Xilinx Vivado/SDK Tutorial

(Laboratory Session 1, EDAN15)

Flavius.Gruian@cs.lth.se

March 21, 2017

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.

note Please be aware that, as with any complex tool chain, the software we use in the lab may contain bugs as well as discrepancies between different versions. The actions described in this tutorial were carried out for Vivado version 2016.4, so there might be inconsistencies with your current tool chain. With a bit of flexibility, similar actions can be carried out in your lab setup.

The Sample Design

This tutorial[1] 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 an 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.

An 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

[1]This document is loosely based on the *Lab 3: Using the Embedded MicroBlaze Processor* from the Xilinx *Embedded Processor Hardware Design*, UG940 (v2016.3) October 19, 2016. For the readers interested in more details we strongly recommend the UG940 document.

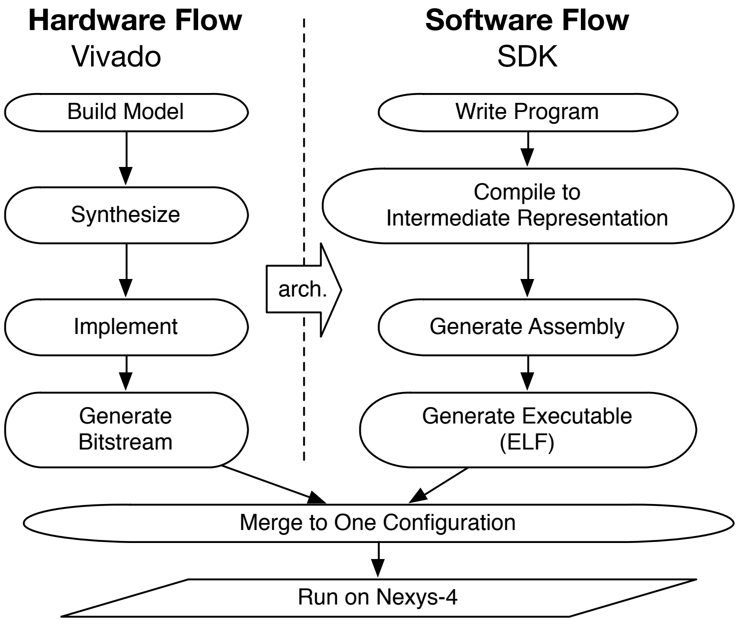


Figure 1: Overview of the Design Flow in this Tutorial (simplistic)

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

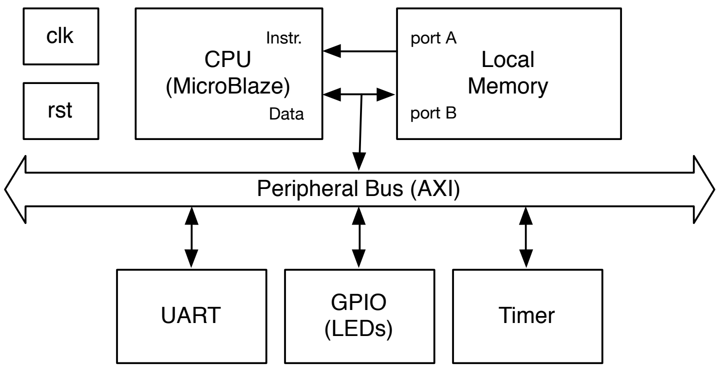


Figure 2: Overview of the Hardware Architecture in this Tutorial

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 (Figure 3), select **Create New Project**. The New Project wizard opens.
2. In the **Project Name** dialog box, type the project name and location.

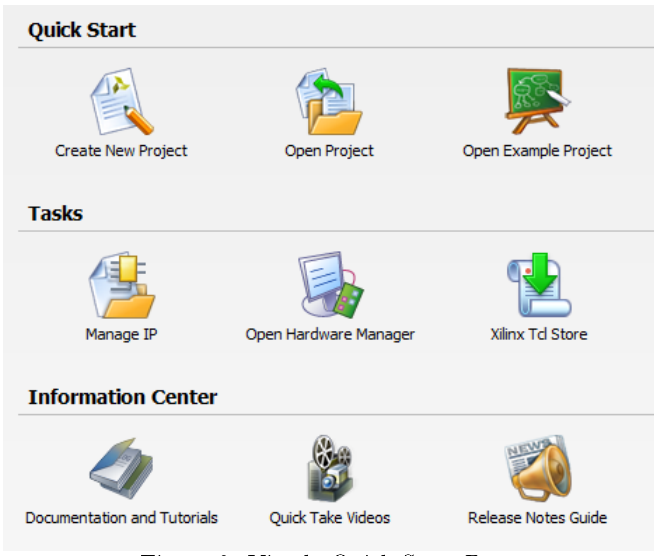


Figure 3: Vivado Quick Start Page

note Note that path cannot contain spaces. Also, due to group policies in our Windows machines, you will only be able to properly work with the project from certain directories. **Make sure to locate all your projects under `C:/Users/user-id/Program/` or you will not be able to run certain steps!** At the end of each lab session you may want to save your project on `H:`, to be able to use it on another machine.

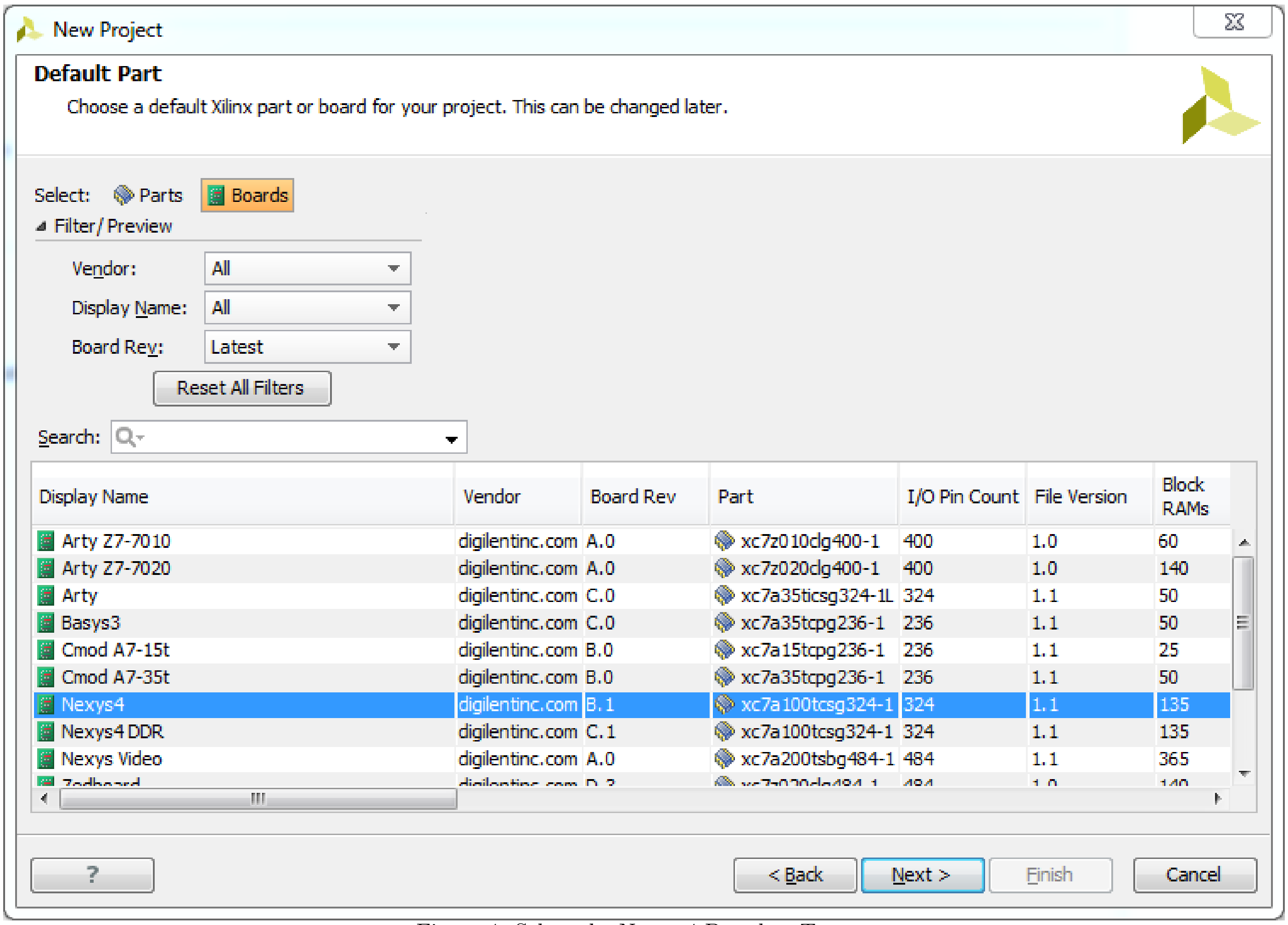
Make sure that **Create subdirectory** is checked. Click **Next**.

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** or **Verilog**. 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**, see Figure 4. 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 5.
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, shown in Figure 5.
5. In the Block Design area, Diagram tab should look like in Figure 6. 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 7, 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.


New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts **Boards**

Filter/ Preview

Vendor: All

Display Name: All

Board Rev: Latest

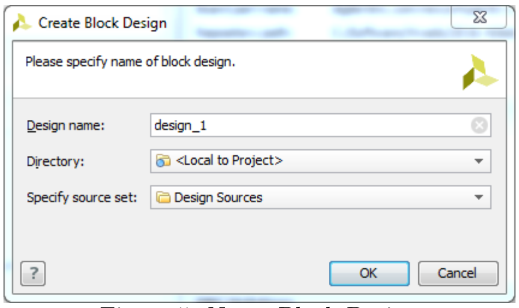
Reset All Filters

Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version	Block RAMs
Arty Z7-7010	digilentinc.com	A.0	xc7z010clg400-1	400	1.0	60
Arty Z7-7020	digilentinc.com	A.0	xc7z020clg400-1	400	1.0	140
Arty	digilentinc.com	C.0	xc7a35ticsg324-1L	324	1.1	50
Basys3	digilentinc.com	C.0	xc7a35tcpg236-1	236	1.1	50
Cmod A7-15t	digilentinc.com	B.0	xc7a15tcpg236-1	236	1.1	25
Cmod A7-35t	digilentinc.com	B.0	xc7a35tcpg236-1	236	1.1	50
Nexys4	digilentinc.com	B.1	xc7a100tcsg324-1	324	1.1	135
Nexys4 DDR	digilentinc.com	C.1	xc7a100tcsg324-1	324	1.1	135
Nexys Video	digilentinc.com	A.0	xc7a200tsbg484-1	484	1.1	365
Zedboard	digilentinc.com	D.3	xc7z020clg484-1	484	1.0	140

? | < Back | Next > | Finish | Cancel

Figure 4: Select the Nexys-4 Board as Target.


Create Block Design

Please specify name of block design.

