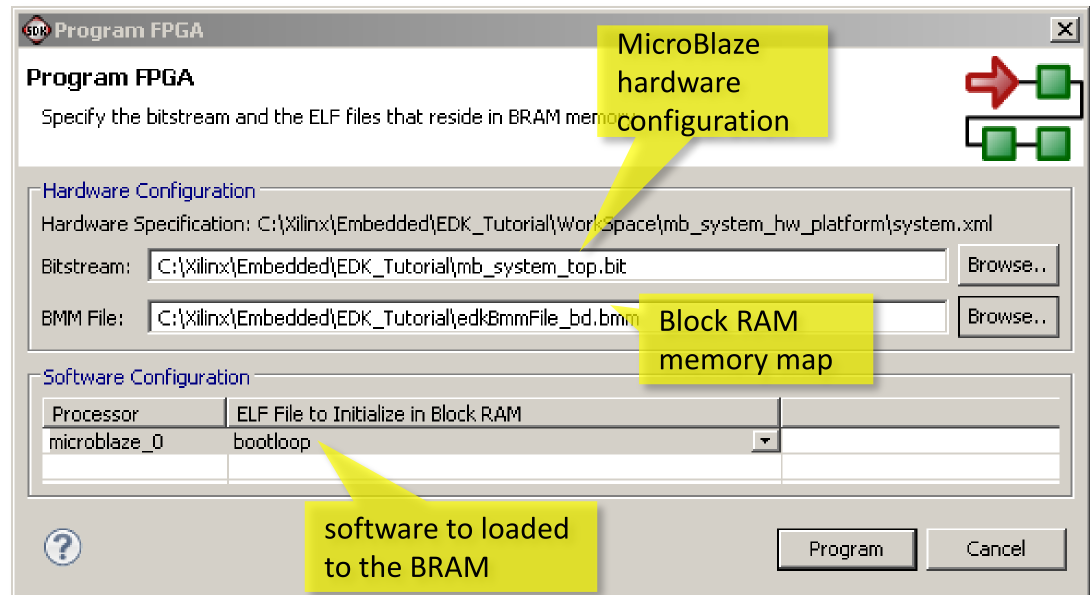
3.5 Testing the Generated System

At first, the hardware configuration is downloaded, and the processor is reset to a state which allows the download of an application (bootloop).

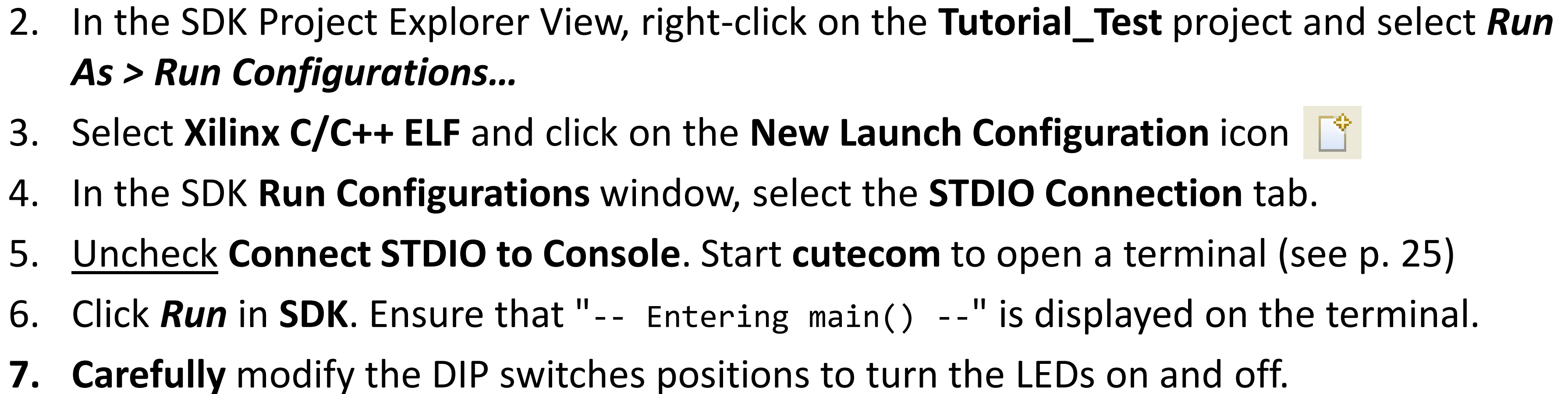
1. In **SDK**, click on the **Program FPGA** icon



