

# PDE3411 Laboratory Worksheet 6

## Xilinx Vivado/SDK Tutorial

02/11/2020

This tutorial shows you how to create and run a simple MicroBlaze-based system on a Digilent Nexys-4 prototyping board. Such a system requires both specifying the hardware architecture and the software running on it. These two are then combined into one FPGA configuration, which is used to configure the Artix-7 FPGA located on the Nexys-4 board.

### The Sample Design

This tutorial revolves around creating a system (hardware and software) that can output a simple message via the UART (in our case over USB-UART) and blink some leds on the board. The simplest of programmable hardware architectures that can be built involves a single processor, a MicroBlaze in our case, and some minimal support for it (memory, interconnect). On top of this, the functionality for communicating over the UART is needed. Furthermore, without great effort, one can add functionality for sensing and controlling other physical components present on the Nexys-4 board (such as push buttons, switches, temperature sensor, accelerometer).

Naturally, a programmable hardware architecture will require a program (software) to run on it. In a second phase, the tutorial will address the steps needed to build a simple software environment via Xilinx SDK.

Finally, the last part of the tutorial describes how to finally configure the FPGA with the hardware and software you just built, how to run your design and actually display the output of the UART.

A simplistic overview of the design flow used for building the whole system is depicted in Figure 1. Note that similar steps are taken both on the hardware and on the software flows, but the terminology is somewhat different for historical reasons.

### Creating the System Architecture (Hardware)

Hardware architectures are created using Xilinx Vivado, a GUI that helps you to specify

- Which processors, memory blocks and other soft IPs (peripherals) to use how the different IPs are interconnected
- The memory map, i.e. for addresses for memory mapped IO/peripherals
- How the different input/output signals map to actual pins on the FPGA and thus resources on the board

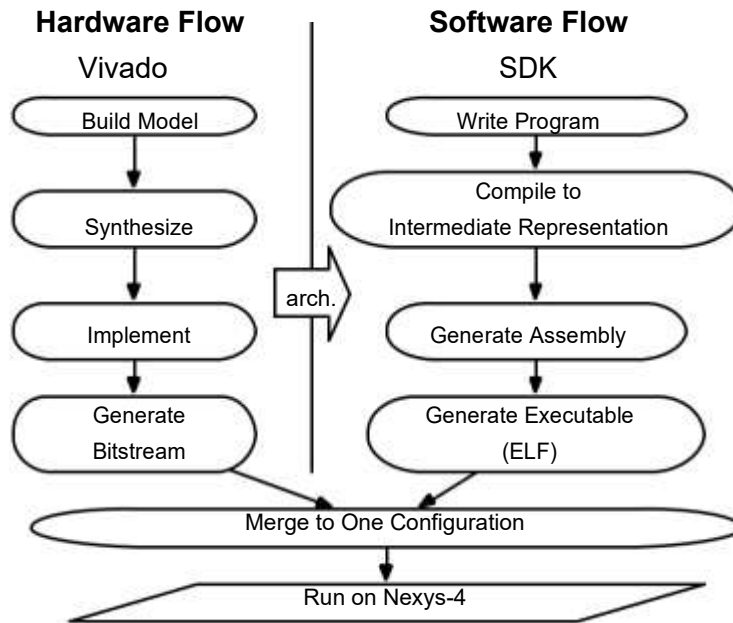


Figure 1: Overview of the Design Flow in this Tutorial (simplistic)

The output from Vivado is that part of the FPGA configuration that describes the hardware of your system. The FPGA and board resources require this configuration to emulate the hardware architecture you described in Vivado.

The hardware architecture in this tutorial, depicted in Figure 2, will include:

- A processor: MicroBlaze
- Associated local memory, dual ported - one port for data, one for instructions
- A few peripherals (IPs): UART (handling serial connections), GPIO (handling onboard leds), Timer (for getting performance data)
- A bus connecting the peripherals with the processor: AXI
- A clock generator and a reset generator associated with the whole system

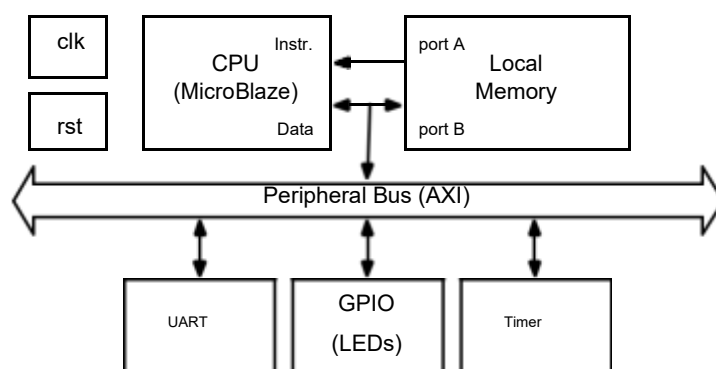


Figure 2: Overview of the Hardware Architecture in this Tutorial

## 1 Invoke the Vivado IDE and Create a Project

1. Open the Vivado IDE by clicking the desktop icon or by typing vivado at a terminal command line. From the Quick Start page, select Create New Project. The New Project wizard opens.
2. In the Project Name dialog box, type the project name and location.

Make sure that Create subdirectory is checked. Click Next.

3. In the Project Type dialog box, select RTL Project. Click Next.
4. In the Add Sources dialog box, ensure that the Target language is set to VHDL. Leave the Simulator Language set to its default value of Mixed. Click Next.
5. In Add Existing IP dialog box, click Next.
6. In Add Constraints dialog box, click Next.
7. In the Default Part dialog box, select Boards and choose **Nexys-4 DDR**. Click Next.
8. Review the project summary in the New Project Summary dialog box before clicking Finish to create the project.

## 2 Create an IP Integrator Design

1. From the Flow Navigator window (usually leftmost in Vivado), under IP Integrator item, select Create Block Design. The Create Block Design dialog box opens, as in Figure 3.
2. Specify the IP subsystem design name. For this step, the tutorial will use the default value, but any name without spaces will do. Leave the Directory field set to its default value of <Local to Project>.
3. Leave the Specify source set drop-down list set to its default value of Design Sources.
4. Click OK in the Create Block Design dialog box.
5. In the Block Design area, Diagram tab should look like in Figure 4. You can either select Add IP here, or find the Add IP button on the left side of the Diagram tab (you will use this button when the design is not empty).
6. As shown in Figure 5, type **micr** in the Search field to find the MicroBlaze IP, then select MicroBlaze and press the Enter key.

Note The IP Details window can be displayed by clicking CTRL+Q key on the keyboard.

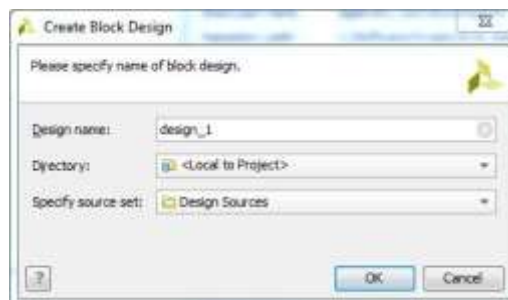


Figure 3: Na Figure 6: Add IP

This design is empty. Press the  button to add IP.

Figure 4: Add IP

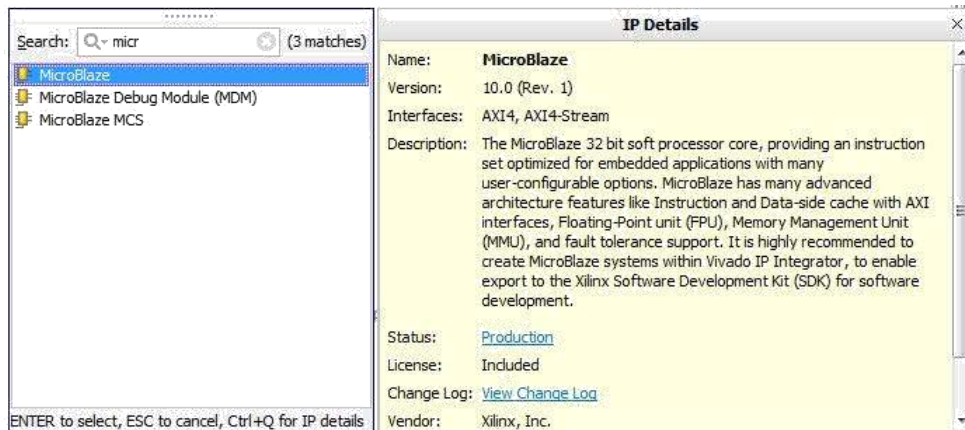


Figure 5 Search field

## Use the Board Tab to Connect to Board Interfaces

There are several ways to use an existing interface in IP Integrator. Use the Board tab to instantiate some of the interfaces that are present on the Nexys-4 board.