Design name: design_1

Directory: <Local to Project>

Specify source set: Design Sources

? | OK | Cancel

Figure 5: Name Block Design

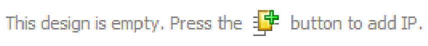
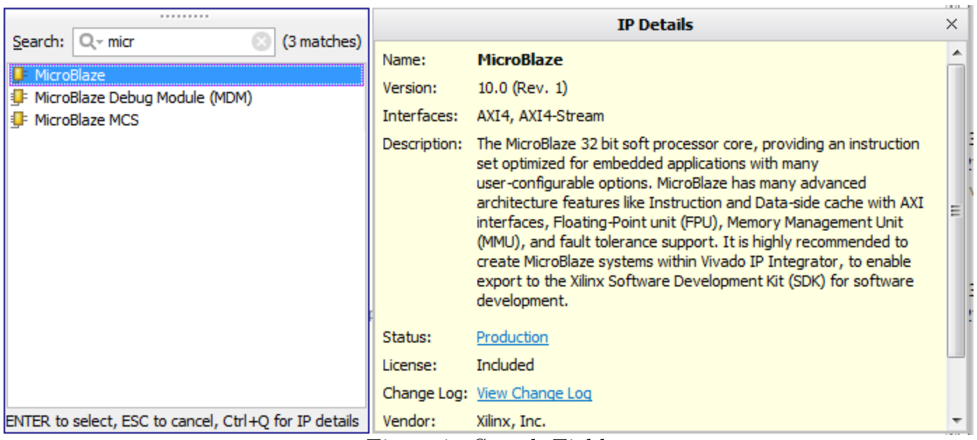

This design is empty. Press the button to add IP.

Figure 6: Add IP


Search: micr (3 matches)

- MicroBlaze
- MicroBlaze Debug Module (MDM)
- MicroBlaze MCS

ENTER to select, ESC to cancel, Ctrl+Q for IP details

IP Details

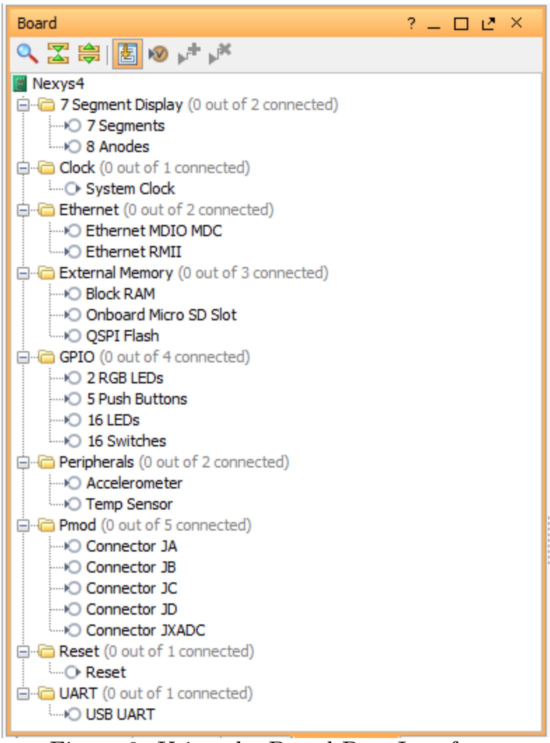
Name:	**MicroBlaze**
Version:	10.0 (Rev. 1)
Interfaces:	AXI4, AXI4-Stream
Description:	The MicroBlaze 32 bit soft processor core, providing an instruction set optimized for embedded applications with many user-configurable options. MicroBlaze has many advanced architecture features like Instruction and Data-side cache with AXI interfaces, Floating-Point unit (FPU), Memory Management Unit (MMU), and fault tolerance support. It is highly recommended to create MicroBlaze systems within Vivado IP Integrator, to enable export to the Xilinx Software Development Kit (SDK) for software development.
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.

Figure 7: 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 8 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.

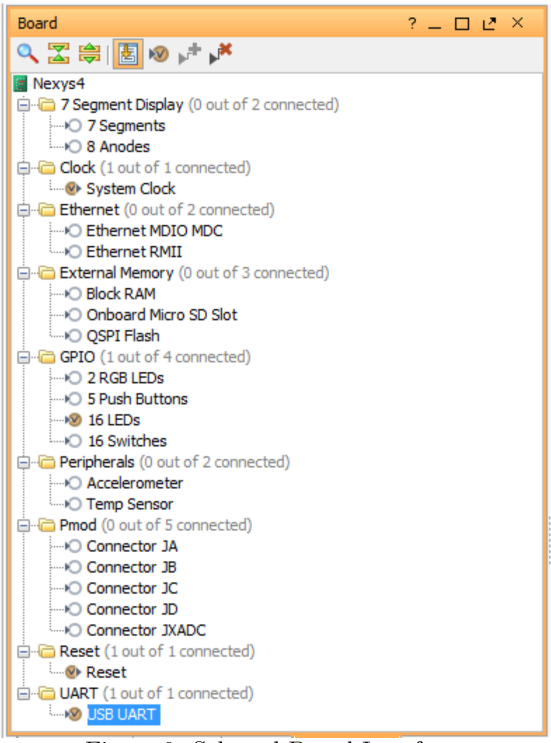

```
Board
Nexys4
  7 Segment Display (0 out of 2 connected)
    7 Segments
    8 Anodes
  Clock (0 out of 1 connected)
    System Clock
  Ethernet (0 out of 2 connected)
    Ethernet MDIO MDC
    Ethernet RMII
  External Memory (0 out of 3 connected)
    Block RAM
    Onboard Micro SD Slot
    QSPI Flash
  GPIO (0 out of 4 connected)
    2 RGB LEDs
    5 Push Buttons
    16 LEDs
    16 Switches
  Peripherals (0 out of 2 connected)
    Accelerometer
    Temp Sensor
  Pmod (0 out of 5 connected)
    Connector JA
    Connector JB
    Connector JC
    Connector JD
    Connector JXADC
  Reset (0 out of 1 connected)
    Reset
  UART (0 out of 1 connected)
    USB UART
```
Figure 8: 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 **Reset** and drag and drop it into the block design canvas. This will allow you to use the Reset push button on the board to reset your system. Click **OK** in the Auto Connect dialog box.

5. From the Board window, select **USB UART** and drag and drop it into the block design canvas (see Figure 9). 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.



```
Board
Nexys4
  7 Segment Display (0 out of 2 connected)
    7 Segments
    8 Anodes
  Clock (1 out of 1 connected)
    System Clock
  Ethernet (0 out of 2 connected)
    Ethernet MDIO MDC
    Ethernet RMII
  External Memory (0 out of 3 connected)
    Block RAM
    Onboard Micro SD Slot
    QSPI Flash
  GPIO (1 out of 4 connected)
    2 RGB LEDs
    5 Push Buttons
    16 LEDs
    16 Switches
  Peripherals (0 out of 2 connected)
    Accelerometer
    Temp Sensor
  Pmod (0 out of 5 connected)
    Connector JA
    Connector JB
    Connector JC
    Connector JD
    Connector JXADC
  Reset (1 out of 1 connected)
    Reset
  UART (1 out of 1 connected)
    USB UART
```

Figure 9: Selected Board Interfaces

6. The block design should now look like in Figure 10.

Run Block Automation

1. Click Run Automation, displayed in Figure 11.

2. The Block Automation dialog box opens as shown in Figure 12. 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 UG940 document *Lab 3* 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)`.

3. Click **OK**.

This generates a basic **MicroBlaze** system in the IP Integrator diagram as shown in Figure 13.