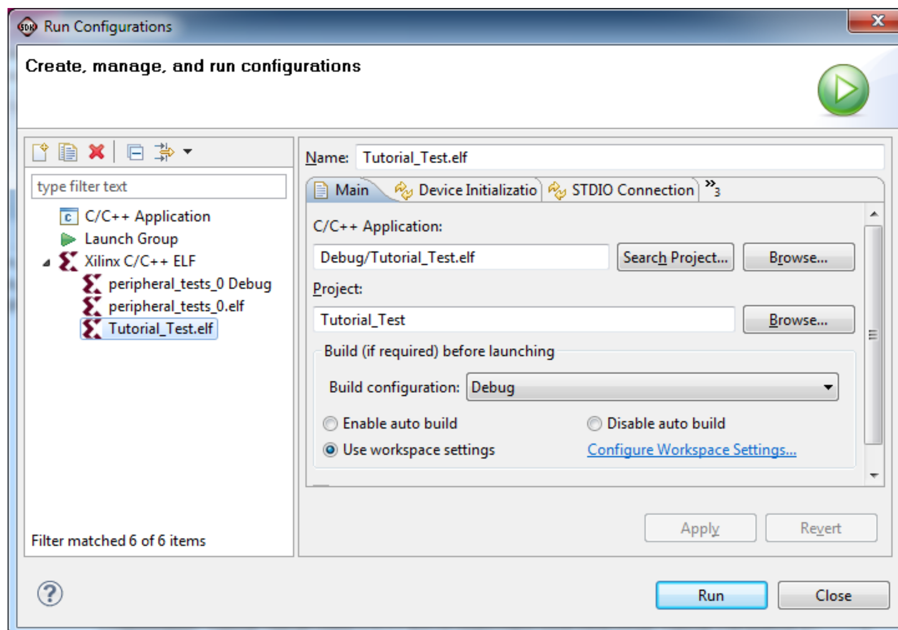
- For the Bitstream, browse to the *EDK_Tutorial* directory and select **mb_system_top.bit**
- For the **BMM** File, browse to the *EDK_Tutorial* directory and select **edkBmmFile_bd.bmm**
- Select ***bootloop*** as ELF file to set processor to a state to accept the download of an application (see next step)
- Click on ***Program***

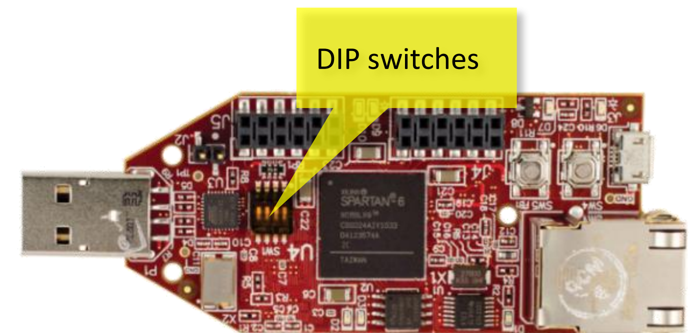


3.5 Testing the Generated System



2. In the SDK Project Explorer View, right-click on the **Tutorial_Test** project and select ***Run As > Run Configurations…***
3. Select **Xilinx C/C++ ELF** and click on the **New Launch Configuration** icon
4. In the SDK **Run Configurations** window, select the **STDIO Connection** tab.
5. Uncheck **Connect STDIO to Console**. Start **cutecom** to open a terminal (see p. 25)
6. Click ***Run*** in **SDK**. Ensure that "`-- Entering main() --`" is displayed on the terminal.
7. **Carefully** modify the DIP switches positions to turn the LEDs on and off.