1. Click the Board tab. You can see as in Figure 6 that there are several components listed in that tab. These components are present on the Nexys-4 board. They are listed under different categories in the Board window.

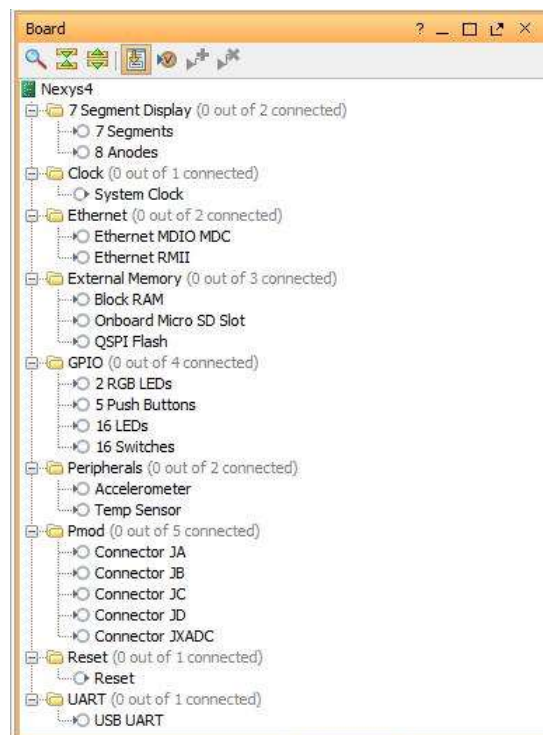


Figure 6 Using the board part interfaces

2. From the Board window, select **System Clock** from the Clock folder and drag and drop it into the block design canvas. This will allow you to use the 100MHz clock generated on board and add clocking logic to your design. (Since our designs are all synchronous, clocks will always be required!) Click OK in the Auto Connect dialog box.

3. From the Board window, select **16 LEDs** from GPIO folder and drag and drop it into the block design canvas. This will allow you control the 16 on/leds on the board. Click OK in the Auto Connect dialog box.
4. From the Board window, select **USB UART** and drag and drop it into the block design canvas (see Figure 7). This will allow you to communicate via the UART interface of the board with your design. This will be used to display simple text messages from your software running on your board, in a console connected with the board via a serial. Click OK in the Auto Connect dialog box.

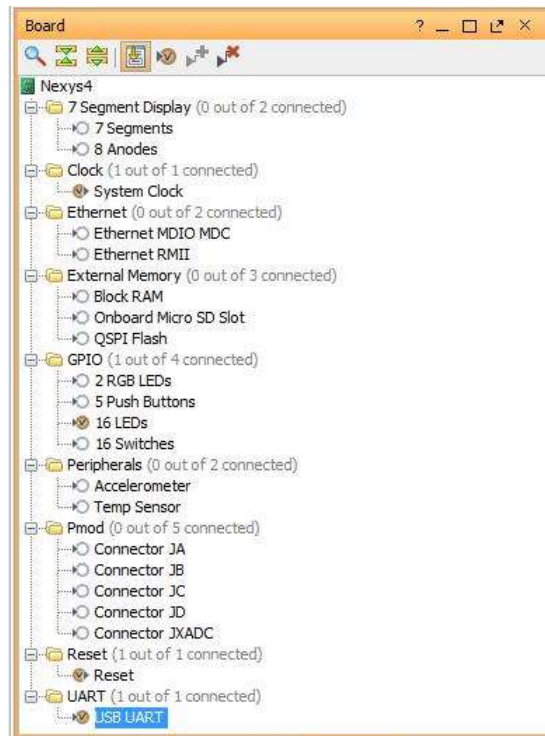


Figure 7 Selected board interfaces

6. The block design should now look like in Figure 8.

## Run Block Automation

1. Click Run Automation, displayed in Figure 9.
2. The Block Automation dialog box opens as shown in Figure 10. Here we recommend you set the following parameters (for your MicroBlaze processor):
  - a. set **Local Memory** to **64KB**.
  - b. leave the **Local Memory ECC** to its default value of **None**.
  - c. leave the **Cache Configuration** to its default value of **None**. Our design will not use cache for now, but for designs using off-chip memory this is a must, or the system will be too slow.
  - d. change the **Debug Module** option to **None**. To simplify the design we will not use debug, but if you want to run your program step-by-step on the board, a debug module will be needed. Refer to the Xilinx UG940 document for more details on how to set up a debug module and also an integrated logic analyser ILA used to monitor design signals.
  - e. leave the **Peripheral AXI Port** option to its default value of **Enabled**.
  - f. make sure the **Clock Connection** option is set to the **100MHz** clock output of the clock wizard: /clk\_wiz\_0/clk\_out1 (100MHz).

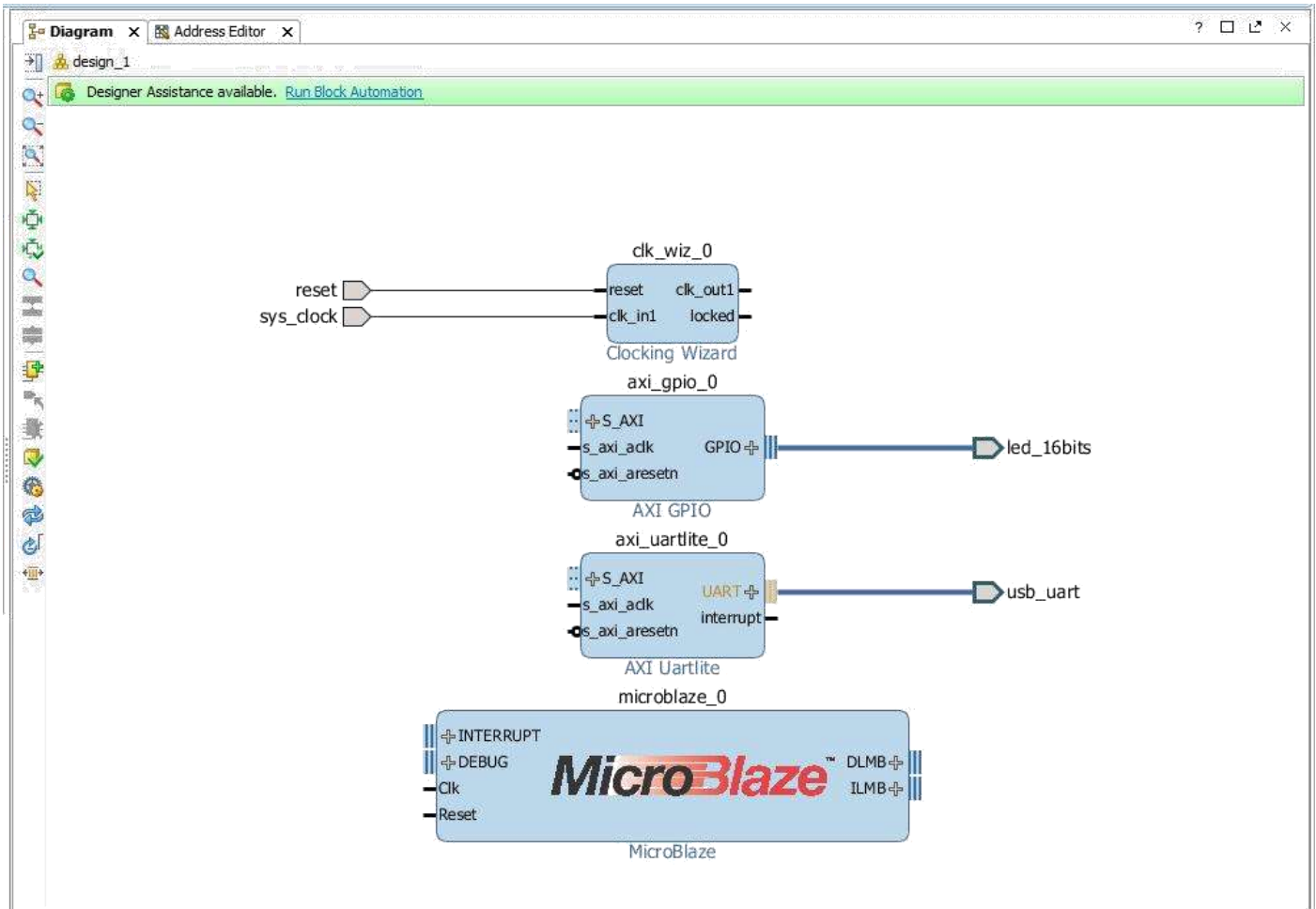


Figure 8 Block Design after connecting the UART

3. Click **OK**.

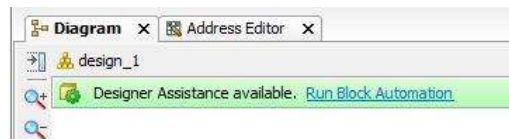


Figure 9 Run block automation

This generates a basic MicroBlaze system in the IP Integrator diagram as shown in Figure 11.