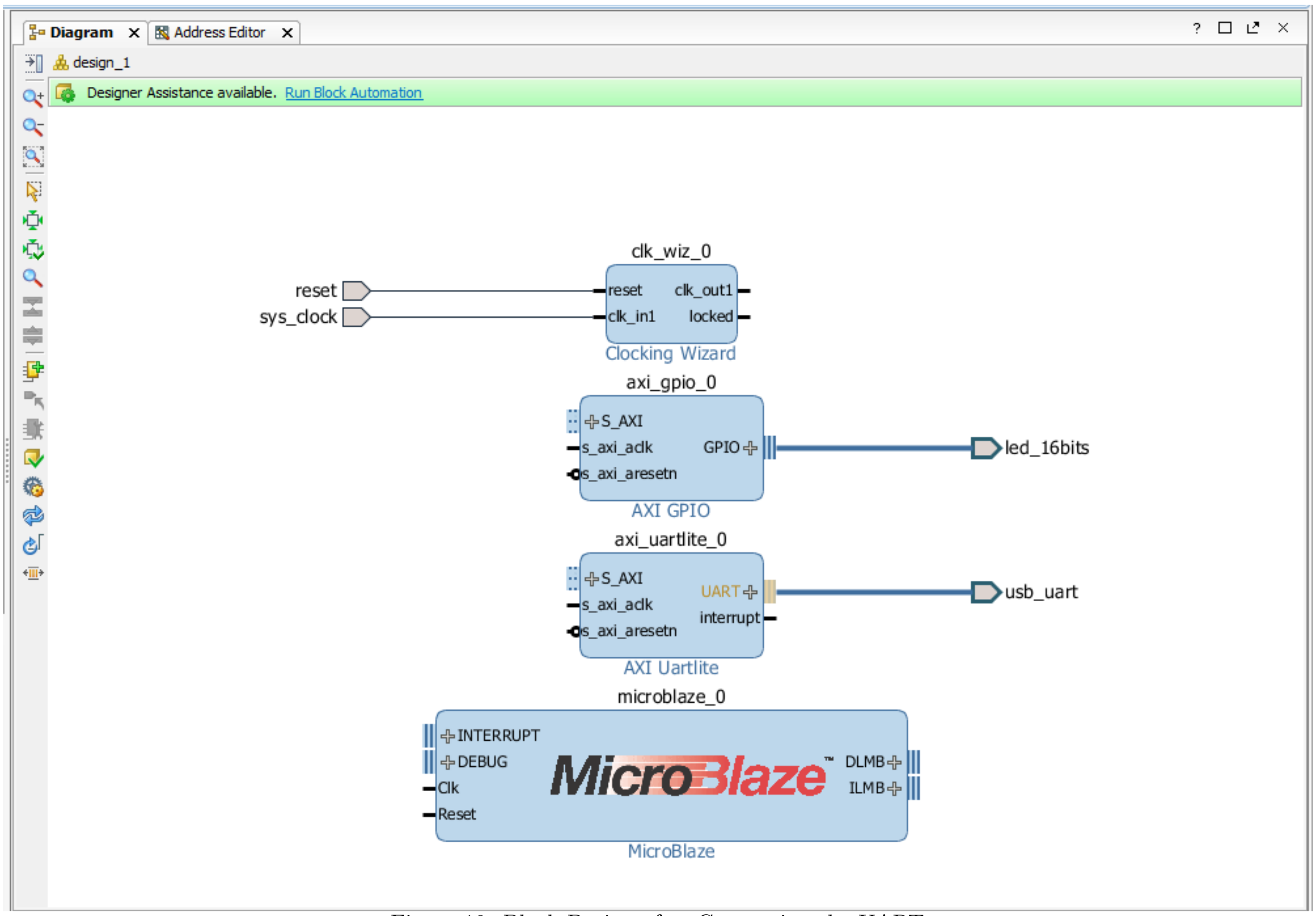


Figure 10: Block Design after Connecting the UART

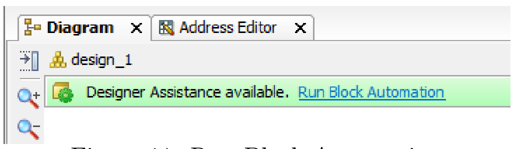
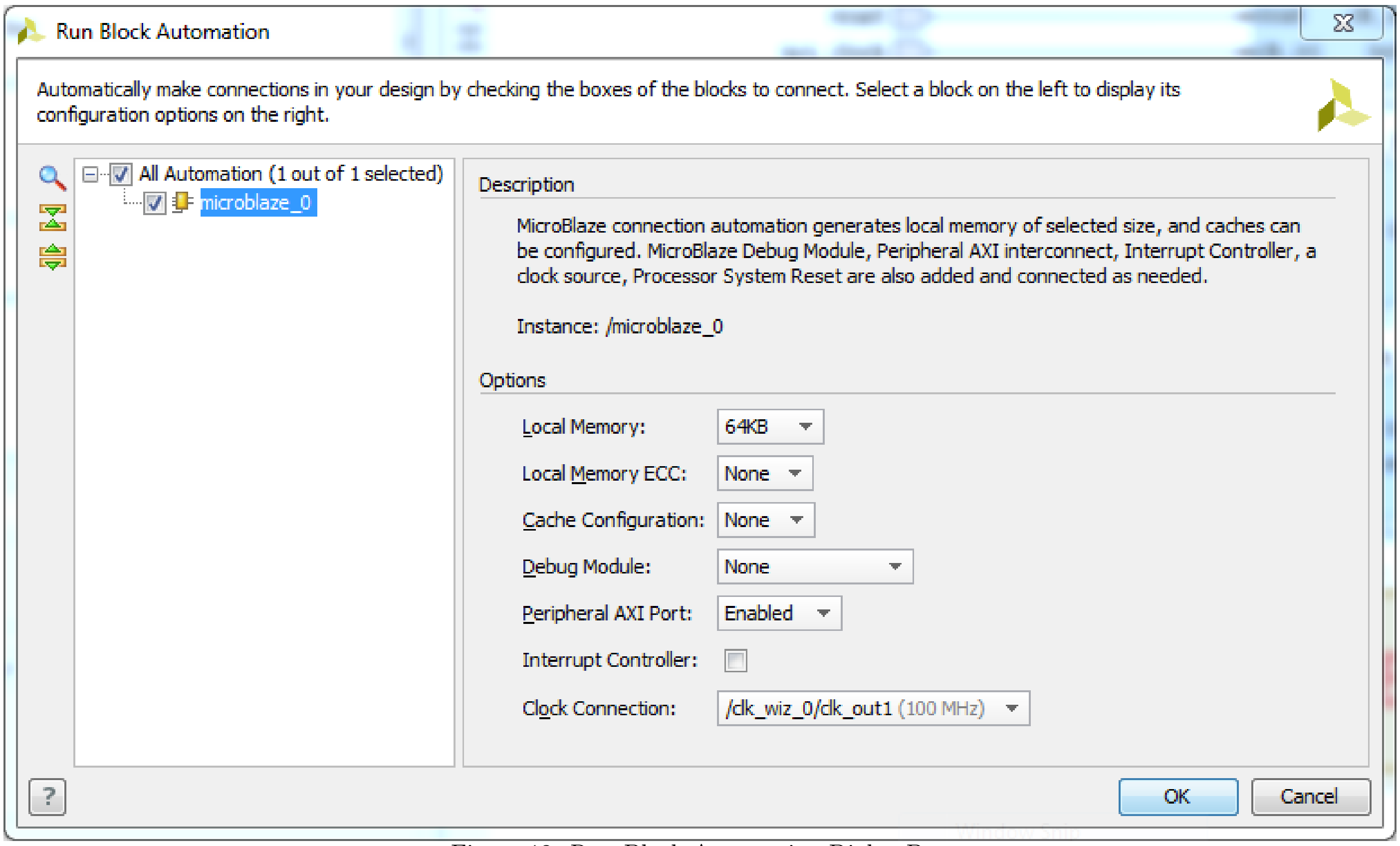


Figure 11: Run Block Automation


Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

☑ All Automation (1 out of 1 selected)
☑ microblaze_0

Description

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze_0

Options

Local Memory:	64KB
Local Memory ECC:	None
Cache Configuration:	None
Debug Module:	None
Peripheral AXI Port:	Enabled
Interrupt Controller:	☐
Clock Connection:	/clk_wiz_0/clk_out1 (100 MHz)

Figure 12: Run Block Automation Dialog Box

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 14.
2. The Run Connection Automation dialog box opens, as in Figure 15. 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.

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, as in the text referring to Figure 6. Let us add a timer, which will be needed to record the performance of our algorithms in the later labs.

1. Select **Add IP** button in the left side of the Diagram tab, Block Design area.
2. As shown in Figure 16, 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 17. The relative placement of your IPs might be slightly different.

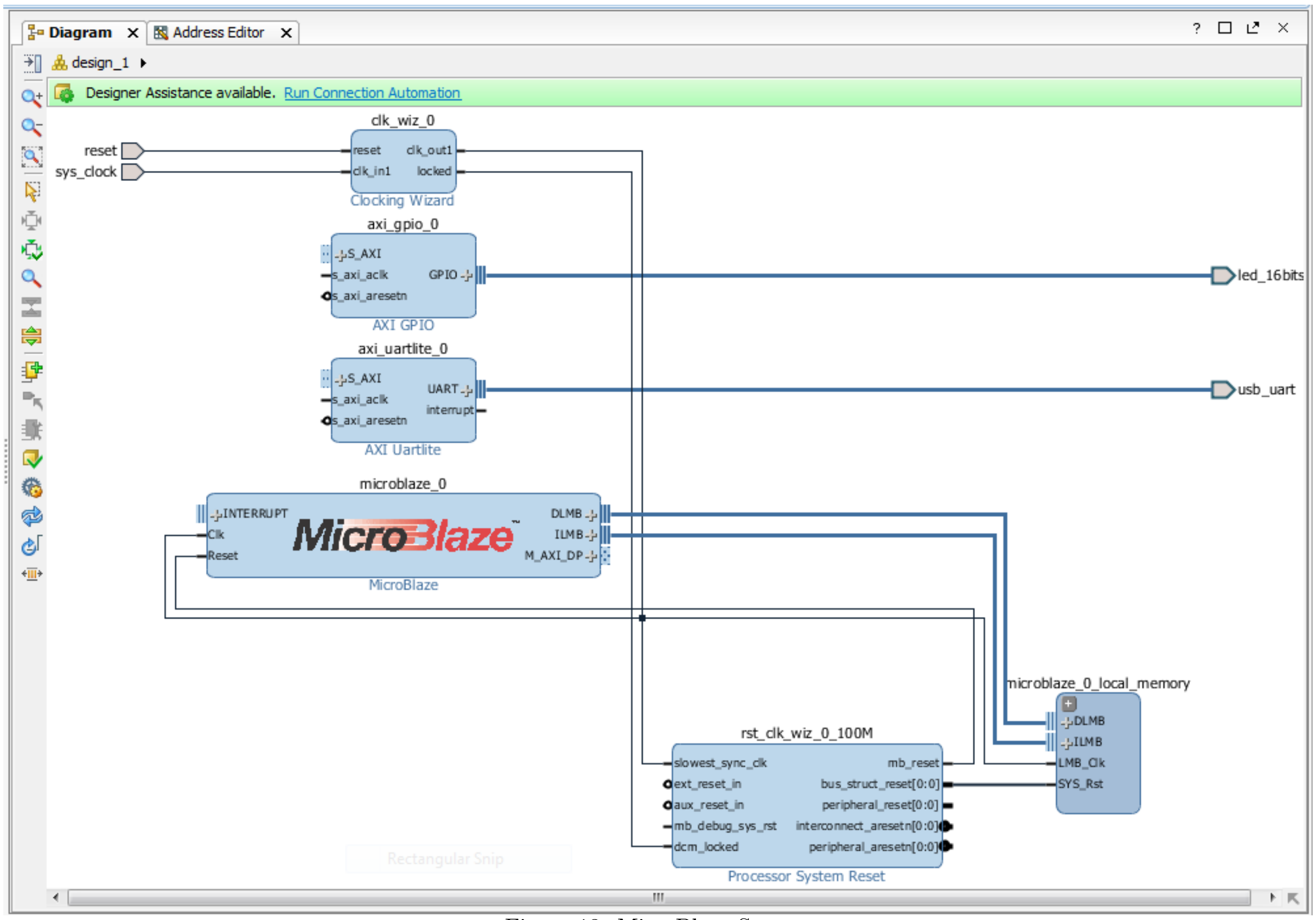


Figure 13: MicroBlaze System

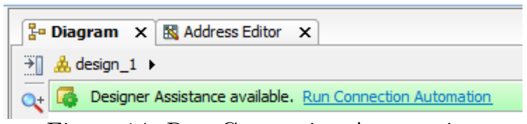
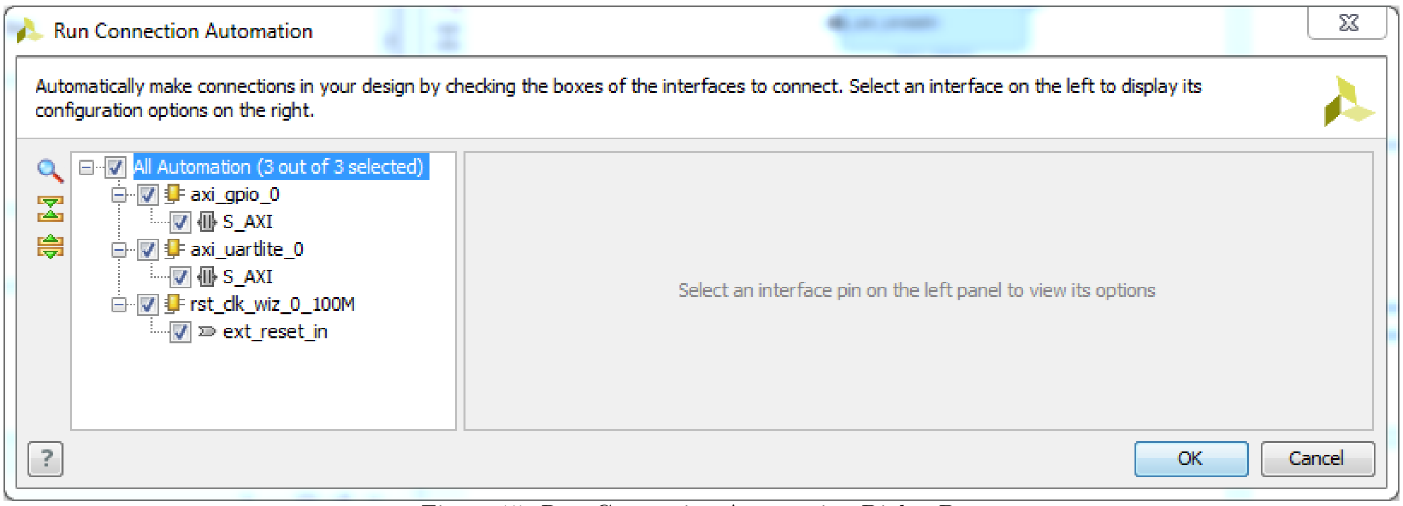
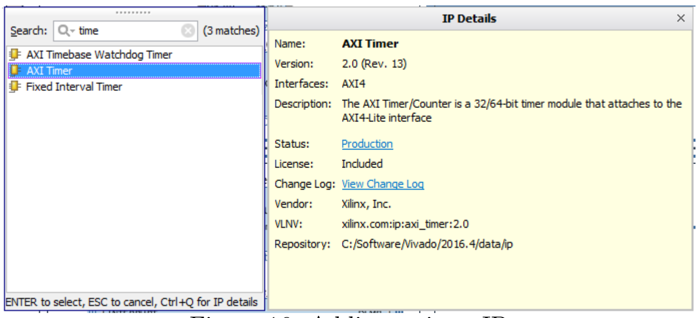