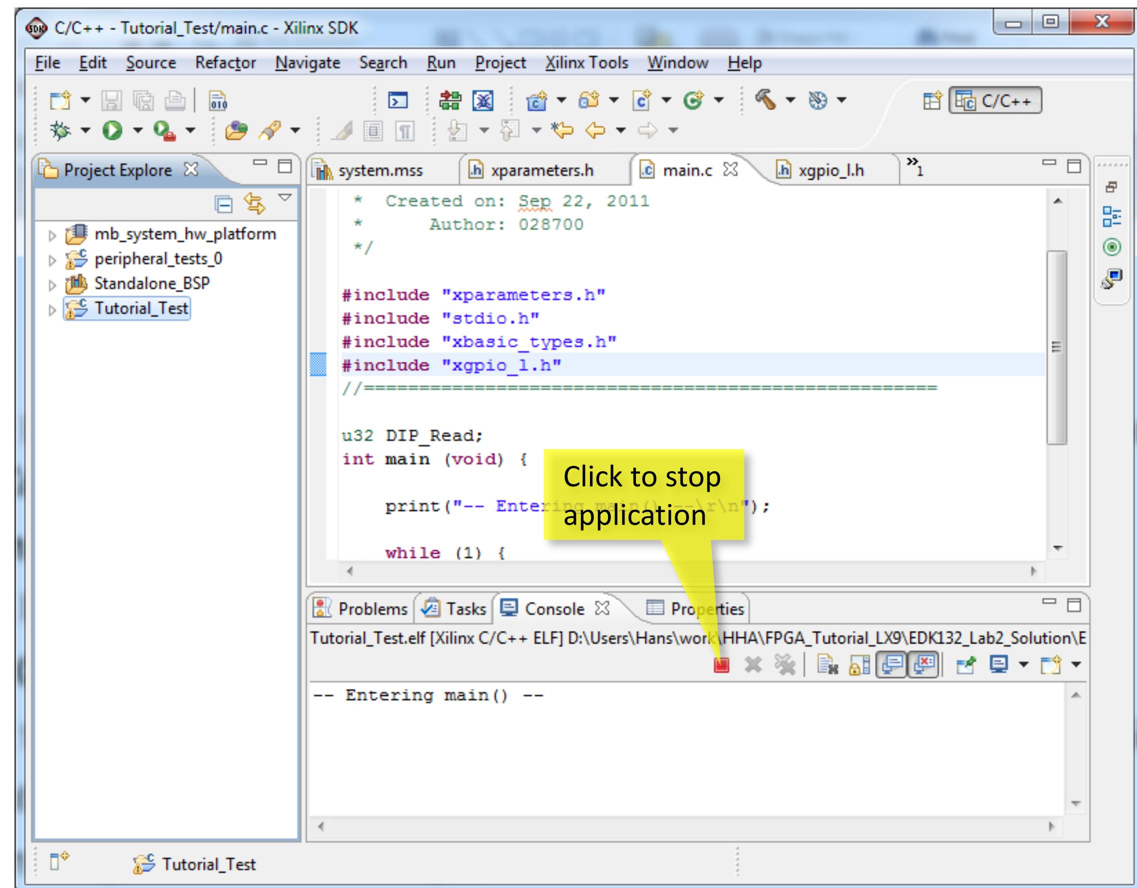




3.5 Testing the Generated System

PHYSICS AT THE TERA SCALE Helmholtz Alliance

8. When finished **Stop/Terminate** the application by pressing the red Stop/Terminate button in the Console window
9. Close **SDK**. This is the end of Tutorial 3.



Tutorial 4: Embedded System Simulation

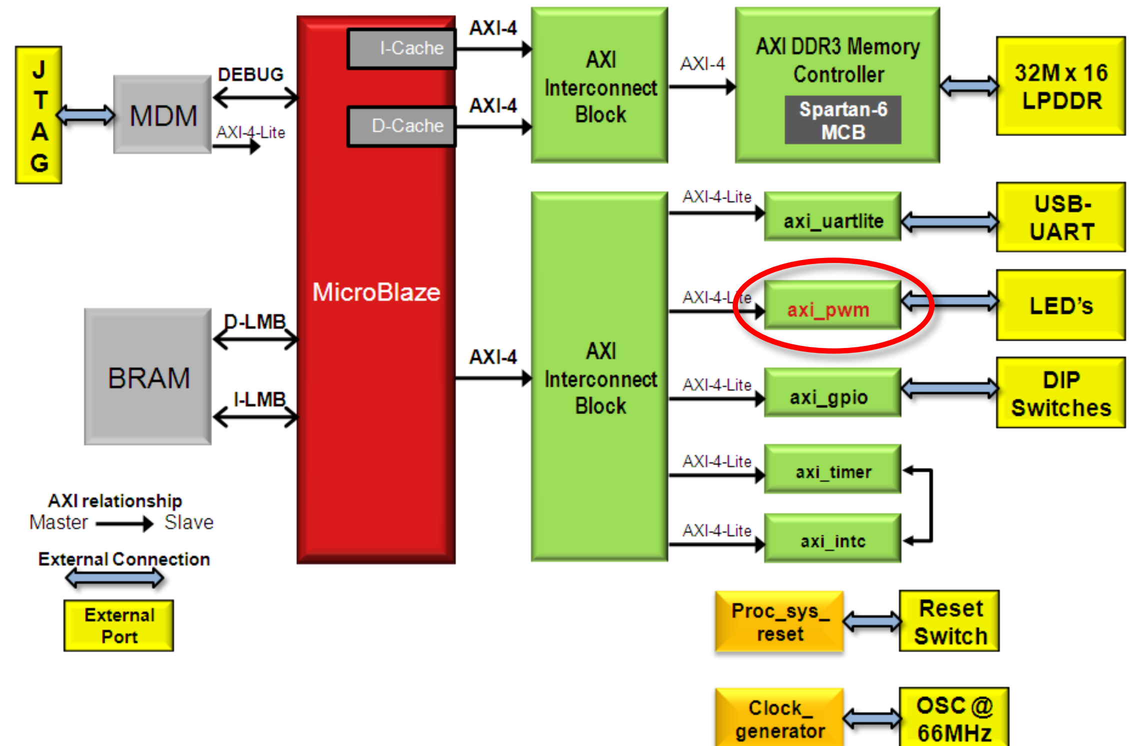
PHYSICS AT THE TERA SCALE
Helmholtz Alliance

Scope of the tutorial

3.1 Setting up the simulation environment

3.2 Adding a testbench file

3.3 Using **Isim** to simulate the system



Hardware Platform

4.1 Setting up the Simulation Environment

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

Very little setup is required to simulate an embedded design from **ISE**. We just need to verify that **ISim** is selected as the main hardware simulator and add the **Tutorial_Test** ELF file created in **SDK**.
NOTE: If this tutorial is your starting point, you can use the **EDK132_Lab03_Solution** to start from.

