ECE4810J System-on-Chip (SoC) Design

Fall 2021

Lab #1 Getting Started with Arty Z7 SoC development platform

Due: 11:59pm Sept. 30th, 2021

Logistics:

- This lab is a team exercise.
- Please use the discussion board on Piazza for Q&A.
- All reports and code (if available) MUST be submitted to the assignment of Canvas.
- Internet usage is allowed and encouraged.
- No late submission is allowed for this lab.

1. Overview

In this lab, you will go through several exercises to set up and get started with the Arty Z7-20 SoC development board. The goals of this lab are to:

- Get familiar with the board
- Set up the environment correctly
- Be able to build a basic Zynq system on the board
- Learn to create custom IP blocks at RTL level (Verilog, VHDL)
- Use AXI bus to connect an IP block with the Zynq PS

2. Environment Setup

Please follow the tutorial below to install Vivado, Xilinx SDK, and Digilent Board Files.
https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-sdk?redirect=1
Note: If you already have Vivado 2018.2 or later on your PC, please skip the first step and go to Step 3 directly.
Regarding the Vivado installation process, you can also refer to canvas>Files>Software>A Brief Tutorial for Xilinx Vivado--New Version.docx.

3. Building a basic ZYNQ system on the Arty Z-7 board

3.1 Create a new Vivado project

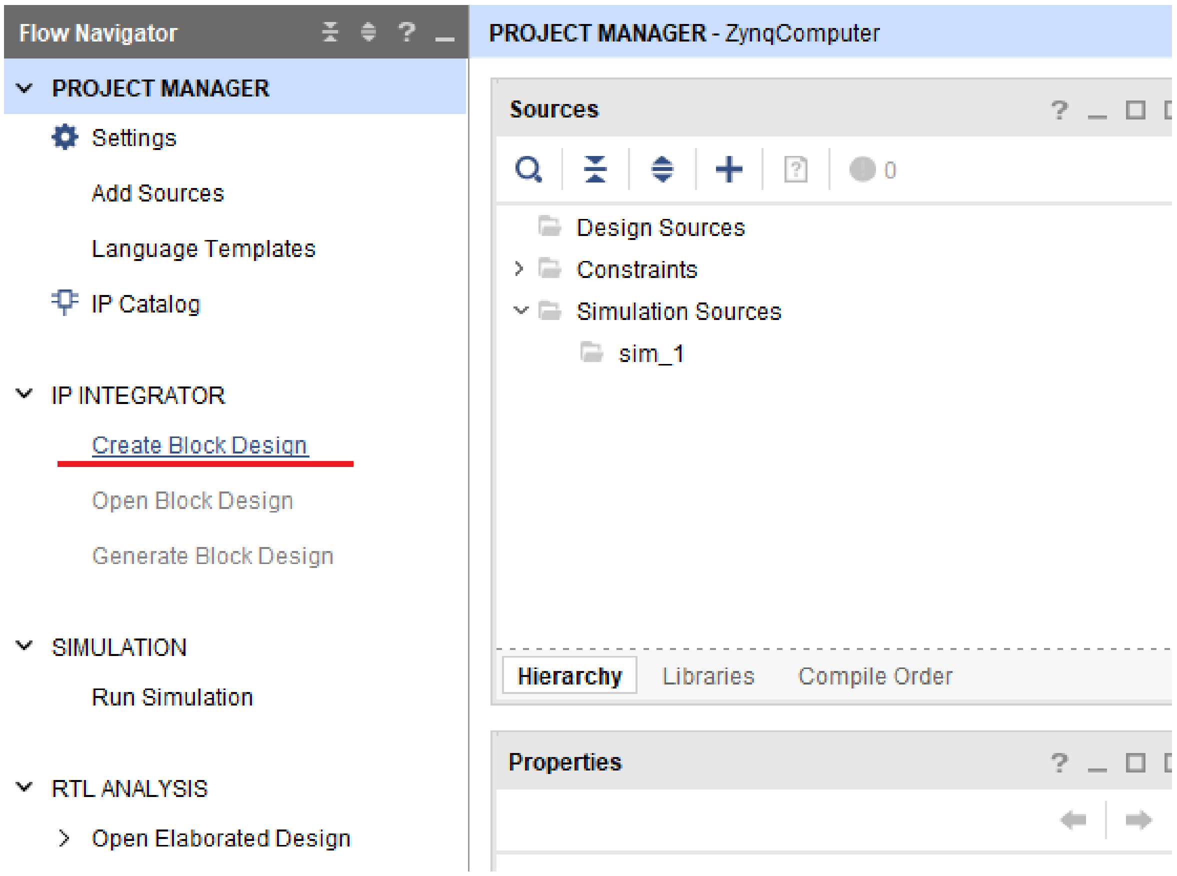
On your computer, go to Start -> Xilinx Design Tools -> Vivado <version>. From the welcome screen, under Quick Start, click Create Project and follow the wizard.