Figure 14: Run Connection Automation


Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

All Automation (3 out of 3 selected)
axi_gpio_0
S_AXI
axi_uartlite_0
S_AXI
rst_clk_wiz_0_100M
ext_reset_in
Select an interface pin on the left panel to view its options
OK
Cancel
Figure 15: Run Connection Automation Dialog Box



Search: time (3 matches)

AXI Timebase Watchdog Timer
AXI Timer
Fixed Interval Timer

ENTER to select, ESC to cancel, Ctrl+Q for IP details

IP Details

Name:	**AXI Timer**
Version:	2.0 (Rev. 13)
Interfaces:	AXI4
Description:	The AXI Timer/Counter is a 32/64-bit timer module that attaches to the AXI4-Lite interface
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.
VLNV:	xilinx.com:ip:axi_timer:2.0
Repository:	C:/Software/Vivado/2016.4/data/ip

Figure 16: Adding a timer IP

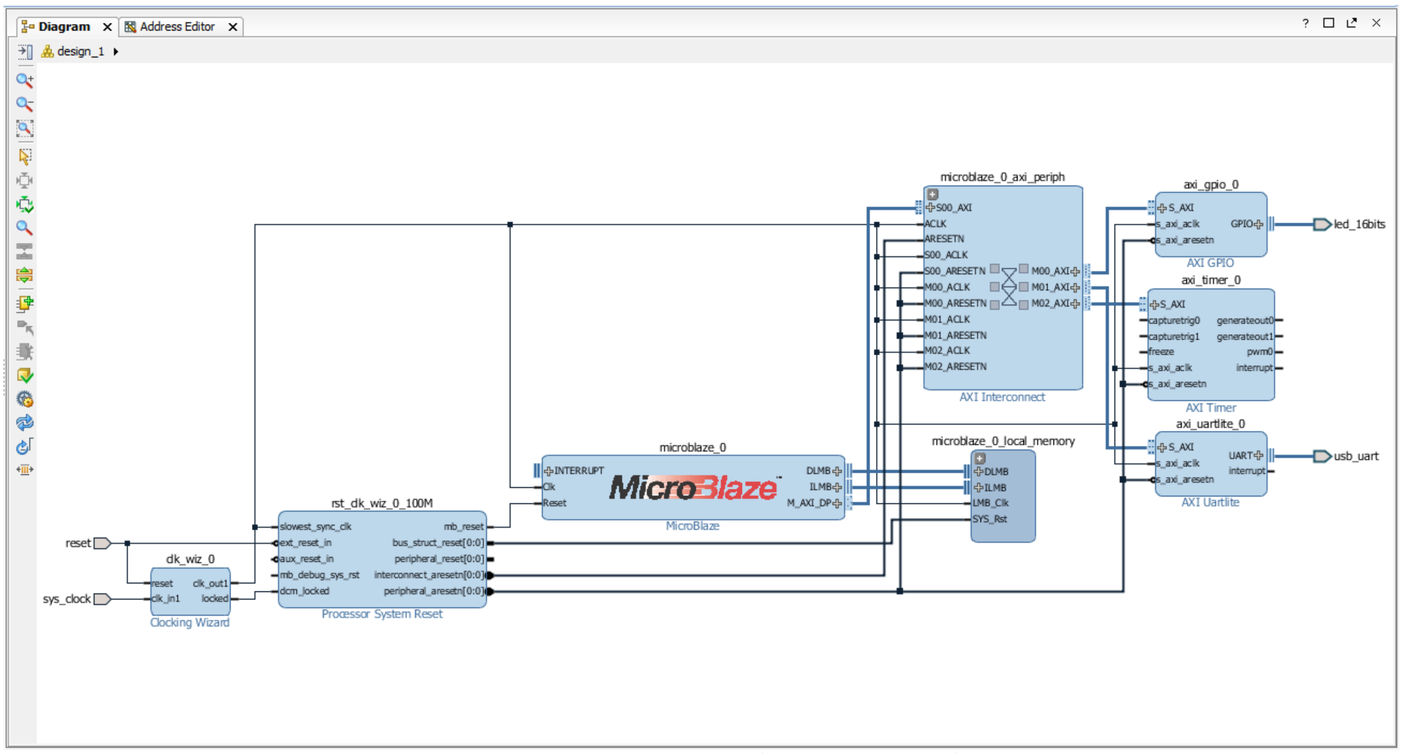


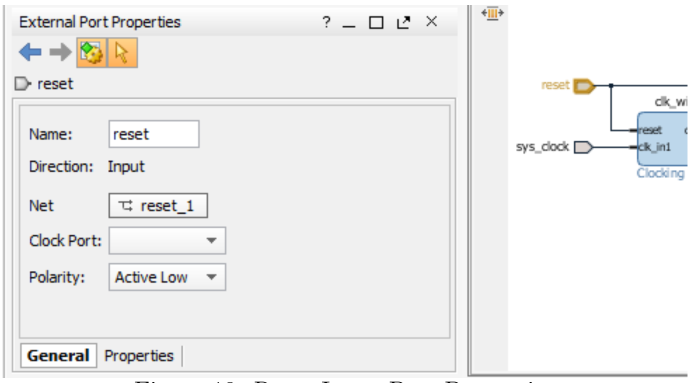
Figure 17: MicroBlaze Connected to UART, GPIO, and AXI Timer

Inspecting and Manually Changing a Design

There are ways to individually alter connections and IPs without appealing to the automatic Vivado features for connections and block automation. We will illustrate some of these capabilities next.

In particular there is a design flaw in our system, which will be identified later on during synthesis and generation of the FPGA configuration by the design rules checking (DRC), but is not obvious at this point. In particular, the Reset input for the system comes from a button, meaning it's driven *low* (logic 0) when pushed. However, as it is right now the clock logic requires a reset active *high* (logic 1). 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 18. Notice that the **Polarity** is *Active Low*.


External Port Properties

reset

Name: reset
Direction: Input
Net: reset_1
Clock Port:
Polarity: Active Low

Figure 18: Reset Input Port Properties

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 19. The parameters for all IPs can be inspected and changed via this dialog box, invoked in a similar way.
3. In the **Clocking Output** tab, notice the *Reset Type* section, which is Active High. Change this to **Active Low**, as shown in Figure 20.
4. Click **OK**. Notice the change in the depiction of the Clocking Wizard – the reset port is now called **resetn** and has a circle at the input (see Figure 21). Both of these are conventions indicating that the signal is active **low** (when 0).
5. Notice that the reset input port and the Clocking Wizard input are no longer connected. Connect the via the mouse by dragging a line from the reset input port to the input of the Clocking Wizard. Once connected your design should look like in Figure 22.

The hardware architecture is now complete.

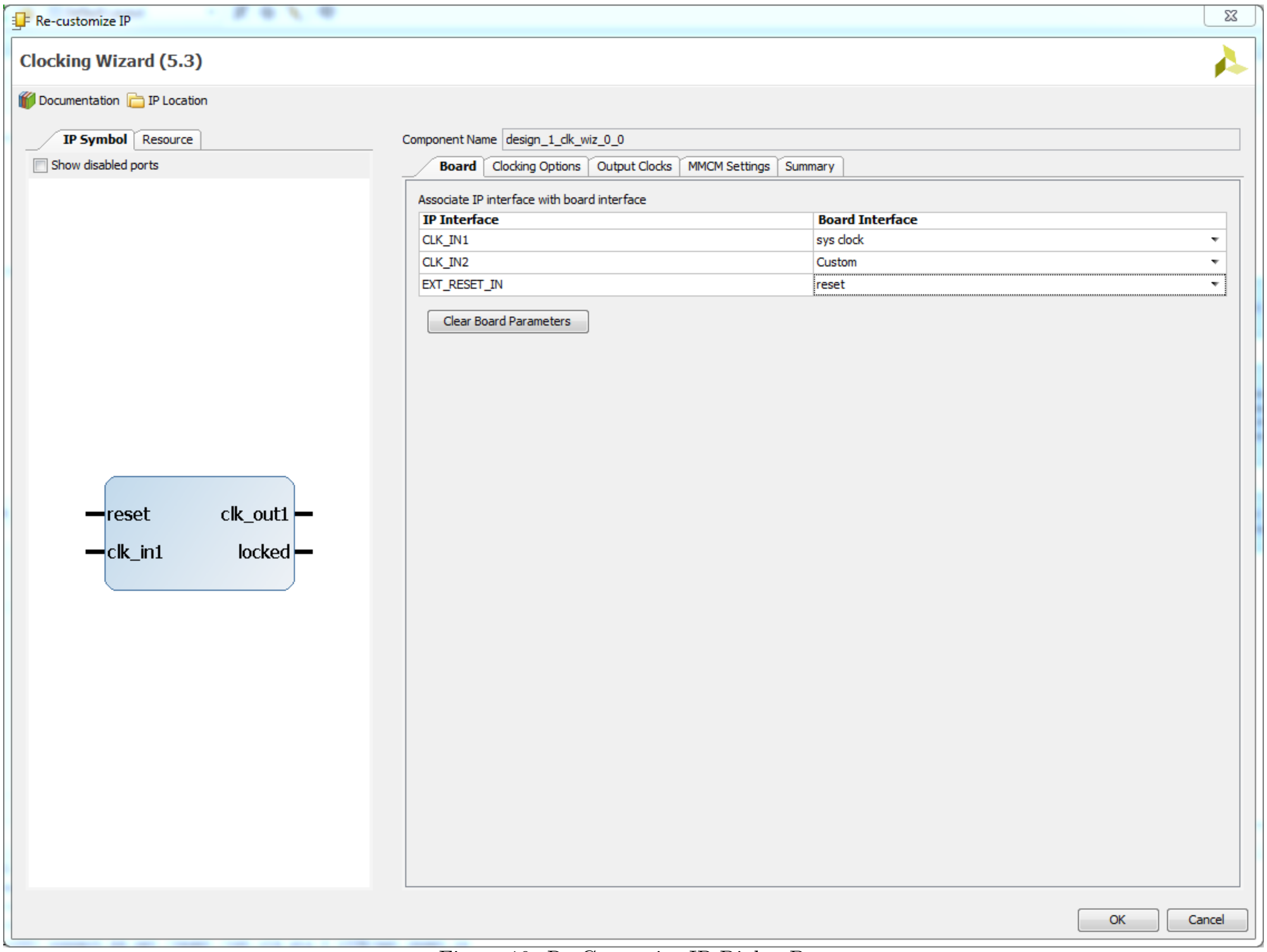
3 Memory-Mapping the Peripherals in IP Integrator

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 23. 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**.