## Use Connection Automation

Vivado also helps you connecting the blocks present in your design. Run Connection Automation provides several options that you can select to make connections.

1. Click **Run Connection Automation** as shown in Figure 12.
2. The Run Connection Automation dialog box opens, as in Figure 13. Check all the interfaces in the left pane as in the figure. You may leave unchecked interfaces, if you plan to connect them at a later time. You should also inspect the options you have for the items you select in the left pane. However, at this point there is no need to change anything.
3. Once you click **OK** the blocks in your design will be connected

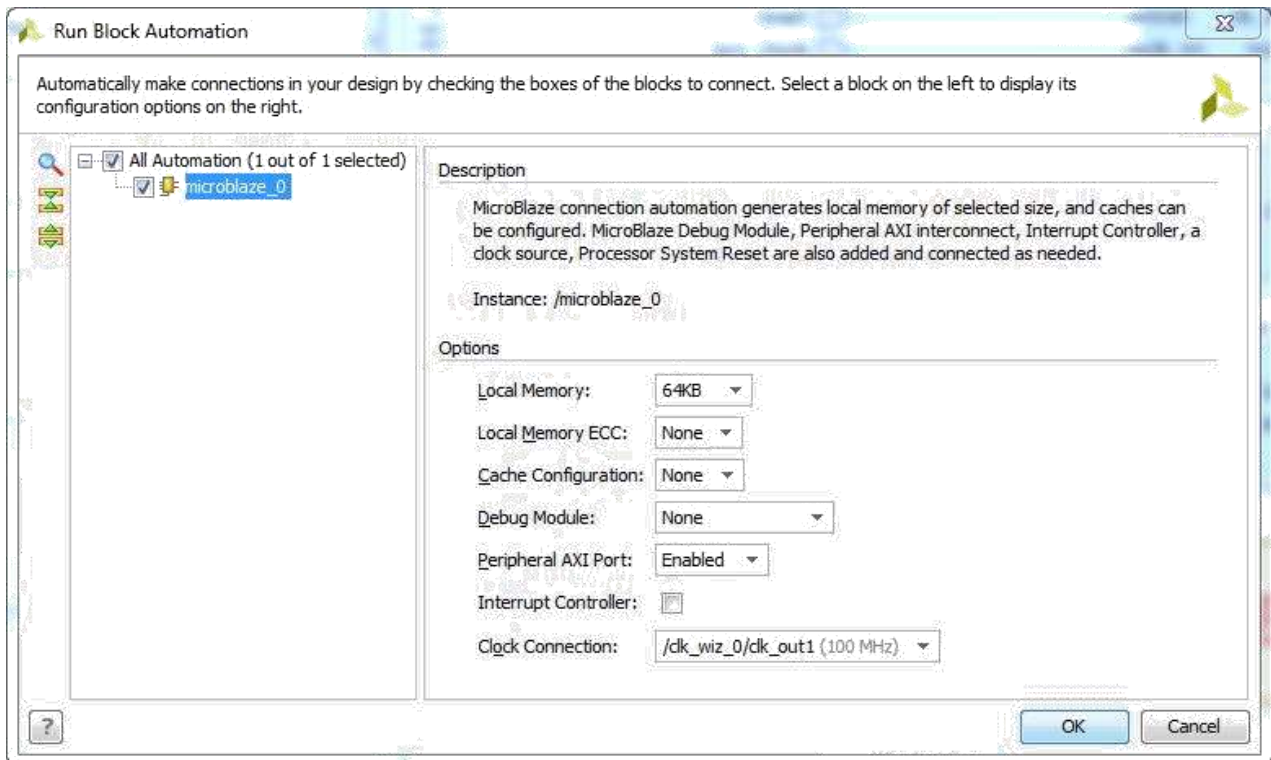


Figure 10: Run Block Automation dialogue box

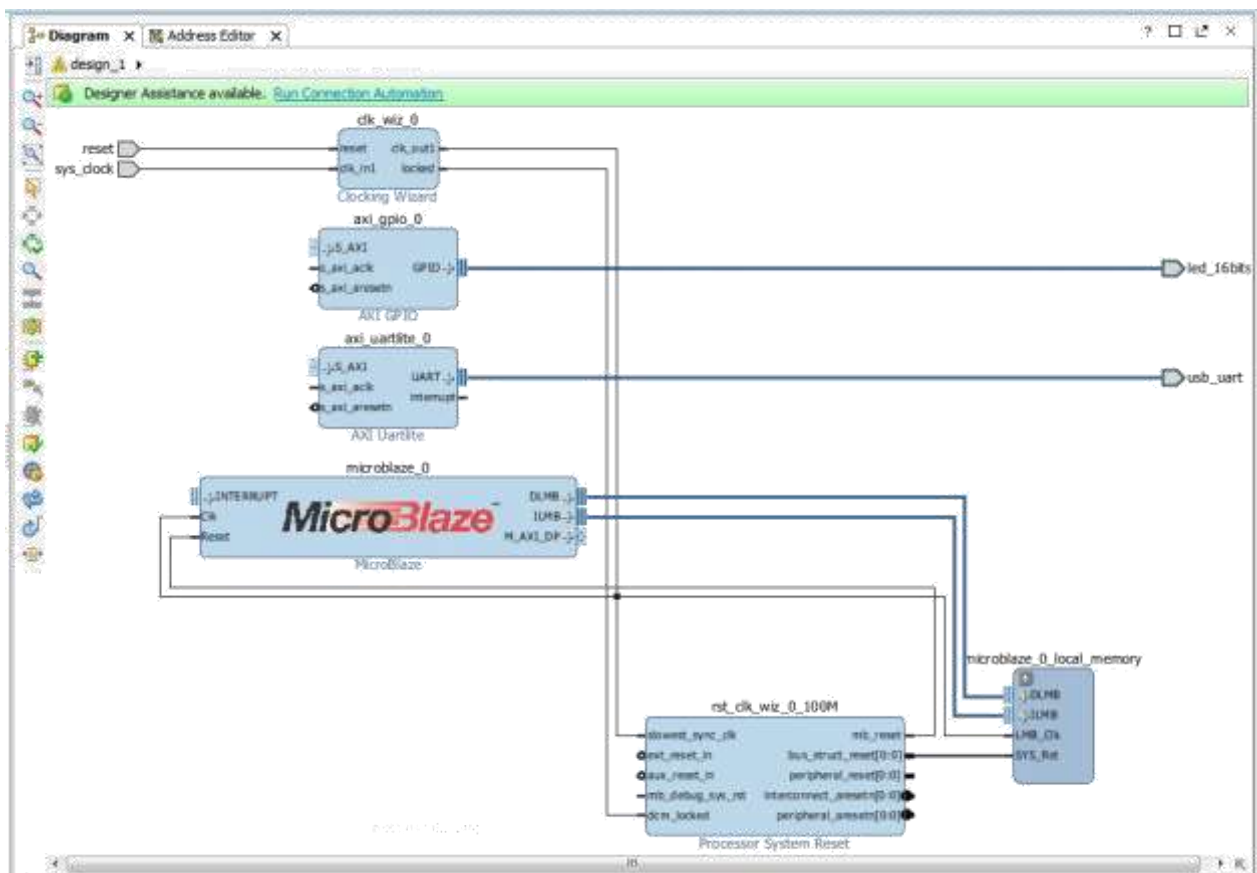



Figure 11: MicroBlaze System

You may use **Run Connection Automation** every time you add new IPs to your design, which need to be connected to the rest of the system.

## Adding More IPs

Some designs require more IPs than the one we introduced until now. These can be added the same way you added the MicroBlaze processor, by clicking the Add IP button. Let us add a timer.

1. Select **Add IP** button in the left side of the Diagram tab, Block Design area.
2. As shown in Figure 14, type **time** in the Search field, select **AXI Timer** and press the **Enter** key.
3. You should now **Run Connection Automation** again.
4. Finally, you may also have Vivado redraw the block diagram by clicking the Regenerate Layout button  found in the same toolbar with the Add IP button.

At this point your block diagram should look similar to the one in Figure 15. The relative placement of your IPs might be slightly different.