1. Start ISE Project Navigator and open the **EDK_Tutorial** project.

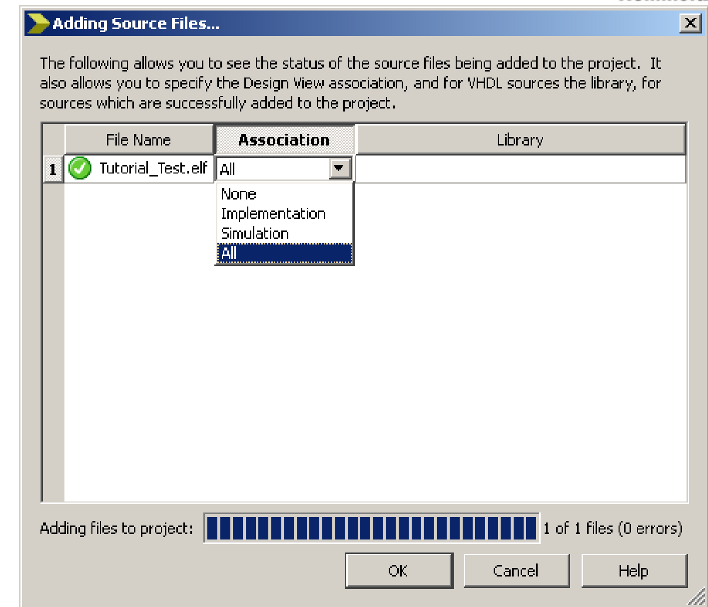
2. Go to **Project > Design Properties** and verify that ISim is selected for simulation. Click **OK**. Choose **Verilog** as Preferred Language (**VHDL** would work as well).

Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog

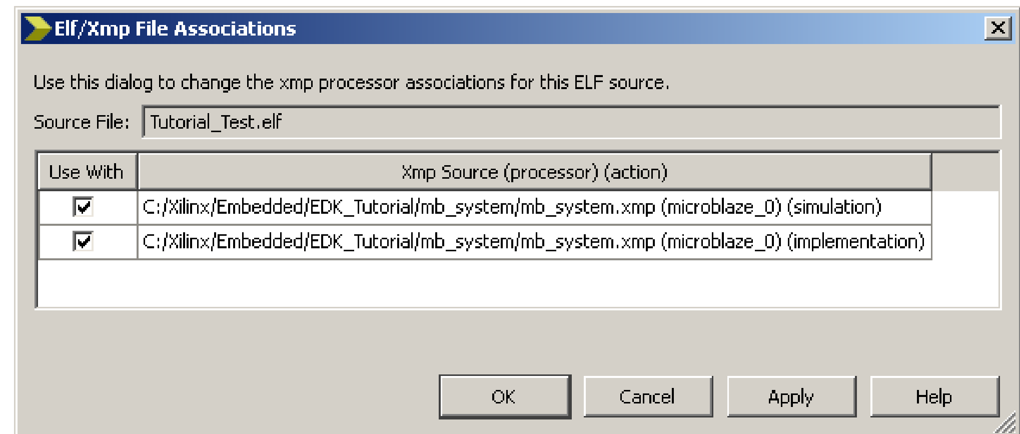
4.1 Setting up the Simulation Environment

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

3. Go to **Project > Add Source**. Browse to the *\EDK_Tutorial\WorkSpace\Tutorial_Test\Debug* directory and open ***Tutorial_Test.elf***.
4. Set the associations to **All** and click OK.



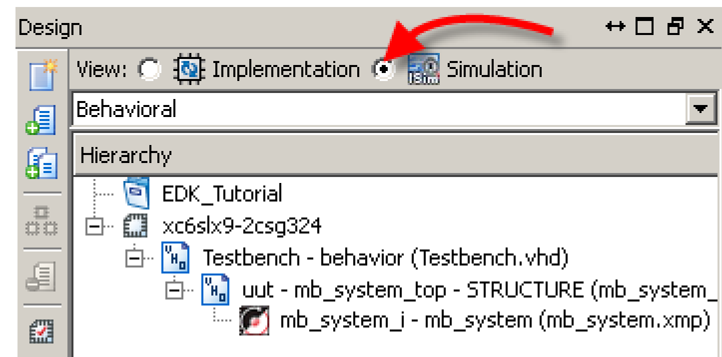
5. Check the box **Use With** box to associate the ELF file to the MicroBlaze processor for **simulation and implementation**. Click **OK**.



4.2 Adding a Test Bench File

To simulate our design we will need to add a HDL test bench. The testbench will instantiate our top level module and provide stimulis for the input ports.

1. In **Project Navigator**, go to **Project > New Source**.
2. Select **Verilog Test Fixture** (or VHDL Test Bench), select *Testbench* for the file name. Click **Next**.
3. Select **mb_system_top** and click **Next**. Click **Finish**.
4. Click on the **Simulation View** radio button. The type of simulation can be changed by using the drop-down list. We will do a **Behavioral** simulation.



4.2 Adding a Test Bench File

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

5. The test bench should be open in the Editor. If not, double-click on **Testbench**.
6. Add stimulus for the reset, and DIP switches. **Reset_in** is active high on the MicroBoard.
7. Save the test bench file.

ISE Project Navigator (O.61xd) - D:\Users\Hans\work\HHA\FPGA_Tutorial_LX9\lectures\EDK132_Lab2_Solution\EDK_Tutor...