Re-customize IP

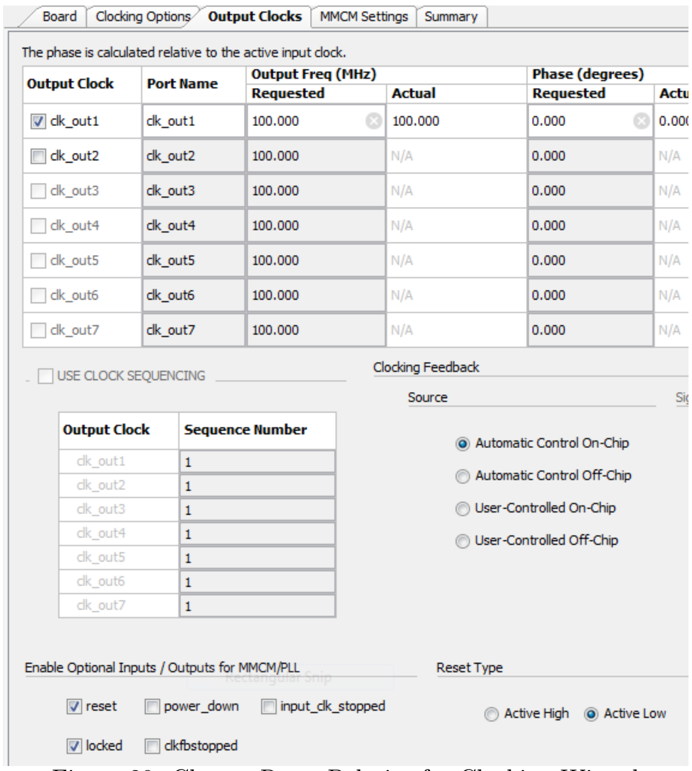
Clocking Wizard (5.3)

Component Name design_1_clk_wiz_0_0

Associate IP interface with board interface

IP Interface	Board Interface
CLK_IN1	sys clock
CLK_IN2	Custom
EXT_RESET_IN	reset

Figure 19: Re-Customize IP Dialog Box



Board | Clocking Options | **Output Clocks** | MMCM Settings | Summary

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)	
		Requested	Actual	Requested	Actu
✓ clk_out1	clk_out1	100.000	100.000	0.000	0.00
clk_out2	clk_out2	100.000	N/A	0.000	N/A
clk_out3	clk_out3	100.000	N/A	0.000	N/A
clk_out4	clk_out4	100.000	N/A	0.000	N/A
clk_out5	clk_out5	100.000	N/A	0.000	N/A
clk_out6	clk_out6	100.000	N/A	0.000	N/A
clk_out7	clk_out7	100.000	N/A	0.000	N/A

USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source

[✓ Automatic Control On-Chip]
Automatic Control Off-Chip
User-Controlled On-Chip
User-Controlled Off-Chip

Enable Optional Inputs / Outputs for MMCM/PLL

[✓ reset] power_down input_clk_stopped
[✓ locked] clkfbstopped

Reset Type

Active High [✓ Active Low]

Figure 20: Change Reset Polarity for Clocking Wizard

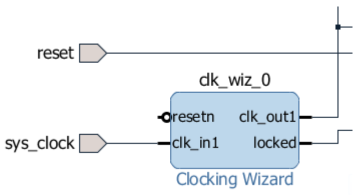


Figure 21: Clocking Wizard Now with Active Low Reset

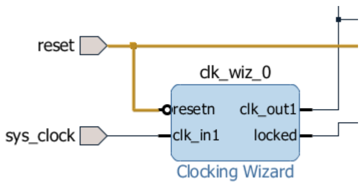


Figure 22: Clocking Wizard Resetn Connected to the Reset Input Port

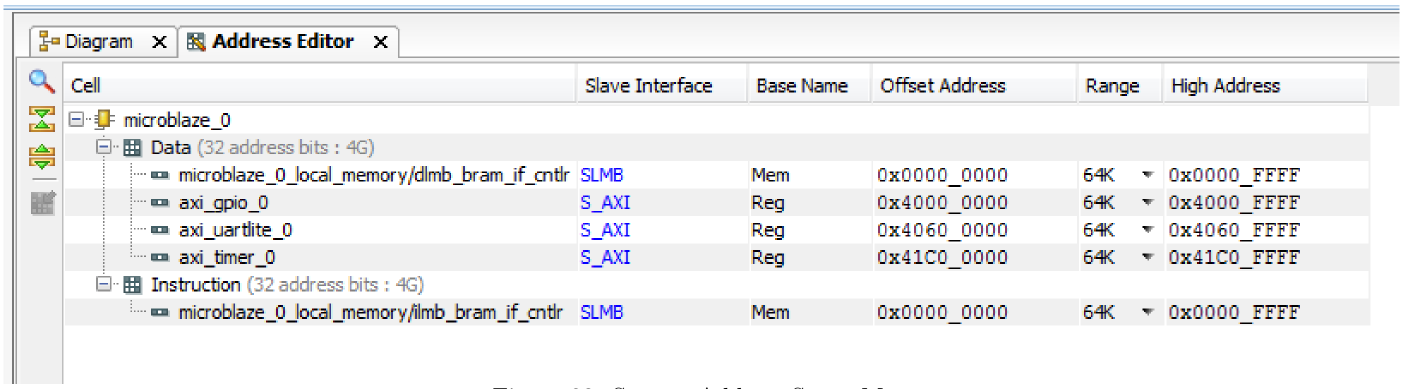


Diagram | **Address Editor**

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_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 23: System Address Space Map

5 Generate Output Products

1. In the Sources window (see Figure 24), 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 25).

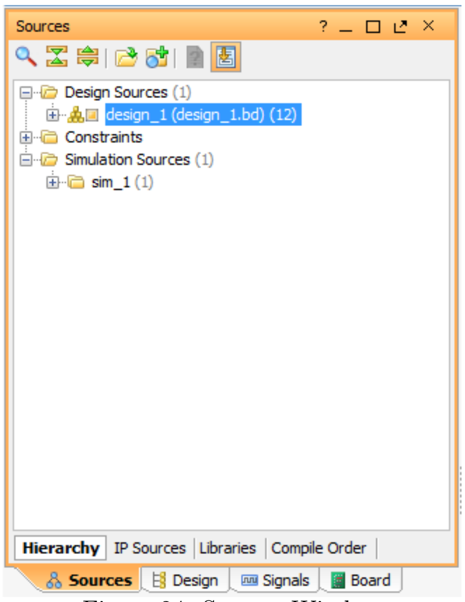


Figure 24: Sources Window

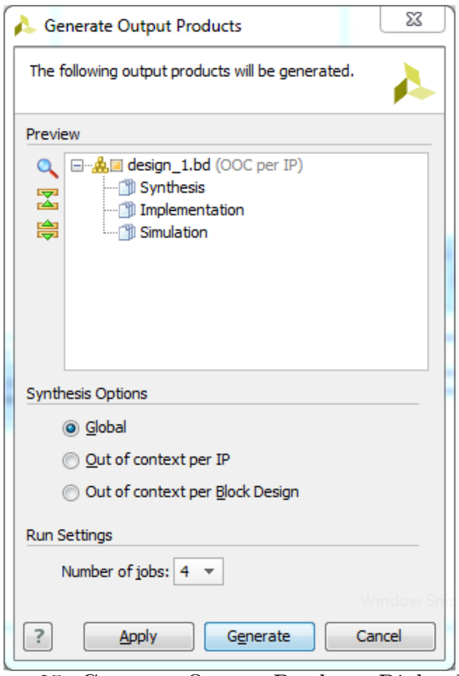


Figure 25: Generate Output Products Dialog Box

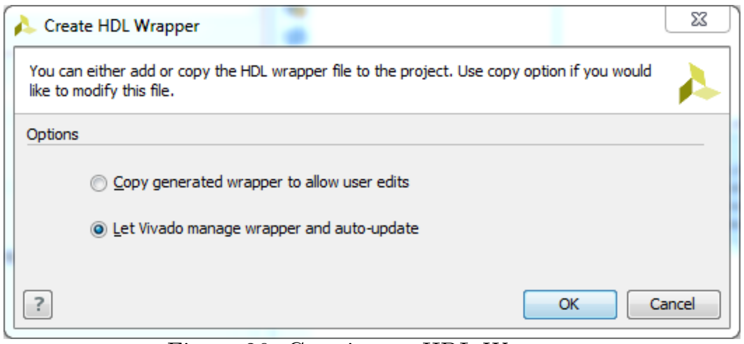
2. Select **Global** in Synthesis Options, and click **Generate**.

3. Click **OK** in the 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 26. Click **OK**.



Create HDL Wrapper

You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

Options

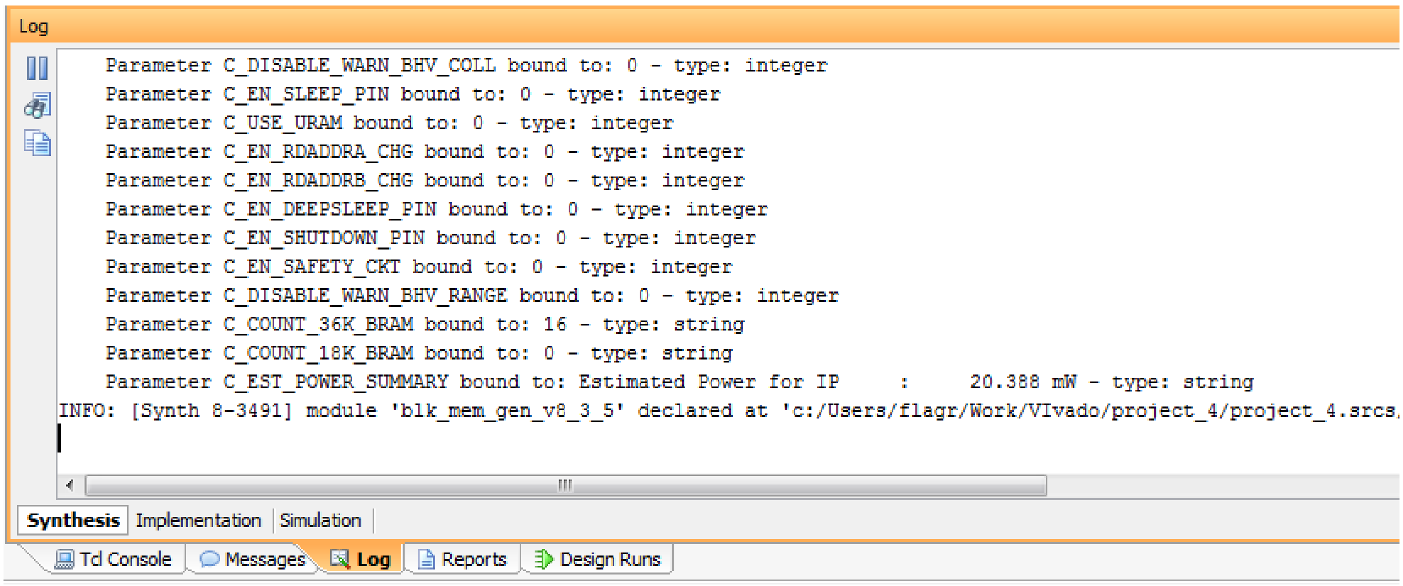
( ) Copy generated wrapper to allow user edits

(●) Let Vivado manage wrapper and auto-update

Figure 26: 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 27).