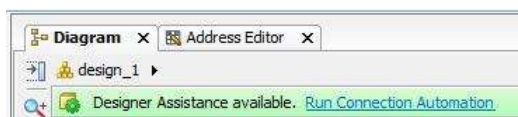


Figure 12: Run connection automation

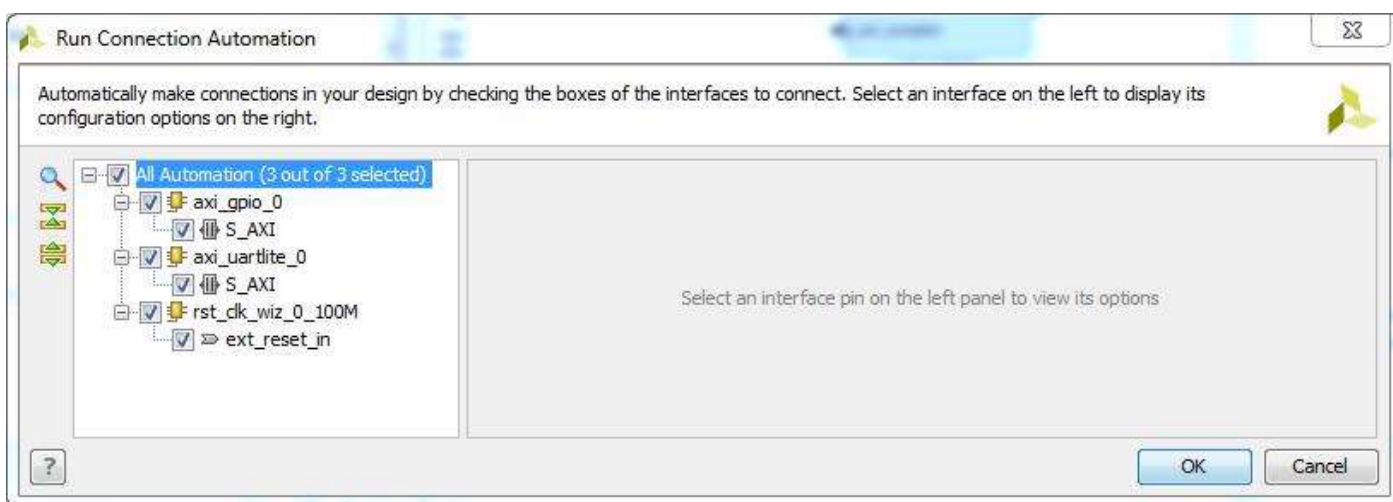


Figure 13: Run connection automation dialog box





## Inspecting and Manually Changing a Design

There are ways to individually check and alter connections and IPs without appealing to the automatic Vivado features for connections and block automation.

For instance, we can check the signals in the block design, and change their attributes if necessary. The Reset input for the system comes from a button, meaning it's driven low (logic 0) when pushed. As it is right now the clock logic requires a reset active high (logic 1). This is why an inverter is automatically added for us to invert the reset signal. Otherwise, a fully implemented system will be thus always reset. . . unless we push the reset button! Let us make sure the two signals have the same polarity:

1. Select the reset input by clicking on it in the block design Diagram. Information about the selected item will be displayed as in Figure 16. Notice that the Polarity is **Active Low**.
2. Now let us examine the clocking logic. Double click on the Clocking Wizard block. This should bring up the Re-customize IP dialog block for the Clocking Wizard, as shown in Figure 17. The parameters for all IPs can be inspected and changed via this dialog box, invoked in a similar way.
3. In the **Output Clocks** tab, notice the Reset Type section, which is Active High. Keep this unchanged as an inverter is used (Figure 18).
4. Click **OK**.
5. Once connected your design should look like in Figure 19 (not as in Figure 20).

The hardware architecture is now complete.

## 3 Memory-Mapping the Peripherals in IP Integrator

The processor (MicroBlaze) sees its memory and peripherals at different addresses in the address space. Accessing a specific address may access the memory or a peripheral, depending on what is mapped at that specific address. To examine the address map (what is mapped where in the address space) select the **Address Editor** tab, located beside the Diagram tab in the Block Design window. The memory map should look similar to the one in Figure 21. You may change any of the mappings here, but Vivado is usually very good at selecting a map that minimizes the address decoding complexity, so there is no need to change anything here.

## 4 Validate Block Design

To run design rule checks on the design:

1. Click **Validate Design** button on the toolbar, or select **Tools->Validate Design**.
2. The Validate Design dialog box informs you that there are no critical warnings or errors in the design. Click **OK**.
3. Save your design using **Ctrl+S** or **File -> Save Block Design**.

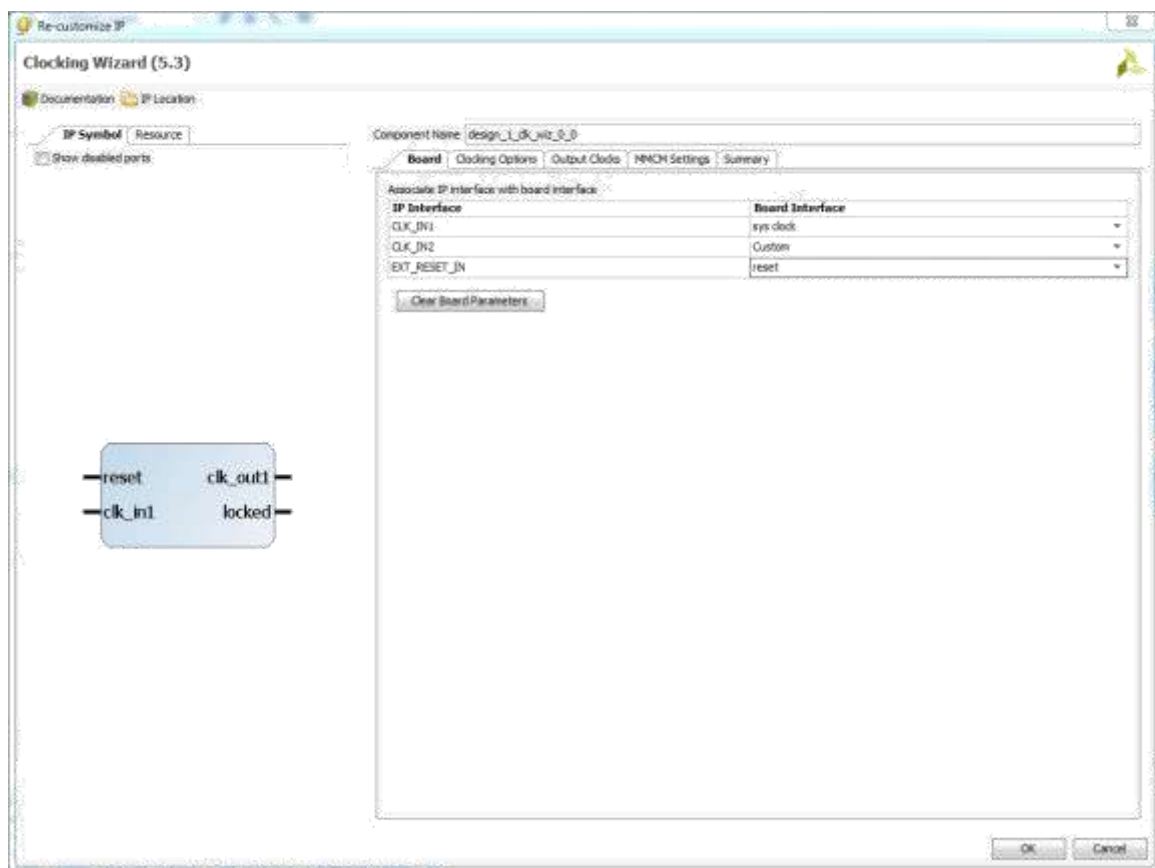


Figure 17: Re-Customize IP Dialog Box

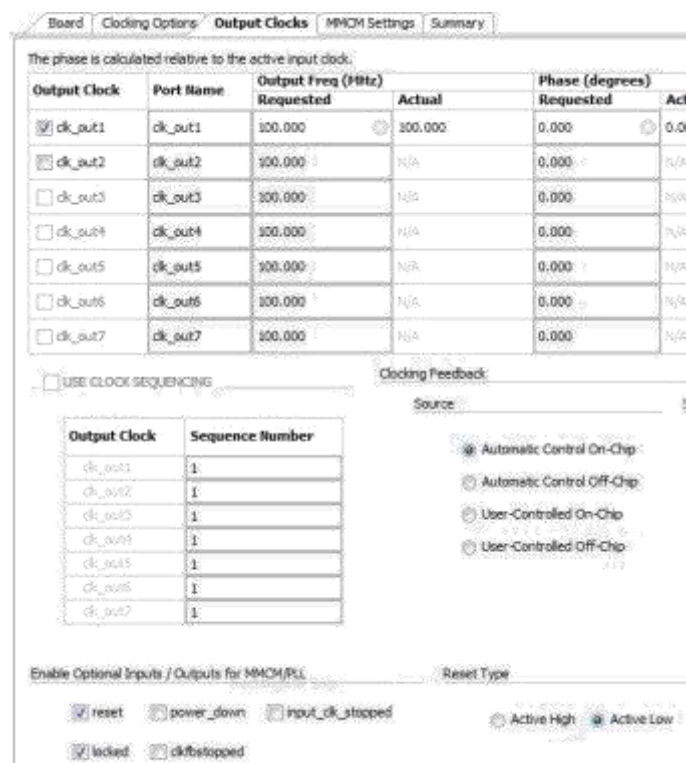


Figure 18: Keep Active High unchanged Polarity for Clocking Wizard (The figure is wrong)