- Name the project as ZynqComputer and locate it in a directory like C:\FPGALabGroupName\.
- Go with RTL Project, Next, Next, and for the Default Part, click on Boards and select the Arty-Z7 board.
- If the board is not listed, you might need to download the board file and add it into the Vivado folder (Section 2 should already cover this part)

3.2 Setup the Zynq Processor System (PS)

The new Vivado project starts off blank, so to create a functional base design, we need to at least add the Zynq PS (processor system) and make the minimal required connections. Follow these steps to add the PS to the project:

- From the Vivado Flow Navigator, under IP Integrator, click Create Block Design.
    - Next, specify a name for the block design, for example Zynq_CPU.



- In the Block Design Diagram, you will be informed that the design is empty. Press the Plus button to browse for existing IP blocks.

- In the IP catalog, search for Zynq and select the ZYNQ7 Processing System by double-clicking on it. The ZYNQ IP block will appear in the Diagram.

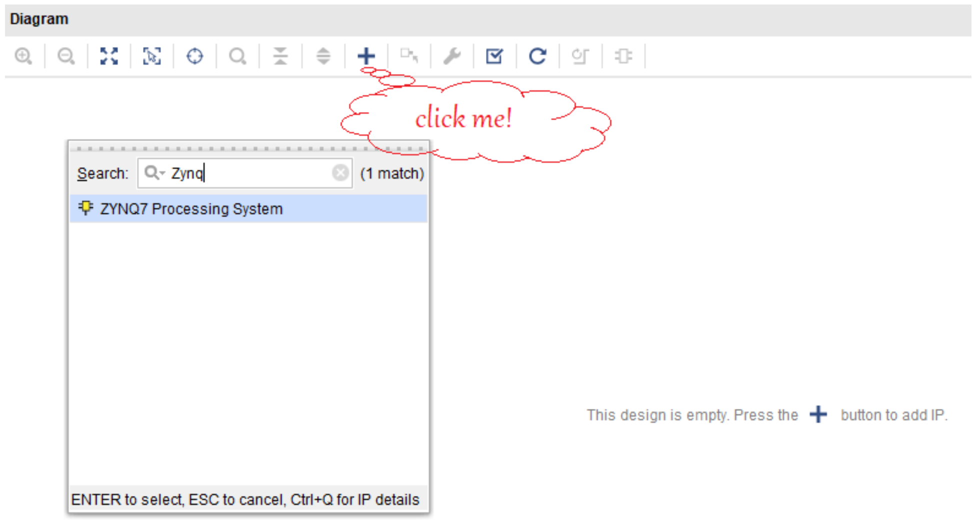
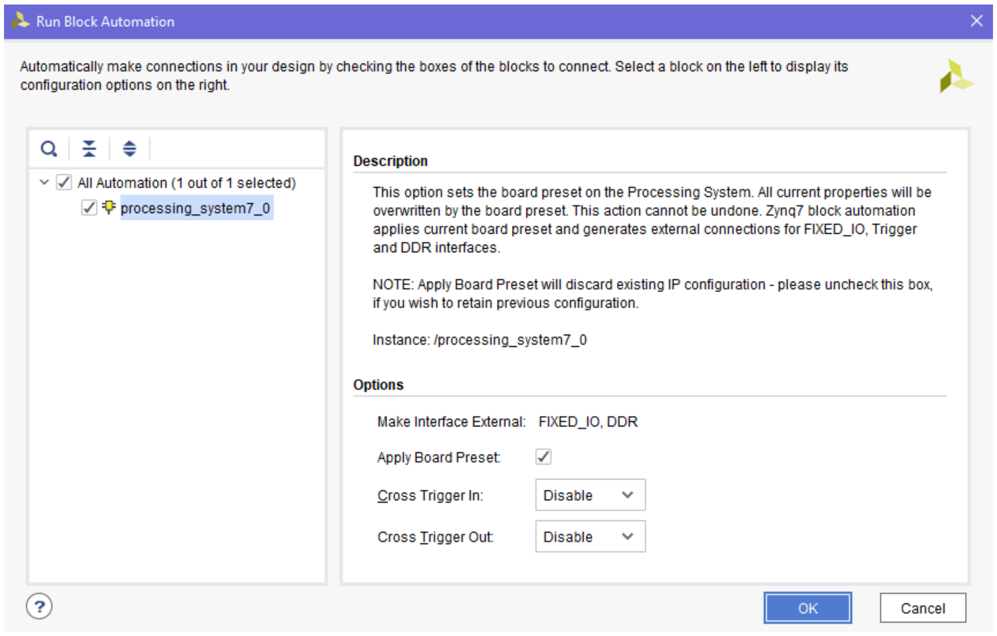