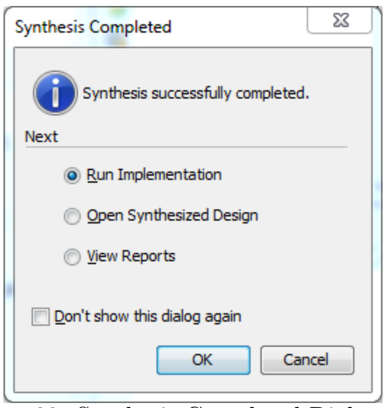
```
    Parameter C_DISABLE_WARN_BHV_COLL bound to: 0 - type: integer
    Parameter C_EN_SLEEP_PIN bound to: 0 - type: integer
    Parameter C_USE_URAM bound to: 0 - type: integer
    Parameter C_EN_RDADDRA_CHG bound to: 0 - type: integer
    Parameter C_EN_RDADDRB_CHG bound to: 0 - type: integer
    Parameter C_EN_DEEPSLEEP_PIN bound to: 0 - type: integer
    Parameter C_EN_SHUTDOWN_PIN bound to: 0 - type: integer
    Parameter C_EN_SAFETY_CKT bound to: 0 - type: integer
    Parameter C_DISABLE_WARN_BHV_RANGE bound to: 0 - type: integer
    Parameter C_COUNT_36K_BRAM bound to: 16 - type: string
    Parameter C_COUNT_18K_BRAM bound to: 0 - type: string
    Parameter C_EST_POWER_SUMMARY bound to: Estimated Power for IP     :     20.388 mW - type: string
INFO: [Synth 8-3491] module 'blk_mem_gen_v8_3_5' declared at 'c:/Users/flagr/Work/VIvado/project_4/project_4.srcs,
```

Figure 27: Following the Synthesis Log

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 28). 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**.

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 29. 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


Synthesis Completed

Synthesis successfully completed.

Next

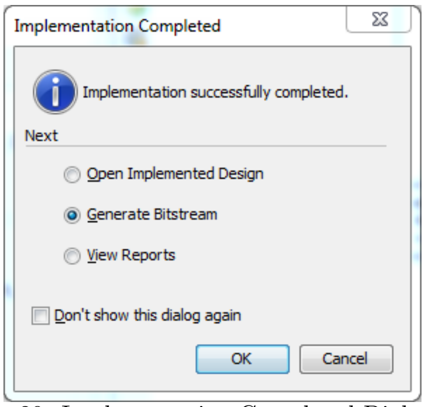
[● Run Implementation]

○ Open Synthesized Design

○ View Reports

☐ Don't show this dialog again

Figure 28: Synthesis Completed Dialog Box


Implementation Completed

Implementation successfully completed.

Next

○ Open Implemented Design

[● Generate Bitstream]

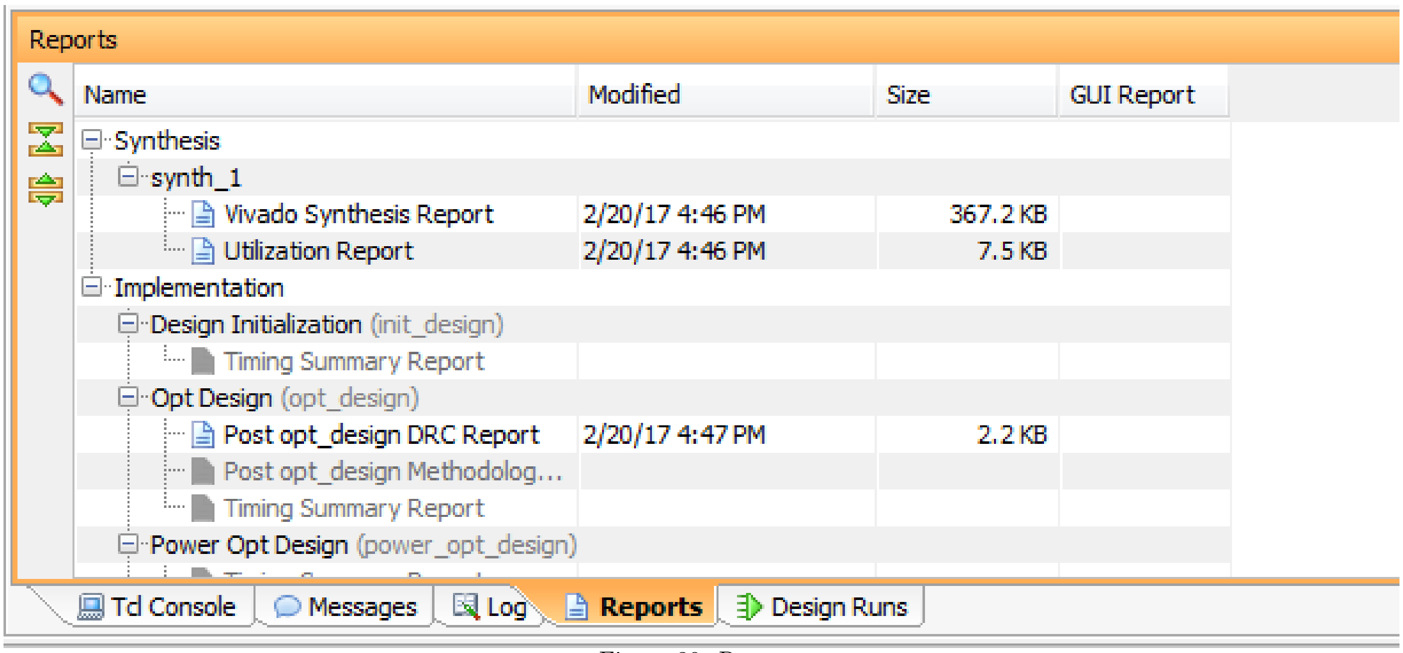
○ View Reports

☐ Don't show this dialog again

Figure 29: Implementation Completed Dialog Box

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 30.


Reports

Name	Modified	Size	GUI Report
Synthesis			
synth_1			
Vivado Synthesis Report	2/20/17 4:46 PM	367.2 KB	
Utilization Report	2/20/17 4:46 PM	7.5 KB	
Implementation			
Design Initialization (init_design)			
Timing Summary Report			
Opt Design (opt_design)			
Post opt_design DRC Report	2/20/17 4:47 PM	2.2 KB	
Post opt_design Methodolog...			
Timing Summary Report			
Power Opt Design (power_opt_design)			

Figure 30: 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 31.

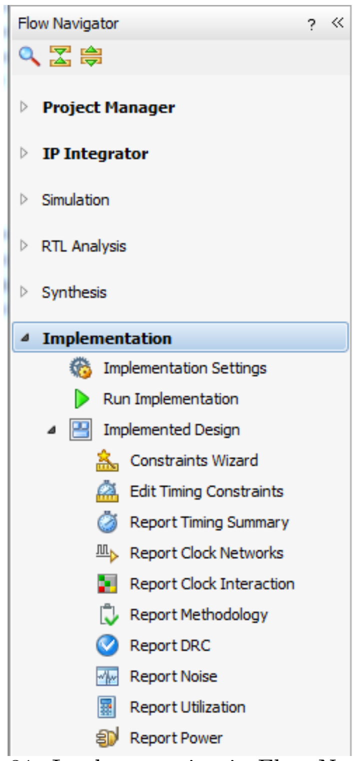
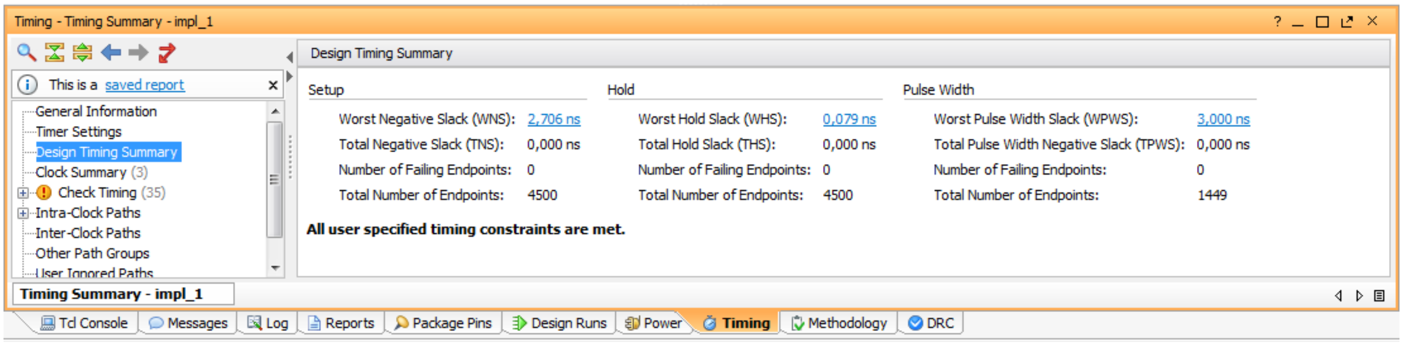


Figure 31: Implementation in Flow Navigator

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 32. Do inspect the different details of the summary.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,706 ns	Worst Hold Slack (WHS):	0,079 ns	Worst Pulse Width Slack (WPWS):	3,000 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4500	Total Number of Endpoints:	4500	Total Number of Endpoints:	1449

All user specified timing constraints are met.