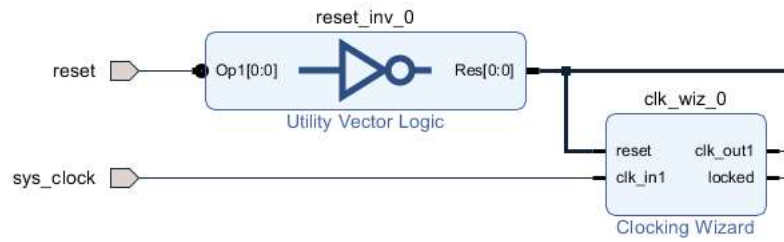


Figure 19: Clocking Wizard with an inverter to align the polarity

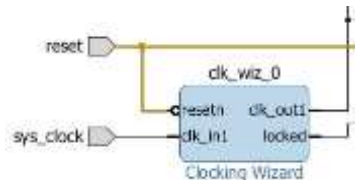


Figure 20: Clocking Wizard with inverted Resetn Connected to the Reset Input Port (if no inverter presented, not in our case)

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF

Figure 21: System Address Space Map

## 5 Generate Output Products

1. In the Sources window (see Figure 22), select the block design, right-click it and select **Generate Output Products**. Alternatively click **Generate Block Design** in the Flow Navigator. The Generate Output Products dialog box opens (see Figure 23).
2. Select **Global** in Synthesis Options, and click **Generate**.
3. Click **OK** in the Generate Output Products dialog box.

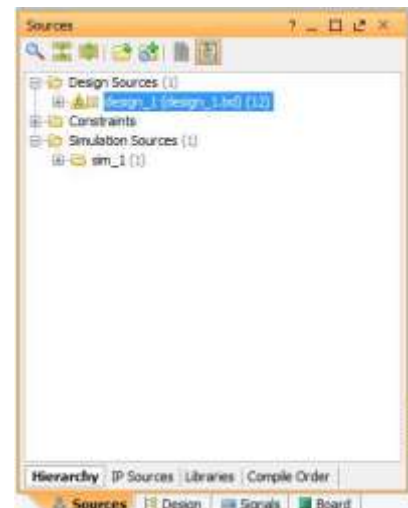


Figure 22: Sources Window

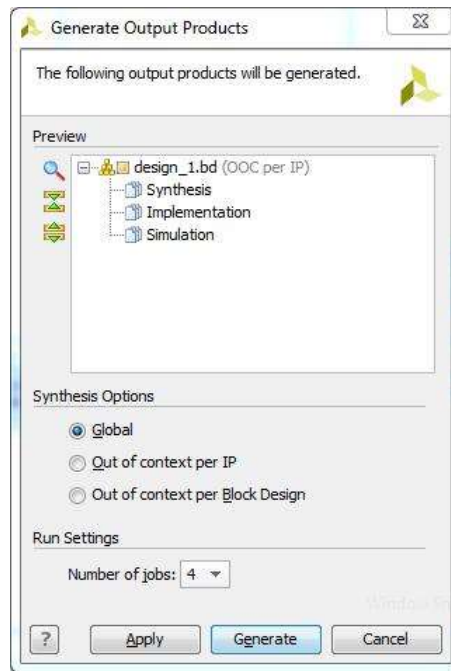


Figure 23: Generate Output Products Dialog Box

## 6 Create Top-level HDL Wrapper

1. Under Sources, right-click your design and click **Create HDL Wrapper**
2. In the Create HDL Wrapper dialog box, select **Let Vivado manage wrapper and auto-update**, as in Figure 24. Click **OK**.

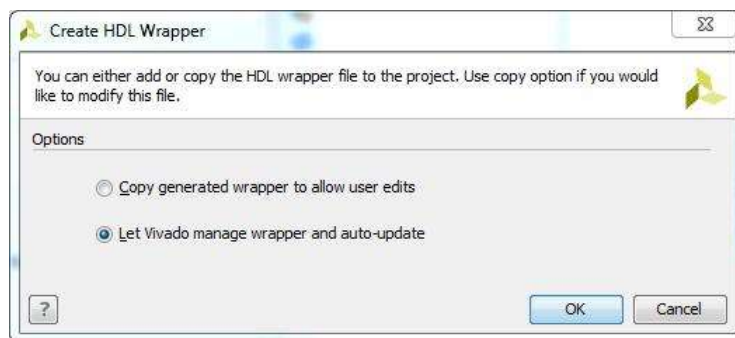


Figure 24: Creating an HDL Wrapper

## 7 Synthesize Design

1. In the Flow Navigator, click on **Run Synthesis**. This will take some time, but you can check the progress and the output messages in the lower right window, Synthesis **Log** tab (see Figure 25).
2. After synthesis finishes, you can choose to have a look at the synthesised design, but finally you should **Run Implementation** in the Synthesis Completed dialog box (see Figure 26). While the synthesis step refers to refining the design to a generic format (e.g. similar to software being compiled to an intermediate format), the implementation step generates a design for the specific FPGA on your board (e.g. generating the assembly code from the intermediate format). Click **OK**.

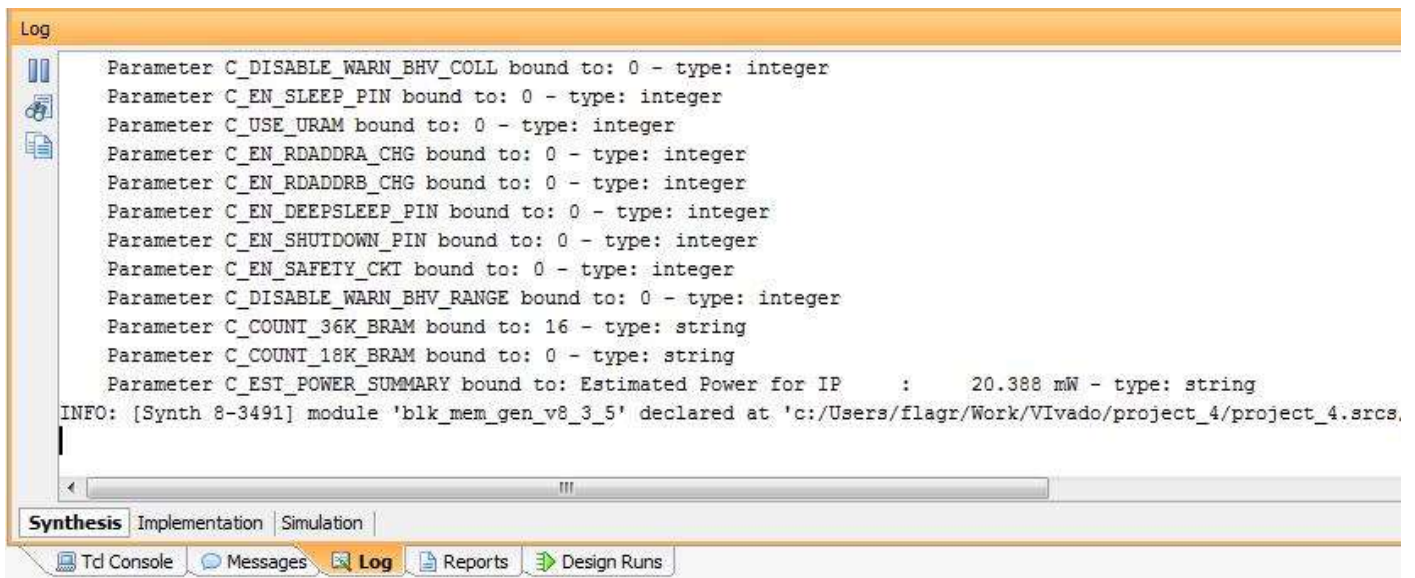


Figure 25: Following the Synthesis Log



Figure 26: Synthesis completed dialog box

## 8 Take the Design through Implementation

The final result of the hardware design step is a bitstream - which is the low level configuration for the target FPGA. In principle choosing **Generate Bitstream** in Flow Navigator will run through all the steps needed to generate this configuration. If you are running through the process as mentioned in the previous section, and you chose to run implementation directly, a new dialog box will open, as in Figure 27. Choose **Generate Bitstream** and click **OK**. To make an analogy with a software flow, if the implementation step generates the assembly code, the generate bitstream step creates the machine code. Note that this is still only the hardware to be emulated, but has no software to be run on it; this will be created later via SDK and incorporated in the bitstream as memory contents (if it's located in the on-chip memory).

Throughout the whole process, you may also check the different sort of reports from the different steps, available in the lower window, **Reports** tab, as in Figure 28.