Diagram
click me!
Search: Zynq (1 match)
ZYNQ7 Processing System
ENTER to select, ESC to cancel, Ctrl+Q for IP details
This design is empty. Press the + button to add IP.

- In the Block Design Diagram, you will see a message that says Designer Assistance available. Run Block Automation. Click on the Run Block Automation link and select processing_system7_0. Block Automation makes connections and pin assignments to external hardware such as the DDR and fixed IO. It does this using the board definition of the hardware platform you specified when you created the project (Arty Z7). We could make these connections ourselves if we were using a custom board, but for off-the-shelf boards, Block Automation makes the process a lot easier.



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)
processing_system7_0

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

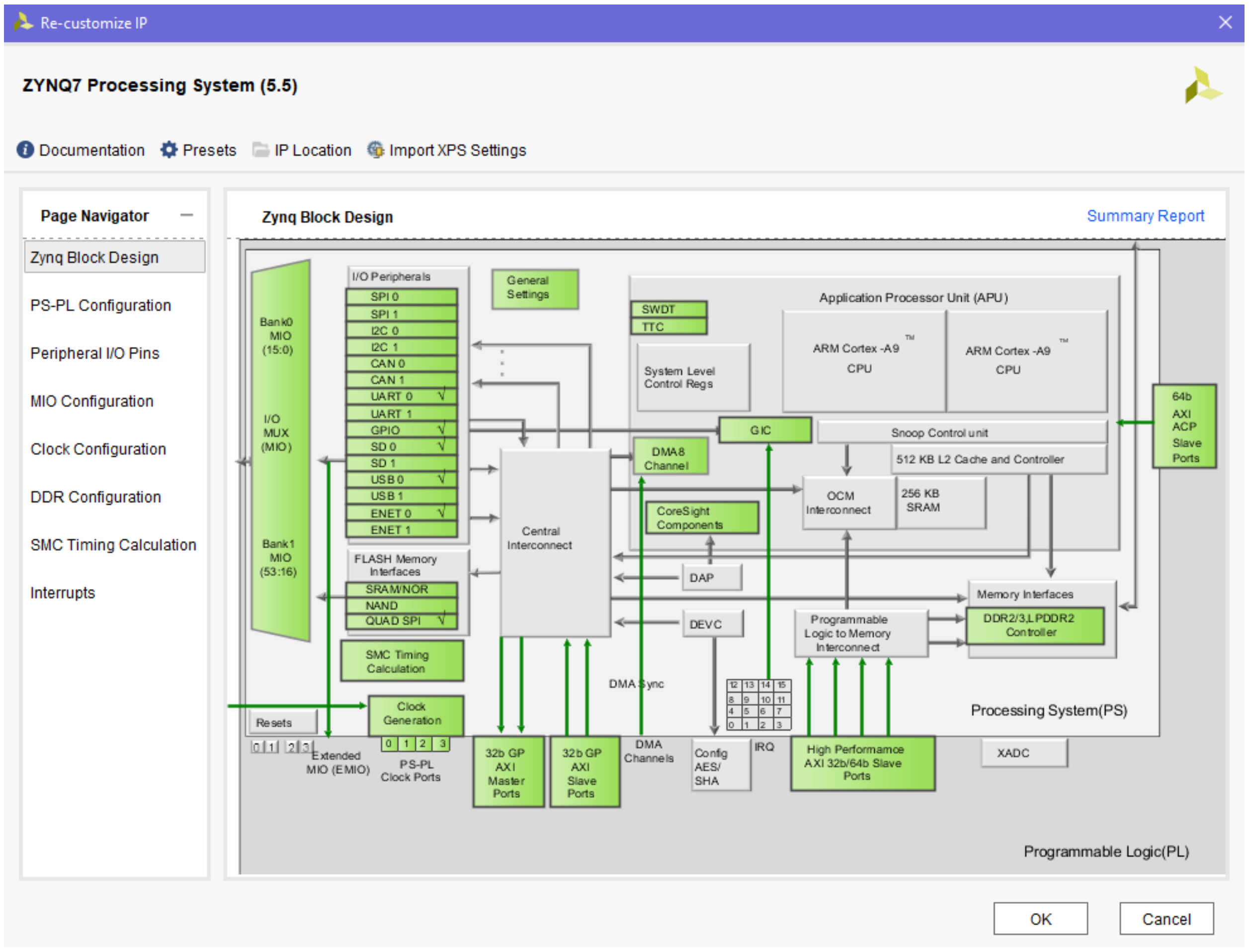
NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