File Edit View Project Source Process Tools Window Layout Help

Add **clock** stimulus

Start with active high **RESET** (change from active low)

Wait for another 100ns for **RESET** to finish

Set **DIP switch** settings (PWM duty Cycle)

Change duty cycle after some delay

```
88
89
90  parameter PERIOD = 15;
91  always begin
92      CLK_66MHZ = 1'b0;
93      #(PERIOD/2) CLK_66MHZ = 1'b1;
94      #(PERIOD/2);
95  end
96
97     initial begin
98        // Initialize Inputs
99        USB_Uart_sin = 0;
100       RESET = 1;
101       CLK_66MHZ = 0;
102       DIP_Switches_GPIO_IO_I_pin = 0;
103
104       // Wait 100 ns for global reset to finish
105       #100;
106       #100;
107       RESET <= 0;
108       DIP_Switches_GPIO_IO_I_pin <= 4'b1000;
109
110
111       // Add stimulus here
112       #70000;
113       DIP_Switches_GPIO_IO_I_pin  <= 4'b0001;
114
115
116    end
117
118 endmodule
119
120
```

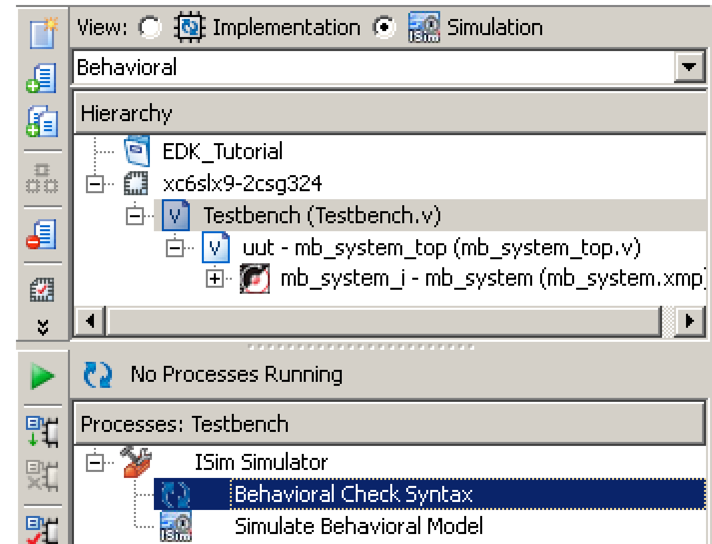
user_logic.vhd | axi_pwm.vhd | Testbench.v

Console | Errors | Warnings | Find in Files Results

4.2 Adding a Test Bench File

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

8. Click on **Testbench – behavior (Testbench.v)** in the **Hierarchy Window**. Then in the Processes expand **ISim Simulator** and double-click **Behavioral Check Syntax**. This will check to see if you have any errors in your testbench.



4.3 Simulating the System

PHYSICS AT THE TERA SCALE Helmholtz Alliance

The simulation can take advantage of code running from BRAMs, providing a cycle accurate MicroBlaze simulation. We will be able to see and trace MicroBlaze execution. **ISim** will use the software application selected to initialize in BRAMs in **XPS**.

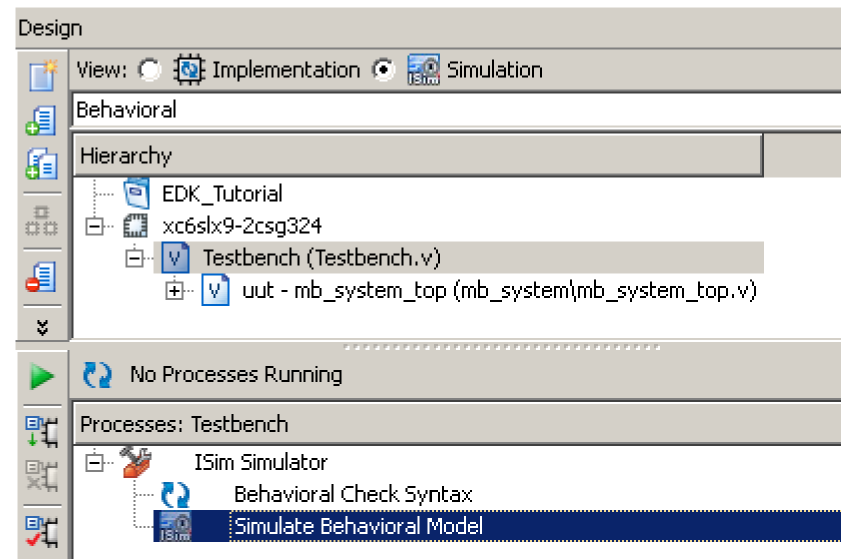
We will select the **Tutorial_Test** (ELF) application from the last tutorial (or the **EDK132_Lab3_Solution** in case you start from here).

At first we need to modify the application in **SDK** since writing to the UART would take too long in a simulation environment. We will need to comment out the print statement. Additionally, we need to make sure the application **Tutorial_Test** is checked to initialize the BRAMs (should have been done on p.68 already).

1. Start **SDK** and select the **Workspace** from **EDK_Tutorial**.
2. Open the **Tutorial_Test main.c** source file and comment out the print statement by adding // at the beginning of the line.
3. Save the **main.c** file. This will automatically compile the new ELF file.
4. Minimize **SDK**. Do not close it.