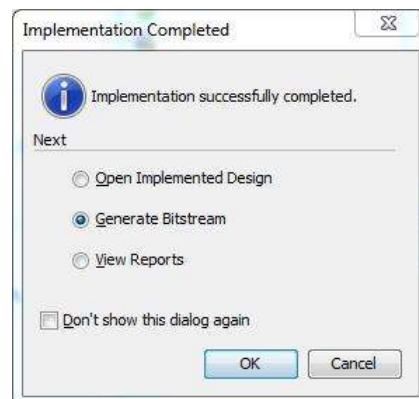


Figure 27: Implementation Completed Dialog Box

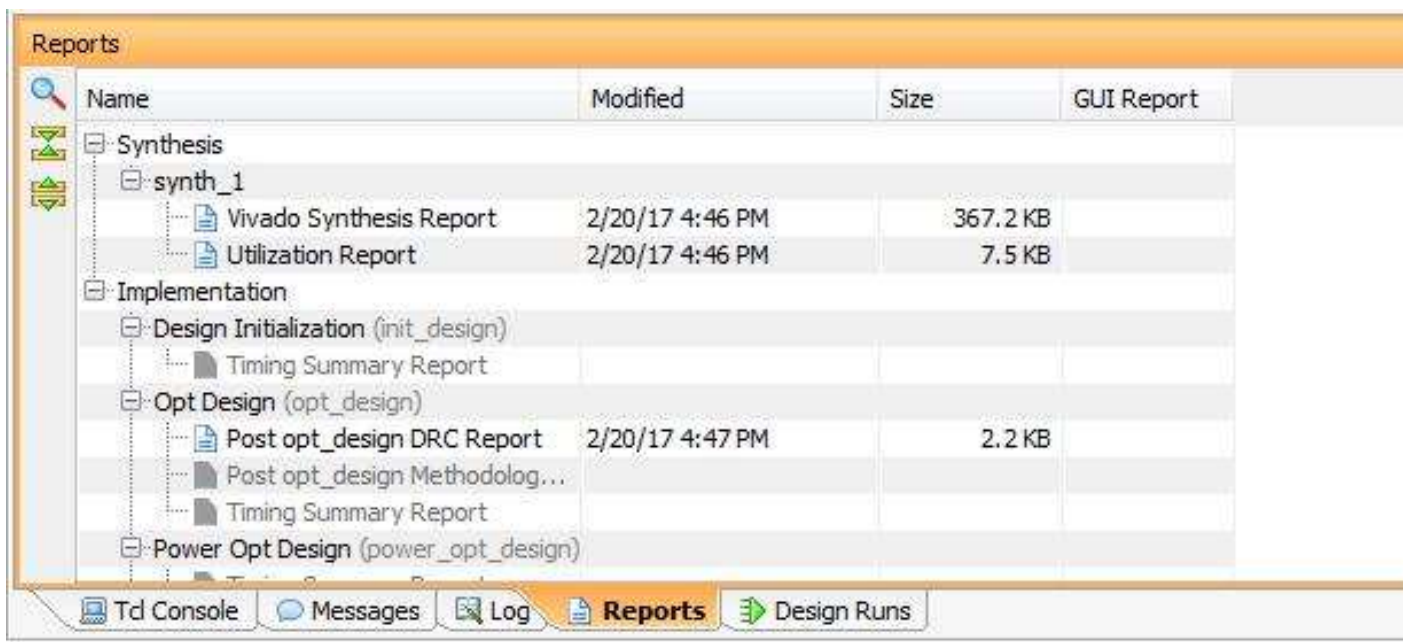


Figure 28: Reports



Once the implementation is done, it is easy to check all sorts of parameters for the design, by selecting the right report in Flow Navigator, **Implementation** section, as in Figure 29.

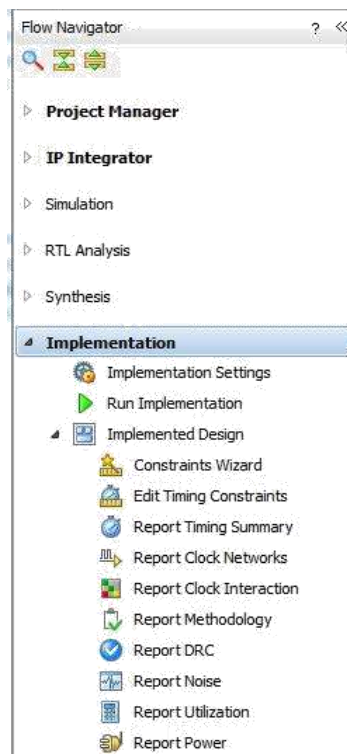


Figure 29: Implementation in Flow Navigator

## 8.1 Timing

You can check, for instance, that the timing constraints have been fulfilled (e.g. the logic can keep up with the specified clock frequency) by selecting the **Report Timing Summary**. This will display detailed information in the lower window, as in Figure 30. Do inspect the different details of the summary.



Figure 30: Timing Summary. All timing constraints are met in this case.



## 8.2 Utilization

You can also get an idea about how much of the FPGA resources your design is using by selecting **Report Utilization**. This will display detailed information in the lower window, as in Figure 31. Do inspect the different details of the summary.

## 8.3 Power

Finally, you can also get an estimate for the power consumption of the design by selecting **Report Power**. This will display detailed information in the lower window, as in Figure 32. Do inspect the different details of the summary.