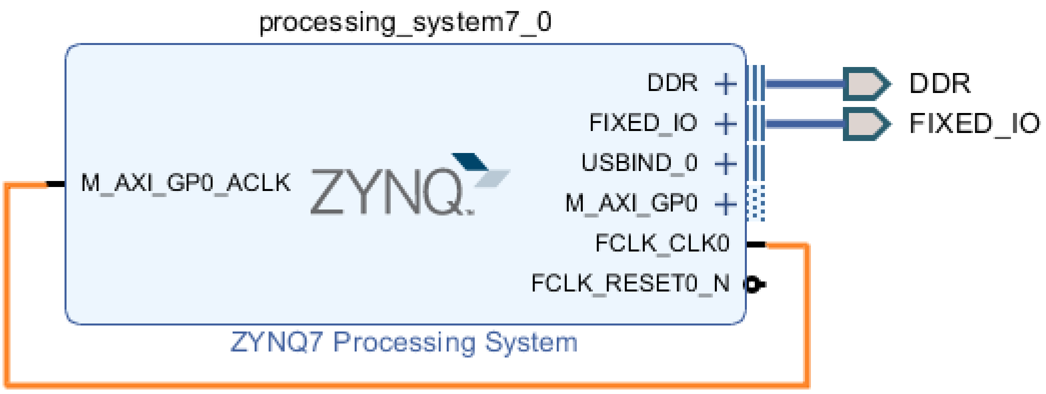
Options

Make Interface External:	FIXED_IO, DDR
Apply Board Preset:	✓
Cross Trigger In:	Disable
Cross Trigger Out:	Disable

- Now the block diagram has changed and we can see that the DDR and FIXED_IO are connected externally. Double-click the ZYNQ PS block and explore the customization options available. Leave all settings as default.