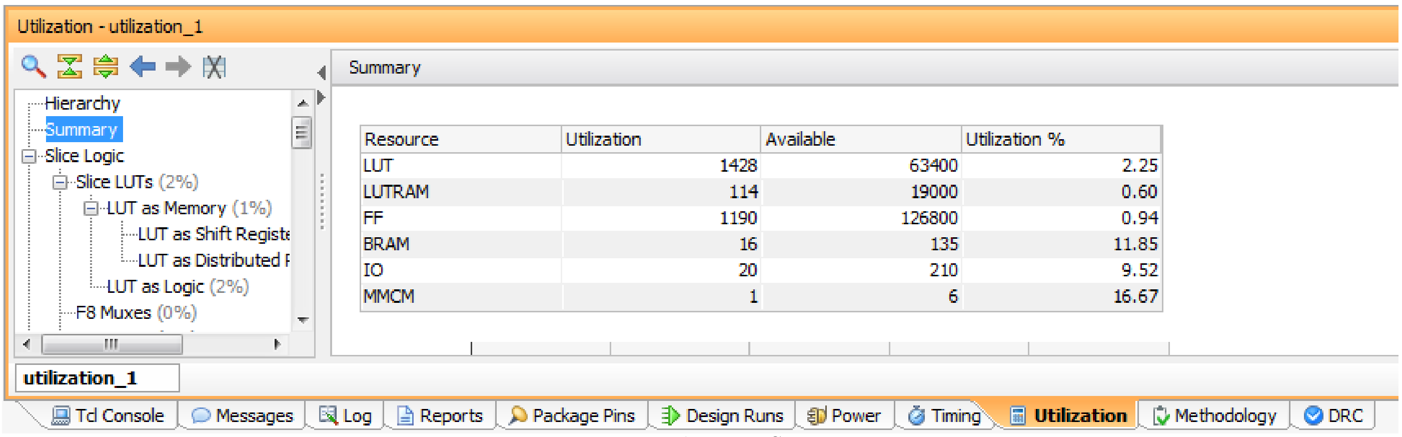
Figure 32: Timing Summary. All timing constraints are met in this case.

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 33. Do inspect the different details of the summary, as you will need to report these figures.

8.3 Power

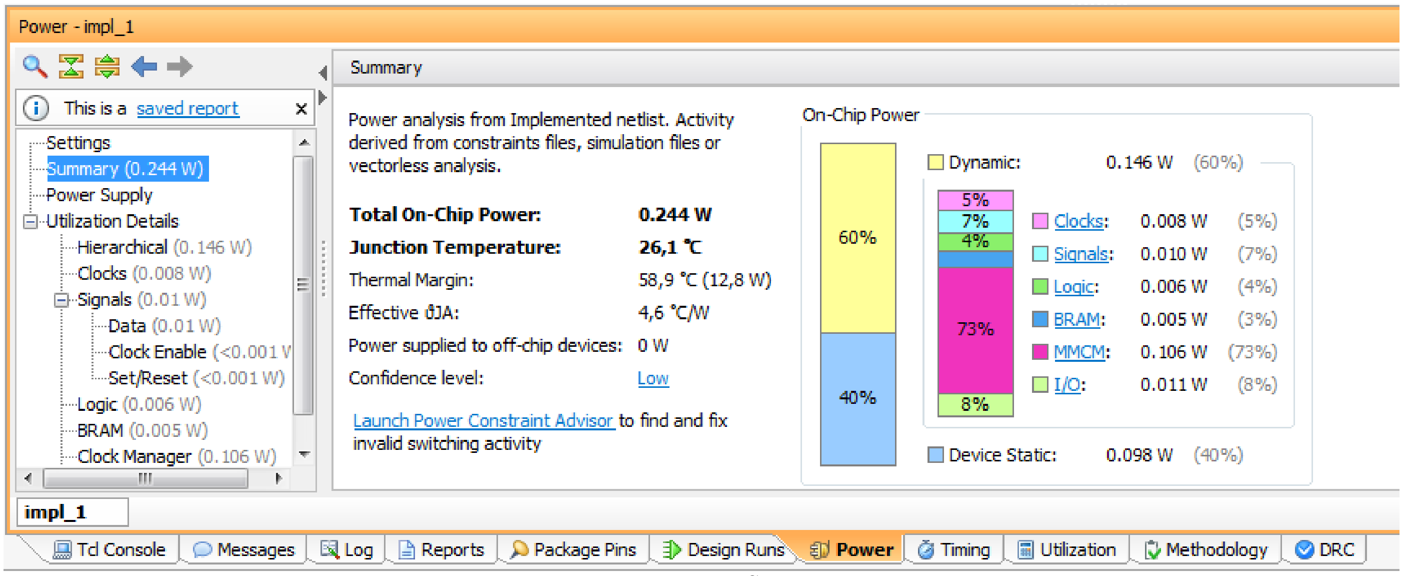
Finally, you can also get an estimate for the power consumption of the design by selecting **Report Power**. This will display detailed information in the lower window, as in Figure 34. Do inspect the different details of the summary, as you will need to report this figure and use it to compute the energy consumption for the different algorithms implemented in future labs.



Summary

Resource	Utilization	Available	Utilization %
LUT	1428	63400	2.25
LUTRAM	114	19000	0.60
FF	1190	126800	0.94
BRAM	16	135	11.85
IO	20	210	9.52
MMCM	1	6	16.67

Figure 33: Utilization Summary



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.244 W**
Junction Temperature:	**26,1 ℃**
Thermal Margin:	58,9 ℃ (12,8 W)
Effective θJA:	4,6 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	0.146 W	(60%)
Clocks:	0.008 W	(5%)
Signals:	0.010 W	(7%)
Logic:	0.006 W	(4%)
BRAM:	0.005 W	(3%)
MMCM:	0.106 W	(73%)
I/O:	0.011 W	(8%)
Device Static:	0.098 W	(40%)

Figure 34: Power Summary

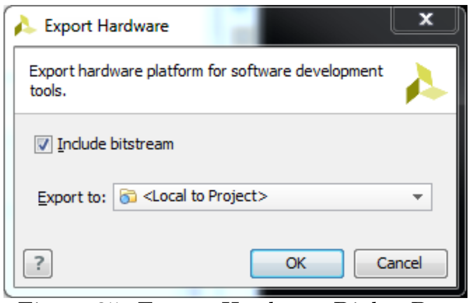
Creating The Software

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 35. Make sure that the **Export to** field is set to <**Local to Project**>.



Export Hardware

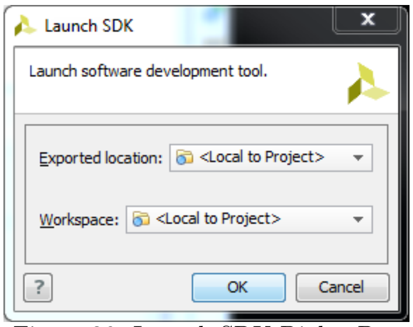
Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

Figure 35: 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**>.



Launch SDK

Launch software development tool.

Exported location: <Local to Project>

Workspace: <Local to Project>

Figure 36: 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. Some of the future labs may provide an `.hdf` file with the hardware, in order to save you some time. For more detailed information on how to use SDK with the `.hdf` please read EIT070 Lab1 Instructions pages 5–9.

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 37.

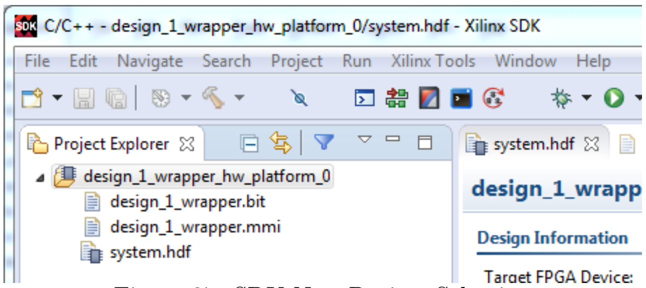

C/C++ - design_1_wrapper_hw_platform_0/system.hdf - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Tools Window Help
Project Explorer
design_1_wrapper_hw_platform_0
design_1_wrapper.bit
design_1_wrapper.mmi
system.hdf
system.hdf
design_1_wrapp
Design Information
Target FPGA Device:

Figure 37: SDK New Project Selection

2. In the New Project dialog box, select **Xilinx Application Project** (Figure 38).

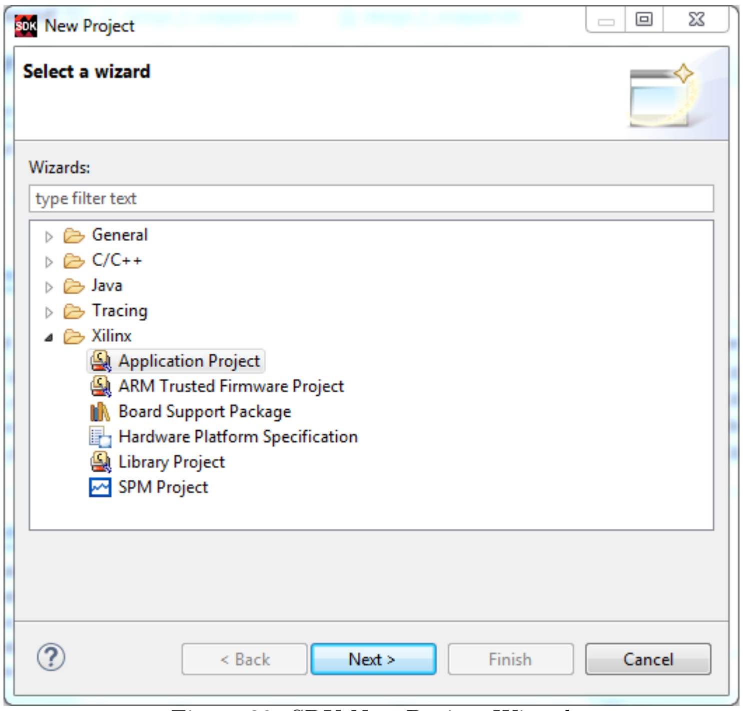
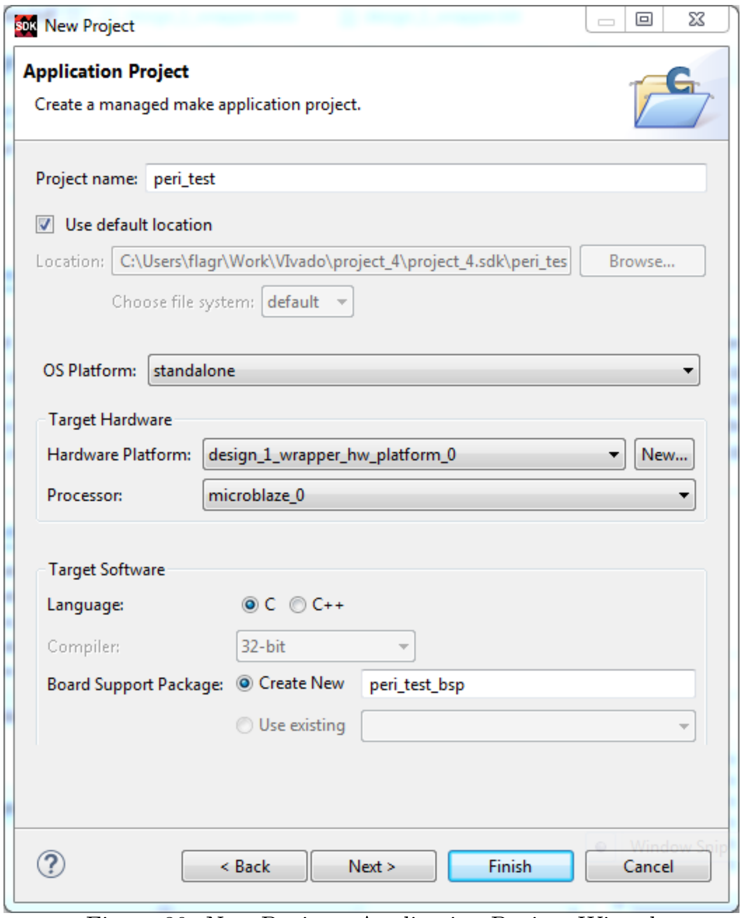


Figure 38: SDK New Project Wizard

3. Click **Next**.
4. Type a name (such as `peri_test`) for your project and choose standalone as OS platform, as shown in Figure 39.
5. Click **Next**.
6. Select the **Peripheral Tests** application template as in Figure 40, 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 41). 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 future labs with the AXI Timer, the `xtmrctr_polled_example.c` is relevant.