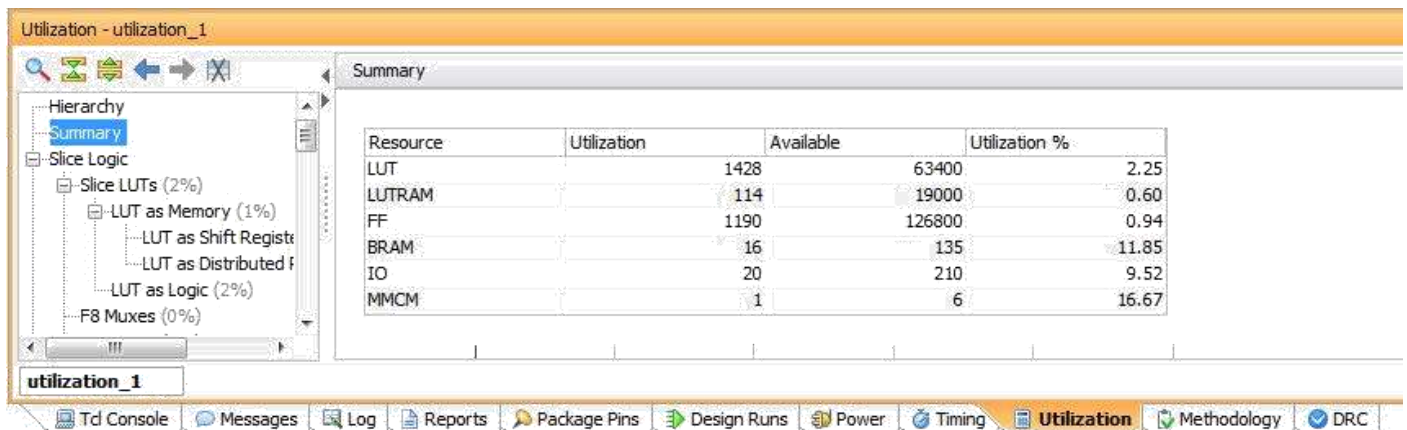


Figure 31: Utilization Summary

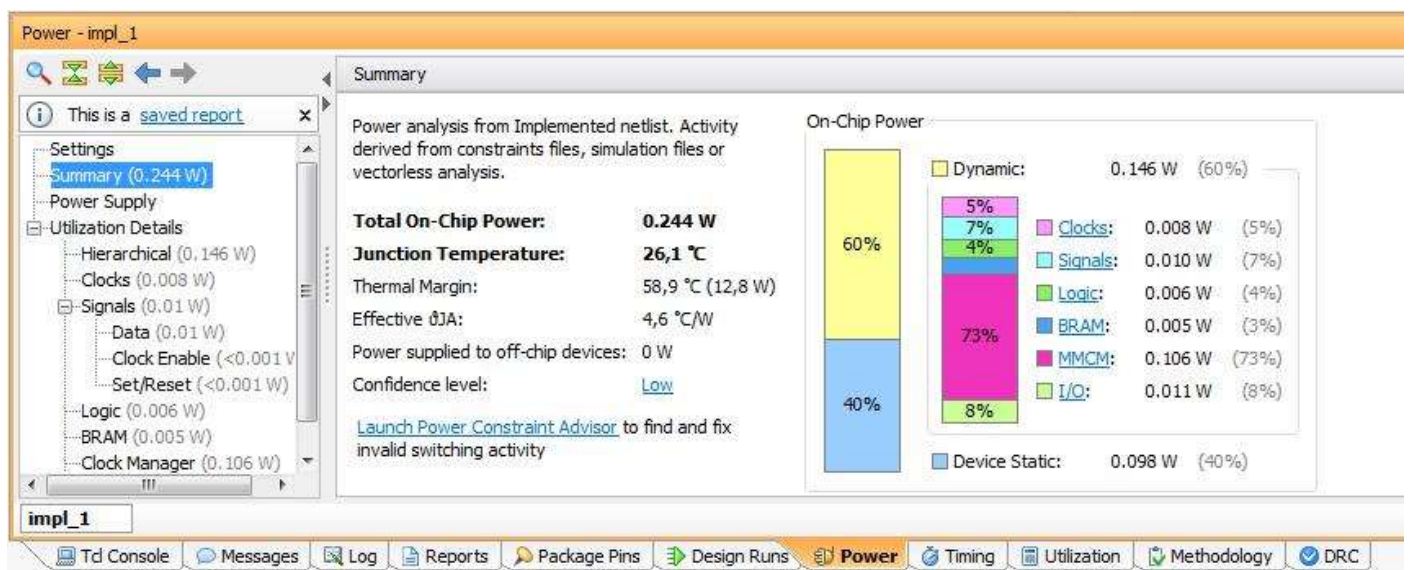


Figure 32: Power Summary

## Creating the Software

Once the bitstream is generated, the hardware architecture is done, but it's a blank slate, with no software to run on it. To create the software, you will use another tool Xilinx SDK, based on Eclipse. For these labs you will work in C, although using C++ is also possible. The compiled software will be merged with the hardware to form the complete FPGA configuration.

## 9 Exporting the Design to SDK

The software development kit needs to know about the underlying hardware architecture. If you just built your hardware, you can export it to SDK.

1. Select **File -> Export -> Export Hardware**.
2. In **Export Hardware** dialog box, select **Include bitstream** check box, shown in Figure 33. Make sure that the **Export to** field is set to <Local to Project>.

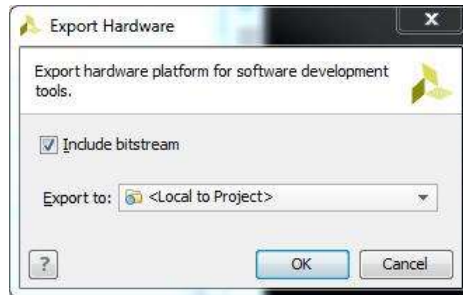


Figure 33: Export Hardware Dialog Box

3. Click **OK**.
4. Select **File -> Launch SDK**. In the Launch SDK dialog box, make sure that both **Exported location** and **Workspace** drop-down lists are set to <Local to Project>.



Figure 34: Launch SDK Dialog Box

5. Click **OK**. SDK launches in a separate window.

Take some time to examine the directory structure and the files residing in your project directory. You will notice a folder ending in **.sdk**, which is an SDK workspace. This also contains one **.hdf** file, the full description of the target hardware architecture.

**note** It is also possible to launch SDK without building a hardware platform, if this is already provided as a **.hdf** file. Then you will need to start SDK from the menu or command line, **Create Application Project** and specify **New... Target Hardware**. A New Hardware Project dialog box opens, where you should **Browse...** after the Target Hardware Specification, namely the **.hdf** file mentioned above.

## 10 Create a "Peripheral Test" Application

As with most of the steps in this tutorial, there are several ways to get the same result, since the same commands can be issued in the user interface in different places. The following shows one of the ways to create a new application.

1. In SDK, right-click **mb-subsystem\_wrapper\_hw\_platform - 0** in the Project Explorer and select **New -> Project**, as shown in Figure 35.

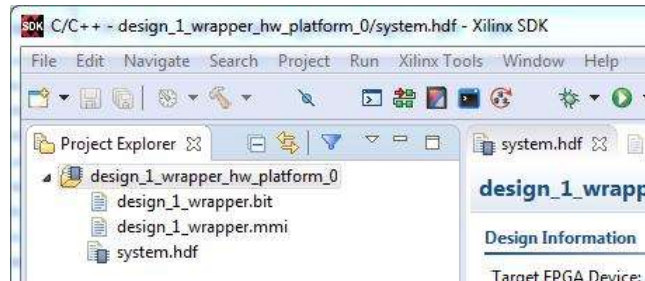


Figure 35: SDK New Project Selection

2. In the New Project dialog box, select **Xilinx Application Project** (Figure 36).

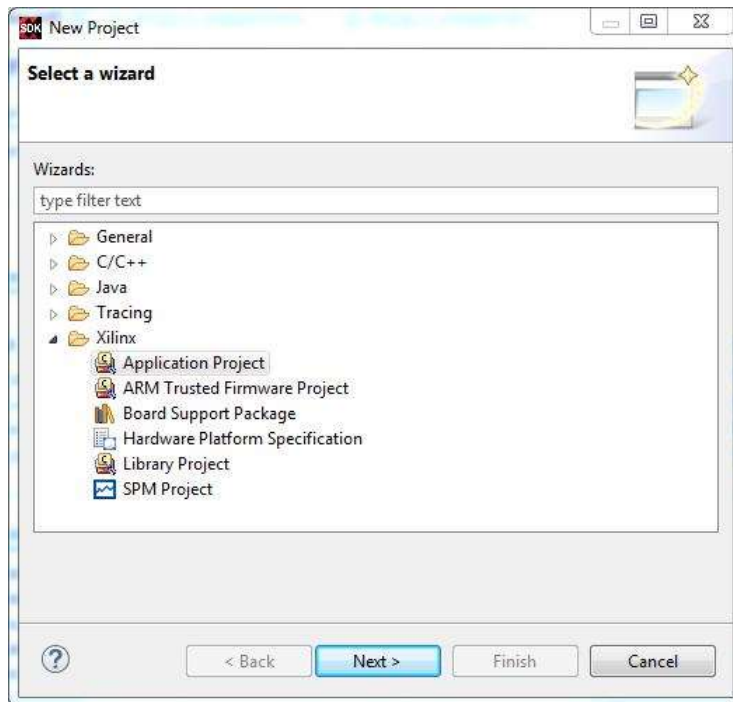


Figure 36: SDK New Project Wizard

3. Click **Next**.
4. Type a name (such as peri test) for your project and choose standalone as OS platform, as shown in Figure 37.
5. Click **Next**.
6. Select the **Peripheral Tests** application template as in Figure 38, and click **Finish**. Note that there are other templates you could choose, providing support for various kinds of applications. It is very common to start with one such application (e.g. "Hello World") and develop it into the application you actually want to implement.

SDK creates a new "peri-test" application (see Figure 39). Note that there are three open projects: one is the hardware description, one is your new application, and the third is the **BSP**, board support package, which contains the libraries needed to program with the IPs in your architecture. Open peri-text bsp, then BSP Documentation, and further tmrctr.v4.2 to examine the documentation for the AXI Timer API. Often this documentation contains example code in the File List. For some works with the AXI Timer, the xtmrctr\_polled example.c is relevant.