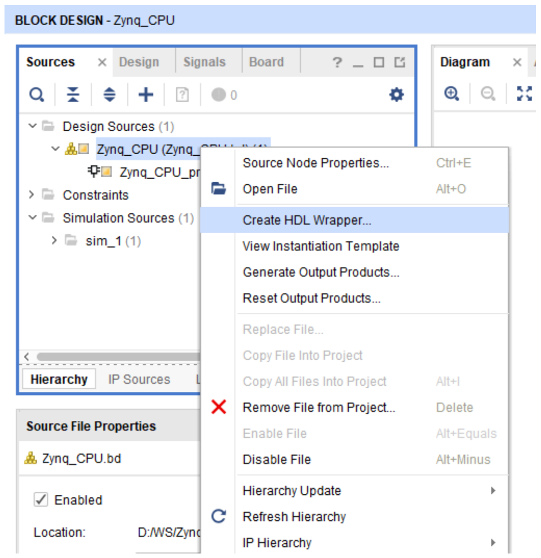
- The only remaining connection to make is the clock that we will use for the AXI buses. Connect the FCLK_CLK0 output to the M_AXI_GP0_ACLK clock input. To do this, click on the FCLK_CLK0 output and drag with the pencil onto the M_AXI_GP0_ACLK input. This will trace a wire between the pins and make the connection.



3.3 Create the HDL wrapper

Now the Zynq Processing System is setup and all we need to do is to create a HDL wrapper for the design.

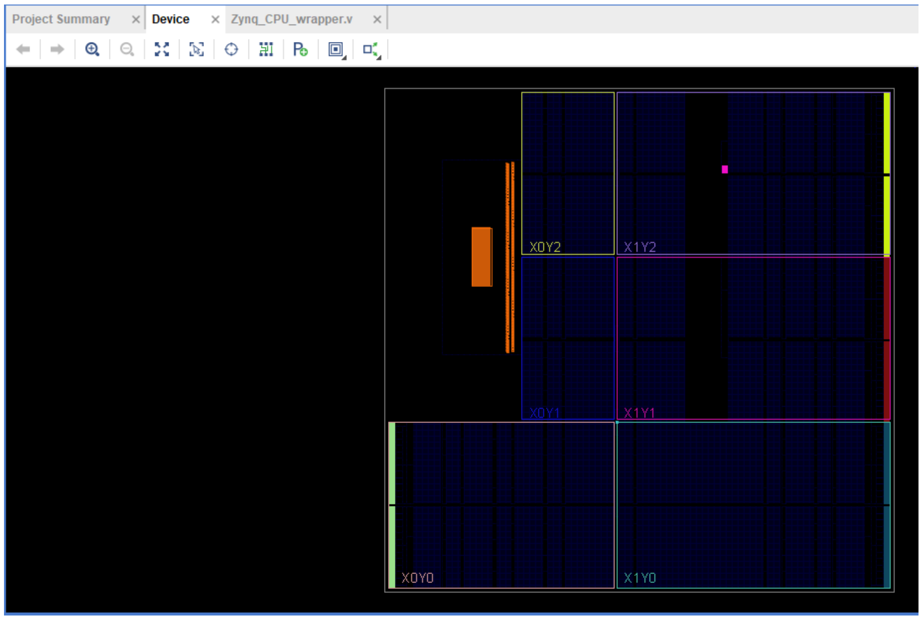
- Save the Block Design, then, under the Sources tab, right-click on Zynq_CPU and select Create HDL wrapper... and then go with the Let Vivado manage wrapper and auto-update.



We now have a base design containing the Zynq PS from which we could generate a bitstream and test on the Arty Z7 board. We haven't exploited any of the FPGA fabric, but the Zynq PS is already connected to the DDR, Gigabit Ethernet PHY, the USB PHY, the SD card, the UART port and the GPIO, all thanks to the Block Automation feature. So there is already quite a lot we could do with the design at this point, such as running Linux on the PS or running a bare metal application on it.

3.4 Generate the bitstream

- To generate the bitstream needed to program the chip, click Generate Bitstream under Program and Debug in the Flow Navigator. You can also go step by step, first with Run Synthesis, then Run Implementation and then Generate Bitstream.
- Once the bitstream is generated, select Open Implemented Design from the Bitstream Generation Completed dialog box.
- The implemented design will open in Vivado showing you a map of the Zynq device and how the design has been placed. In our case, we haven't used any of the FPGA fabric (only the PS), so the map is empty for the most part.