New Project

Application Project

Create a managed make application project.

Project name: peri_test

[✓] Use default location

Location: C:\Users\flagr\Work\VIvado\project_4\project_4.sdk\peri_tes

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: design_1_wrapper_hw_platform_0

Processor: microblaze_0

Target Software

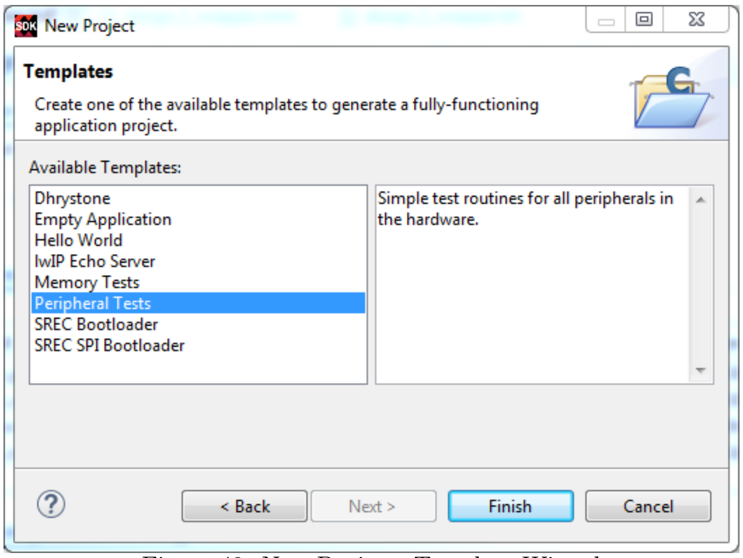
Language: [✓ C] C++

Compiler: 32-bit

Board Support Package: [✓ Create New] peri_test_bsp

Use existing

Figure 39: New Project: Application Project Wizard



New Project

Templates

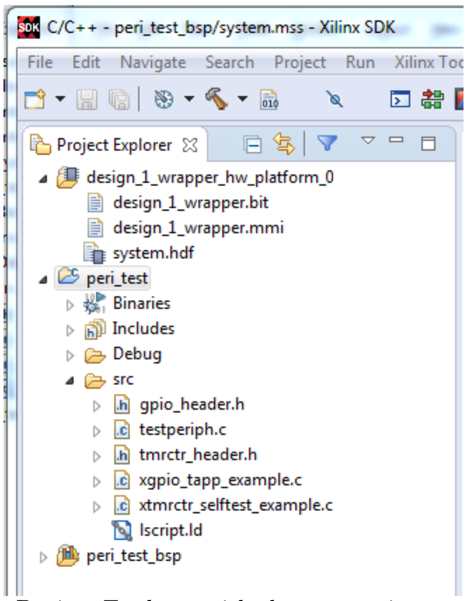
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- Peripheral Tests
- SREC Bootloader
- SREC SPI Bootloader

Simple test routines for all peripherals in the hardware.

Figure 40: New Project: Template Wizard


C/C++ - peri_test_bsp/system.mss - Xilinx SDK

```
design_1_wrapper_hw_platform_0
    design_1_wrapper.bit
    design_1_wrapper.mmi
    system.hdf
peri_test
    Binaries
    Includes
    Debug
    src
        gpio_header.h
        testperiph.c
        tmrctr_header.h
        xgpio_tapp_example.c
        xtmrctr_selftest_example.c
        lscript.ld
peri_test_bsp
```

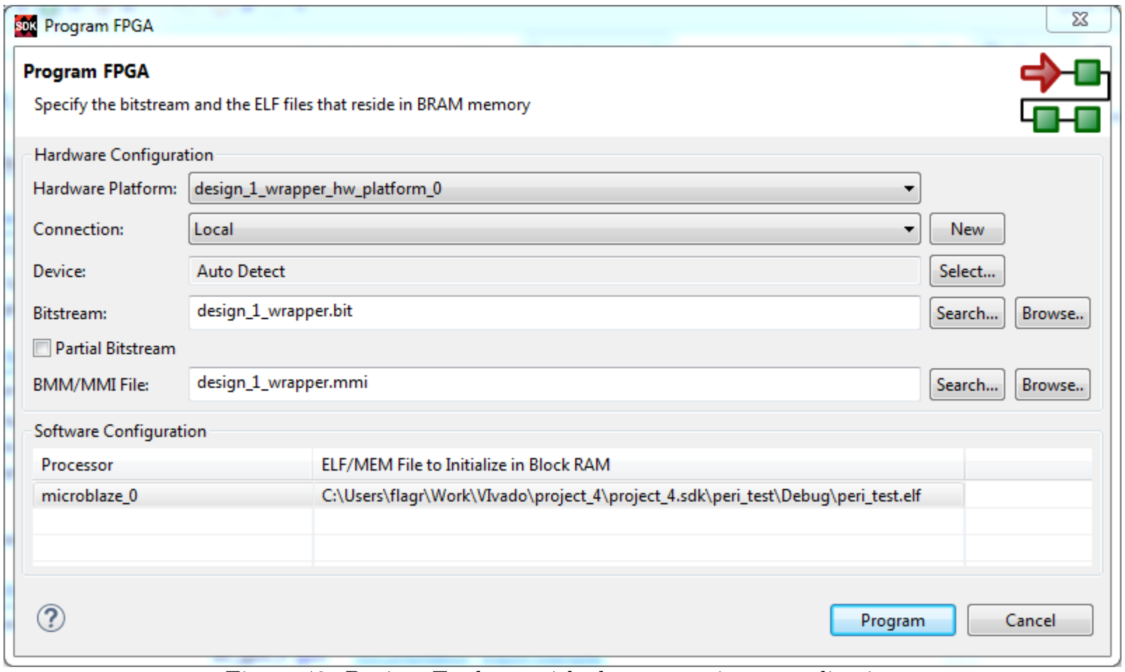
Figure 41: Project Explorer with the new peri_test application

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 42. Click **Program**.


Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

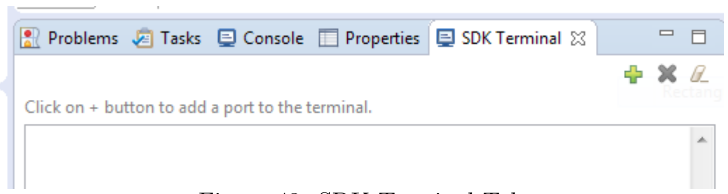
Hardware Platform:	design_1_wrapper_hw_platform_0
Connection:	Local
Device:	Auto Detect
Bitstream:	design_1_wrapper.bit
Partial Bitstream	☐
BMM/MMI File:	design_1_wrapper.mmi

Software Configuration

Processor	ELF/MEM File to Initialize in Block RAM
microblaze_0	C:\Users\flagr\Work\Vivado\project_4\project_4.sdk\peri_test\Debug\peri_test.elf

Figure 42: Project Explorer with the new peri_test application

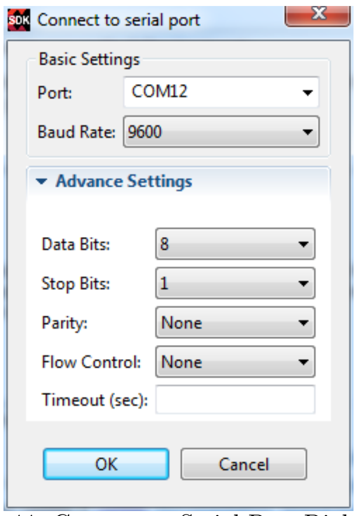
3. Although the system is up and running (check the LEDs), there is no way to see printouts unless we connect to the board via a Terminal. This can be done in SDK. Select **SDK Terminal** tab in the lower/Console region, as shown in Figure 43, then click on + to connect to a serial port.



Click on + button to add a port to the terminal.

Figure 43: 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 44. Click **OK**.



Connect to serial port

Basic Settings

Port: COM12

Baud Rate: 9600

Advance Settings

Data Bits: 8

Stop Bits: 1

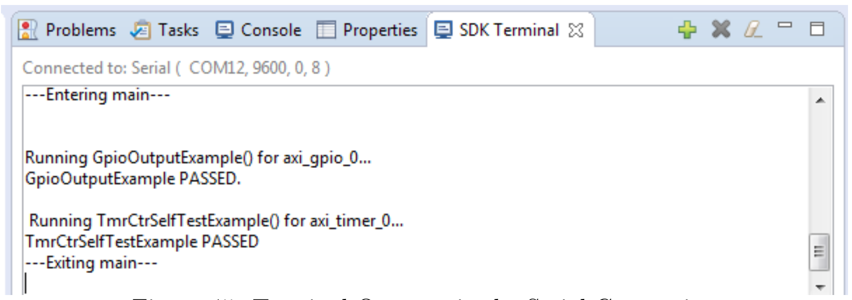
Parity: None

Flow Control: None

Timeout (sec):

Figure 44: Connect to Serial Port Dialog Box

5. Your terminal window should now receive text from the board, similar to Figure 45. Try pushing the red **CPU RESET** button on the Nexys-4 board and examine both the LEDs and the terminal output.



Connected to: Serial (COM12, 9600, 0, 8)

```
---Entering main---


Running GpioOutputExample() for axi_gpio_0...
GpioOutputExample PASSED.

 Running TmrCtrSelfTestExample() for axi_timer_0...
TmrCtrSelfTestExample PASSED
---Exiting main---
```

Figure 45: 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 consumption in Vivado
- export the hardware architecture into SDK
- create a simple application using your architecture in SDK
- run the full system on the Nexys-4 board

Most of these skills will be used in the lab assignments, and can be further develop to allow you to be more efficient. A couple of good documents to get further information (e.g. on debugging) are:

- Xilinx *Embedded Processor Hardware Design*, UG940 (v2016.3) October 19, 2016
- EIT070 Lab1 Instructions