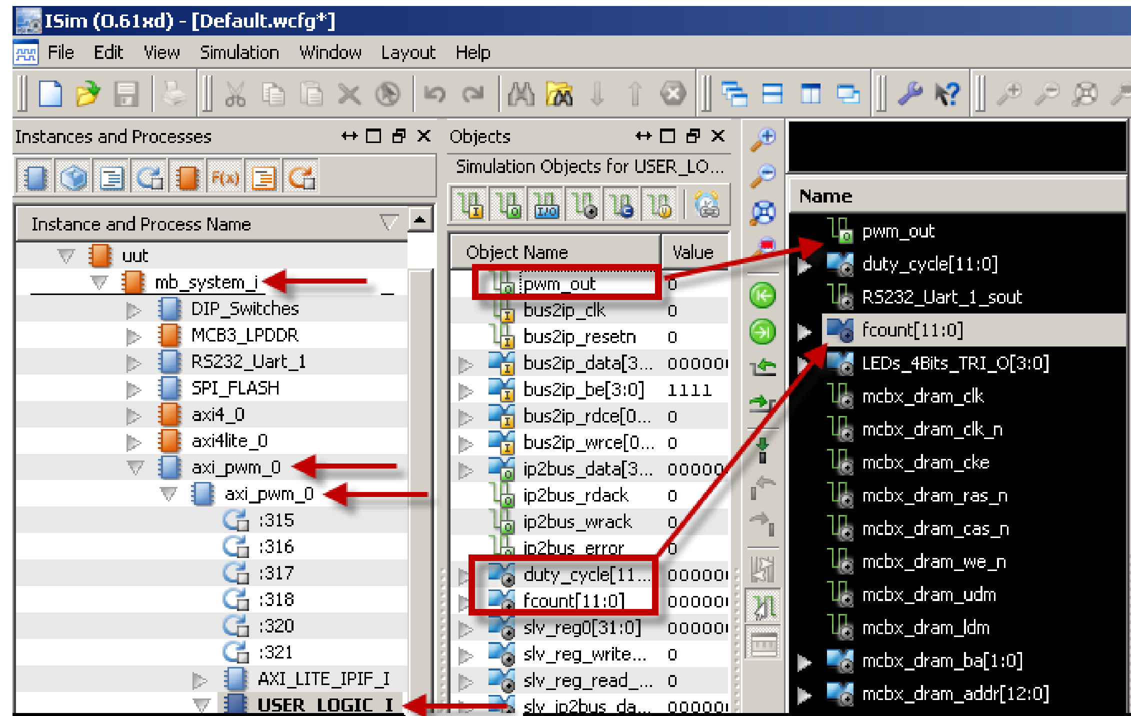
4.3 Simulating the System

5. In Project Navigator, select **Testbench** in the hierarchy view.
6. In the **Processes** window, double-click on **Simulate Behavioral Model**. The tools will load and compile all the necessary files.
7. Wait for **ISim** to open. This could take several minutes as it is running **Simgen**.