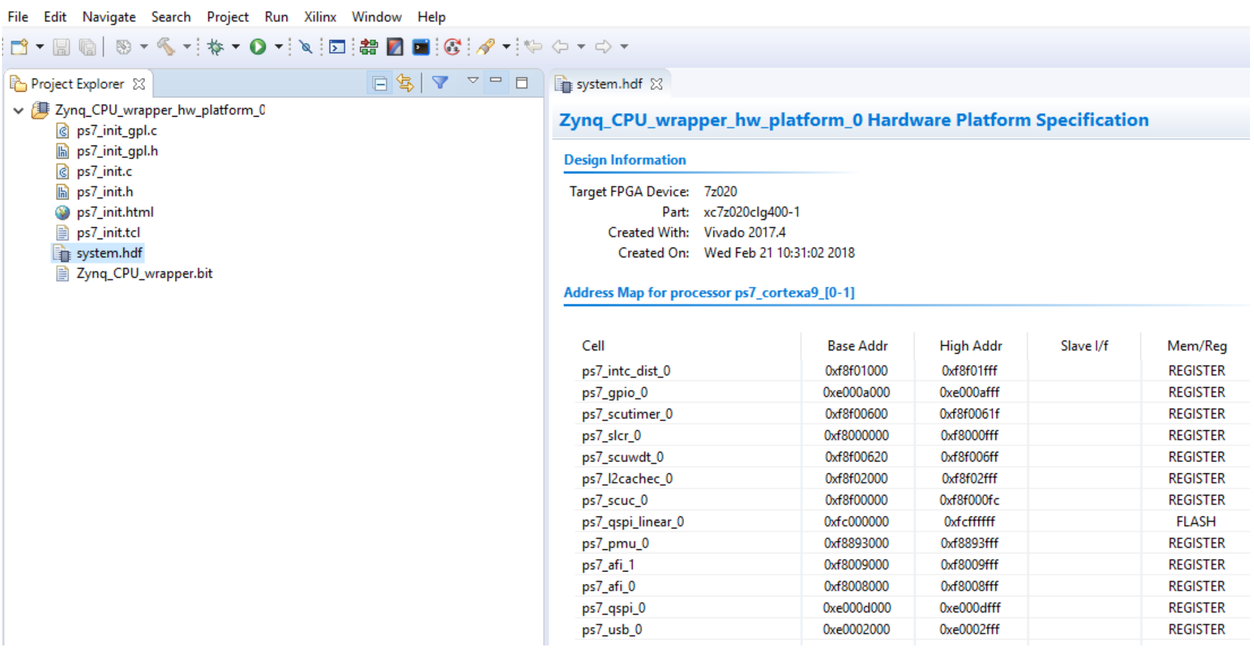


3.5 Export the hardware to SDK

Once the bitstream has been generated, the hardware design is done and we're ready to develop the code to run on the processor. This part of the design process is done in Xilinx Software Development Kit (SDK), so from Vivado we must first export the project to SDK.

- In Vivado, from the File menu, select Export -> Export Hardware..., then check the Include bitstream option and click OK.
- Again from the File menu, select Launch SDK.

At this point, the familiar Eclipse environment SDK loads and a hardware platform specification is created for your design, as shown below. Now, let's create a software application to run on the PS.



Zynq_CPU_wrapper_hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device: 7z020
Part: xc7z020clg400-1
Created With: Vivado 2017.4
Created On: Wed Feb 21 10:31:02 2018

Address Map for processor ps7_cortexa9_[0-1]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
ps7_qspi_linear_0	0xfc000000	0xfcffffff		FLASH
ps7_pmu_0	0xf8893000	0xf8893fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_qspi_0	0xe000d000	0xe000dfff		REGISTER
ps7_usb_0	0xe0002000	0xe0002fff		REGISTER