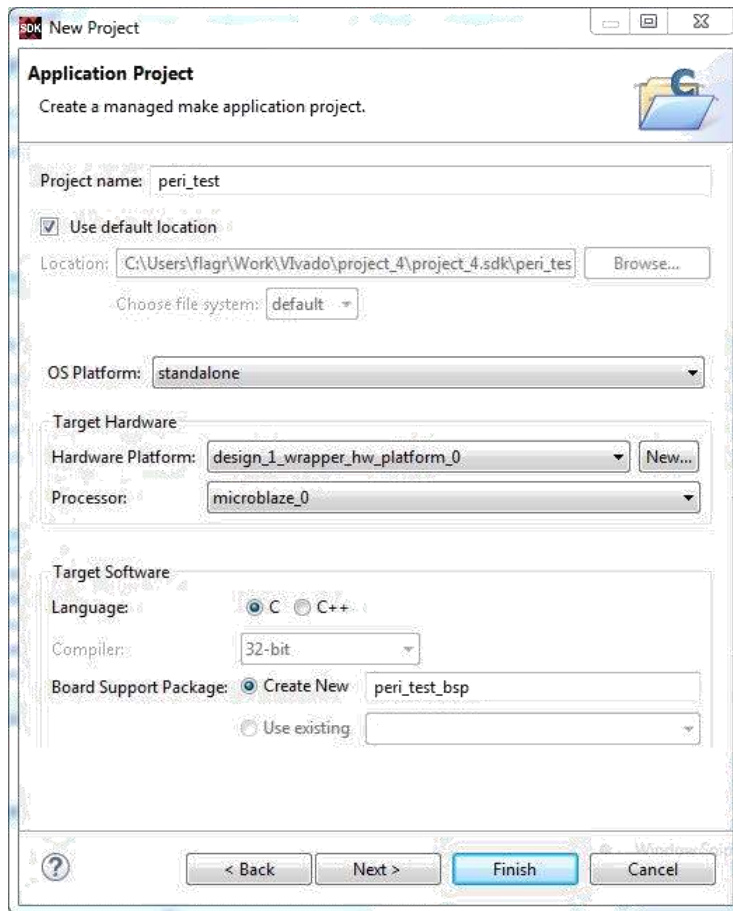


Figure 37: New Project: Application Project Wizard

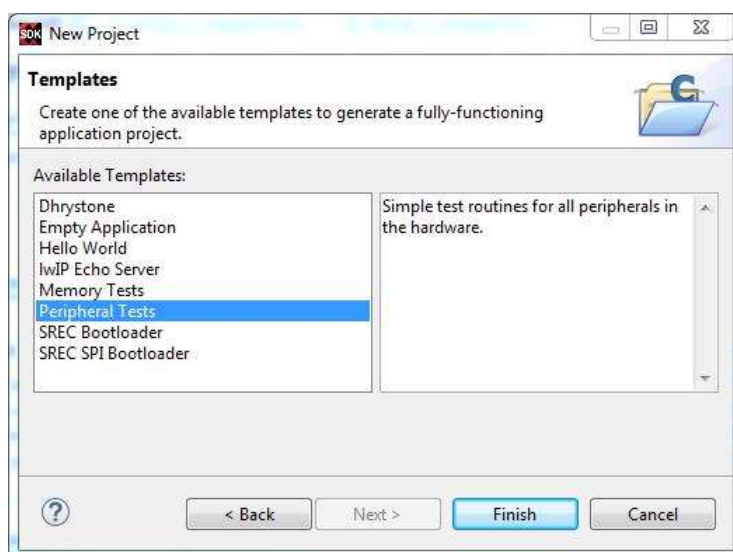


Figure 38: New Project: Template Wizard

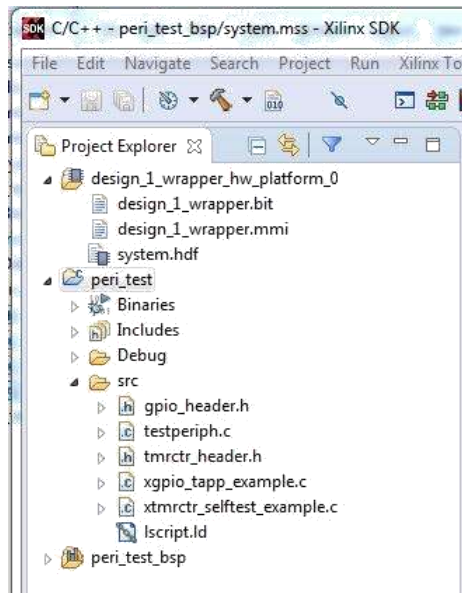


Figure 39: Project Explorer with the new peri\_test application

## 11 Executing the Application on Nexys-4

Finally it is time to run the software on the hardware architecture designed earlier, which is emulated on the FPGA. This means that the FPGA will be configured with the hardware and software you produced in this tutorial.

Make sure that you have connected the target board to the host computer (USB - PROG/UART pin), and the board is turned on.

1. Select **Xilinx Tools -> Program FPGA** to open the Program FPGA dialog box.
2. In the **Program FPGA** dialog box, make sure to change the bootloop (does nothing / used in debugging) in Software Configuration to point to the right executable, as shown in Figure 40. Click **Program**.

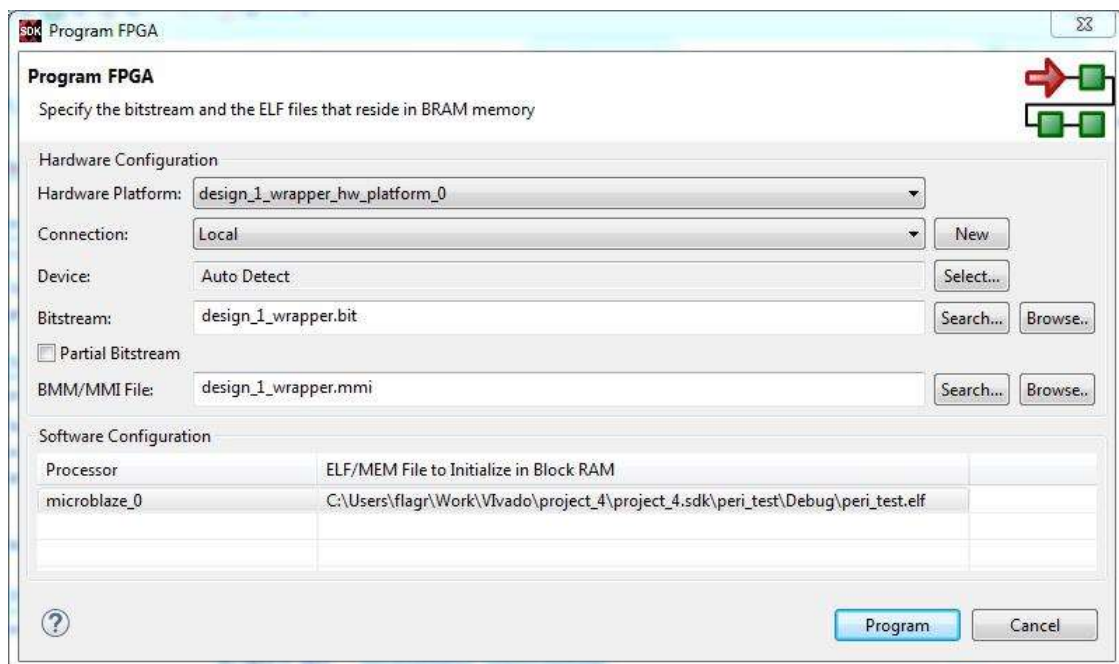


Figure 40: Project Explorer with the new peri test application

3. Although the system is up and running (check the LEDs), there is no way to see printouts unless we connect to the board via a Terminal. This can be done in SDK. Select **SDK Terminal** tab in the lower/Console region, as shown in Figure 41, then click on + to connect to a serial port.

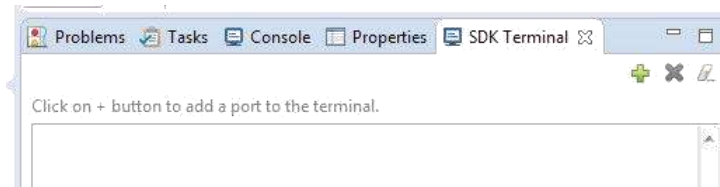


Figure 41: SDK Terminal Tab

4. In the **Connect to serial port** dialog box, select the right COM port (may vary on your system) and make sure the parameters are set up similar to Figure 42. Click **OK**.

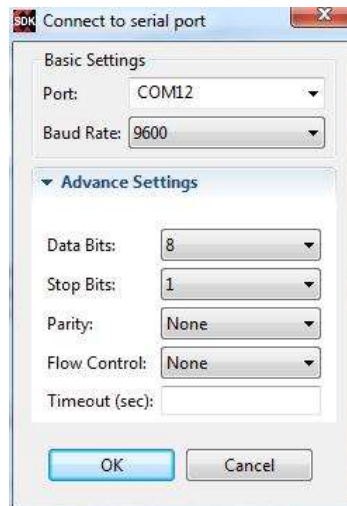


Figure 42: Connect to Serial Port Dialog Box

5. Your terminal window should now receive text from the board, similar to Figure 43. Try pushing the red **CPU RESET** button on the Nexys-4 board and examine both the LEDs and the terminal output.

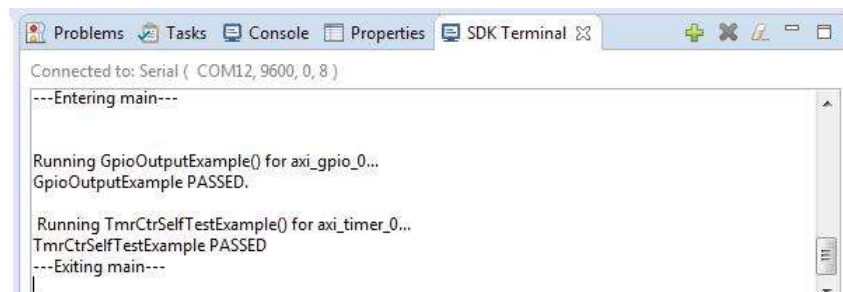


Figure 43: Terminal Output via the Serial Connection

## 12 Summary

At the end of this tutorial you should be able now to:

- create a simple, single MicroBlaze hardware architecture in Vivado
- gather statistics about your design timing, device utilization and power use in Vivado
- export the hardware architecture into SDK
- create simple application using architecture in SDK
- run the full system on the Nexys-4 board

Most of these skills will be used in future assignment, and can be further develop to allow you to be more efficient.