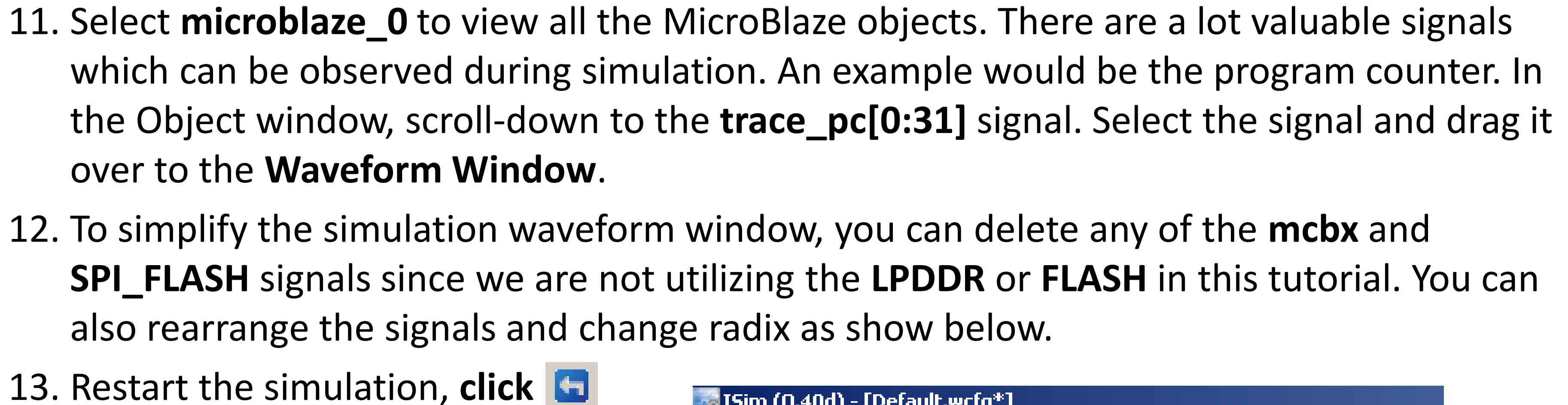


4.3 Simulating the System

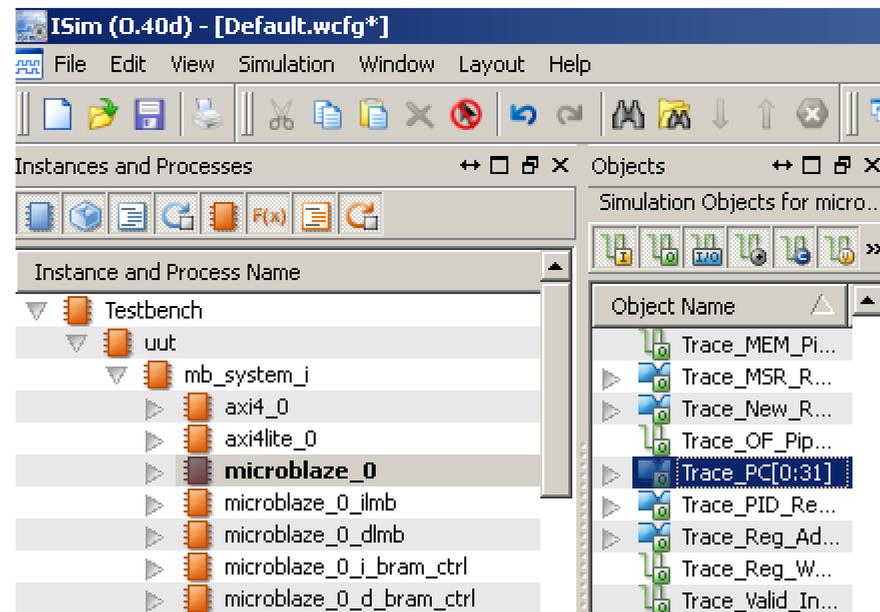
8. We will add some internal signals to the waveform. Select the **Instances and Processes** window on the left side.
9. Expand **testbench,** then **UUT,** then **mb_system_i** to view the embedded system components. (see below.)
10. Select **axi_pwm_0**, then immediately below, expand the **axi_pwm_0** and scroll down to view the **USER_LOGIC_I** signals. From the Objects window, drag **pwm_out**, **fcount[11:0]** and **duty_cycle[11:0]** to the waveform window.



4.3 Simulating the System

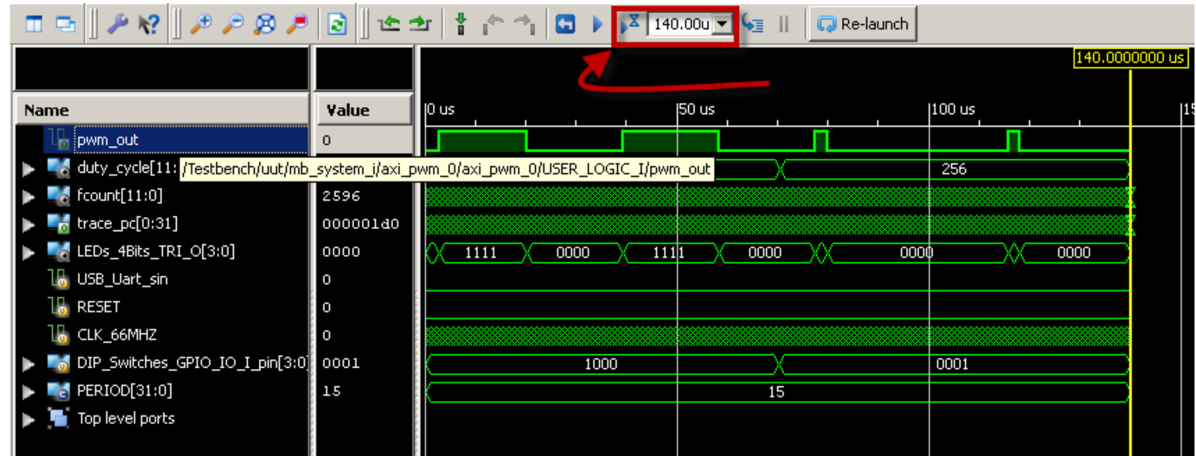


11. Select **microblaze_0** to view all the MicroBlaze objects. There are a lot valuable signals which can be observed during simulation. An example would be the program counter. In the Object window, scroll-down to the **trace_pc[0:31]** signal. Select the signal and drag it over to the **Waveform Window**.
12. To simplify the simulation waveform window, you can delete any of the **mcbx** and **SPI_FLASH** signals since we are not utilizing the **LPDDR** or **FLASH** in this tutorial. You can also rearrange the signals and change radix as show below.
13. Restart the simulation, **click**



4.3 Simulating the System

14. Run the simulation for 140 us. In the console window, type: **run 140us**. Or type **140us** into the window as shown below and click **Run for Specified Time**,. This may take a few minutes to load.



15. Look at the cycles on the AXI bus and observe the **pwm_out** signal. You can see it goes valid twice during this period. In the simulation testbench, we've set to duty cycle widths and they are shown here. For the first and second PWM cycles, the duty cycle is set to 2048, and **pwm_out** is valid until **fcount** exceeds that value. In the third and forth PWM pulses, the duty cycle is set to 256.
16. You can also correlate the program counter (**trace_pc**) to the C application.
17. In SDK, expand the **Tutorial_Test/Debug** project. Double-click on the **Tutorial_Test.elf** executable file.

4.3 Simulating the System

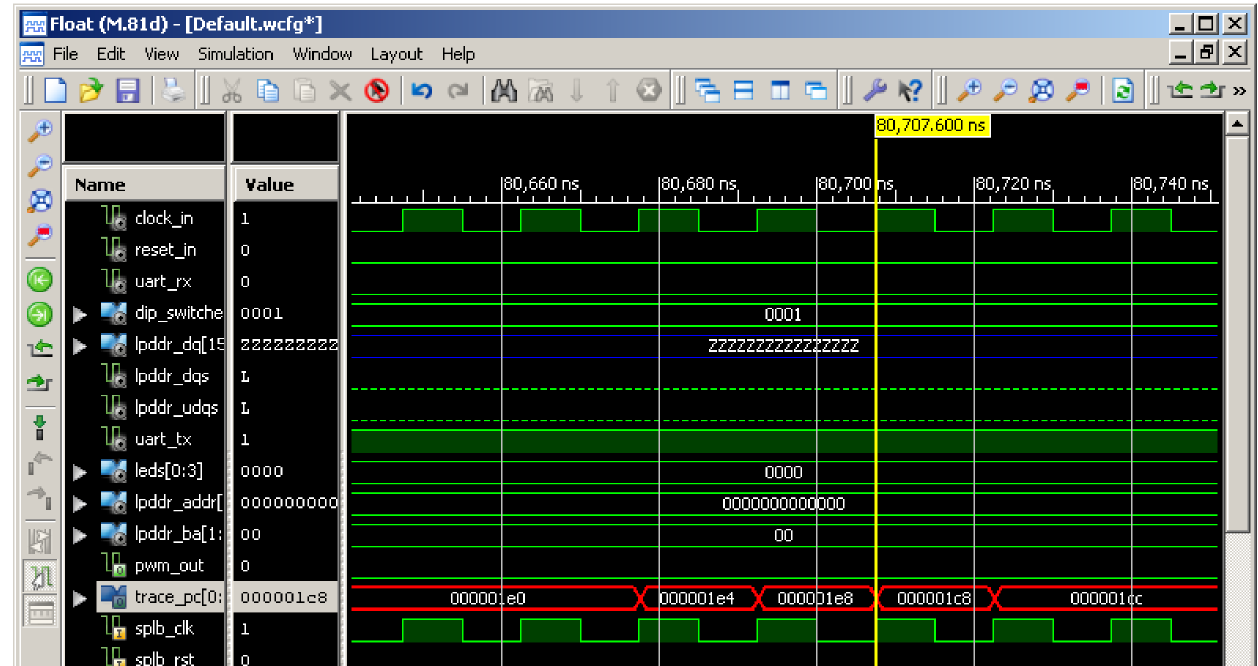
18. Scroll down to view the disassembly of the C code.

```
int main (void) {
 1b0:   3021fff8    addik   r1, r1, -8
 1b4:   fa610004    swi r19, r1, 4
 1b8:   12610000    addk    r19, r1, r0

    //print("-- Entering main() --\r\n");

    Duty_Cycle = (u32 *)XPAR_PLB_PWM_0_BASEADDR;
 1bc:   b000c9c0    imm -13888
 1c0:   30600000    addik   r3, r0, 0
 1c4:   f8600844    swi r3, r0, 2116    // 844 <Duty_Cycle>

    while (1) {
        DIP_Read = XGpio_ReadReg(XPAR_DIP_SWITCHES_BASEADDR, 0);
 1c8:   b0008142    imm -32446
 1cc:   30600000    addik   r3, r0, 0
 1d0:   e8630000    lwi r3, r3, 0
 1d4:   f8600840    swi r3, r0, 2112    // 840 <DIP_Read>

        //XGpio_WriteReg(XPAR_LEDS_4BITS_BASEADDR, 0, DIP_Read);

        //Use the DIP Switches value for the duty cycle
        *(Duty_Cycle) = DIP_Read << 8;
 1d8:   e8800844    lwi r4, r0, 2116    // 844 <Duty_Cycle>
 1dc:   e8600840    lwi r3, r0, 2112    // 840 <DIP_Read>
 1e0:   64630408    bslli   r3, r3, 8
 1e4:   f8640000    swi r3, r4, 0
```

PHYSICS AT THE TERA SCALE
Helmholtz Alliance

4.3 Simulating the System

19. In the **ISim** simulation window, look at the **trace_pc** bus (change the radius to HEX). The PC goes between 0x1C8 and 0x1E4 which are the instructions contained in the while loop.



20. There are many more MicroBlaze signals which can be observed.

21. Close **ISim** when finished. Close **SDK**.

22. In **Project Navigator**, switch back to the **Implementation** view by clicking the Implementation radio button.

That concludes this tutorial.