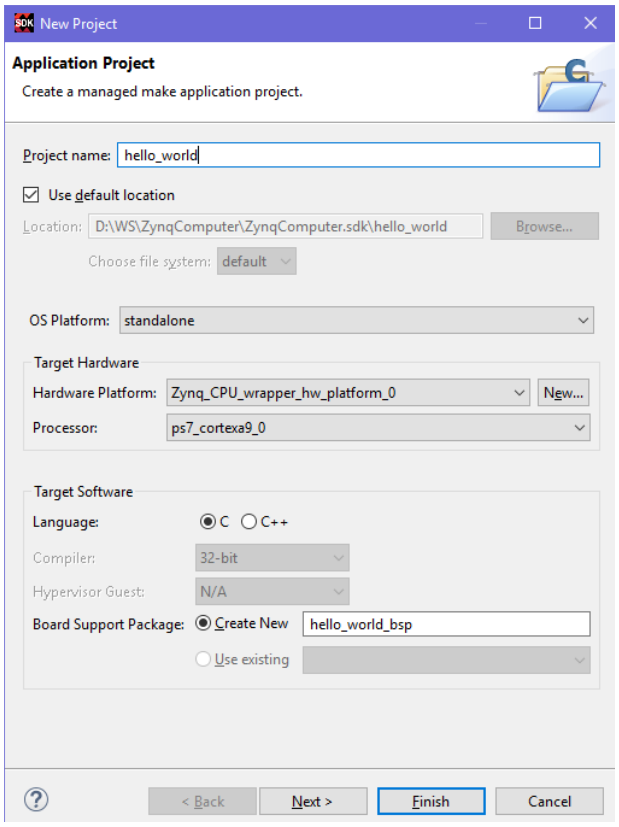
3.6 Create a Software application

To demonstrate creating an application for the Zynq, we'll create a hello world application that will send Hello, World! out the UART and to your PC.

- From the File menu, select New -> Application Project.
- Set the name of the project as hello_world, the OS Platform is standalone, the Hardware Platform is our previously created Zynq_CPU_wrapper_hw_platform_0, etc.
- Click Next, then, on the Templates page, select the Hello World template and click Finish.

The SDK will generate two new project folders, hello_world and hello_world_bsp. The hello_world folder contains the software application, which you can browse and modify. I suggest you take a look at the code that is contained here and get familiar with it. The hello_world_bsp folder contains the Board Support Package, which is a bunch of libraries that are provided for accessing the various peripherals and features of the Zynq. In general, it's better not to modify this code because it gets written over every time you use the Clean Project option.

Once the software application has been created, it is automatically built. Once the application is built, we are ready to test the design on hardware.


New Project

Application Project

Create a managed make application project.

Project name: hello_world

☑ Use default location

Location: D:\WS\ZynqComputer\ZynqComputer.sdk\hello_world

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: Zynq_CPU_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

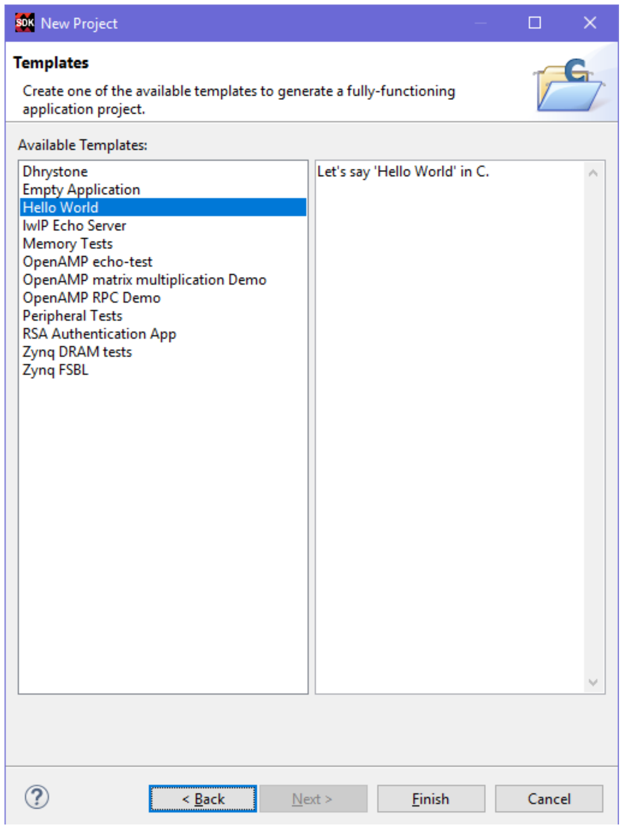
Language: [✓ C] C++

Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] hello_world_bsp

Use existing


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
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

Let's say 'Hello World' in C.

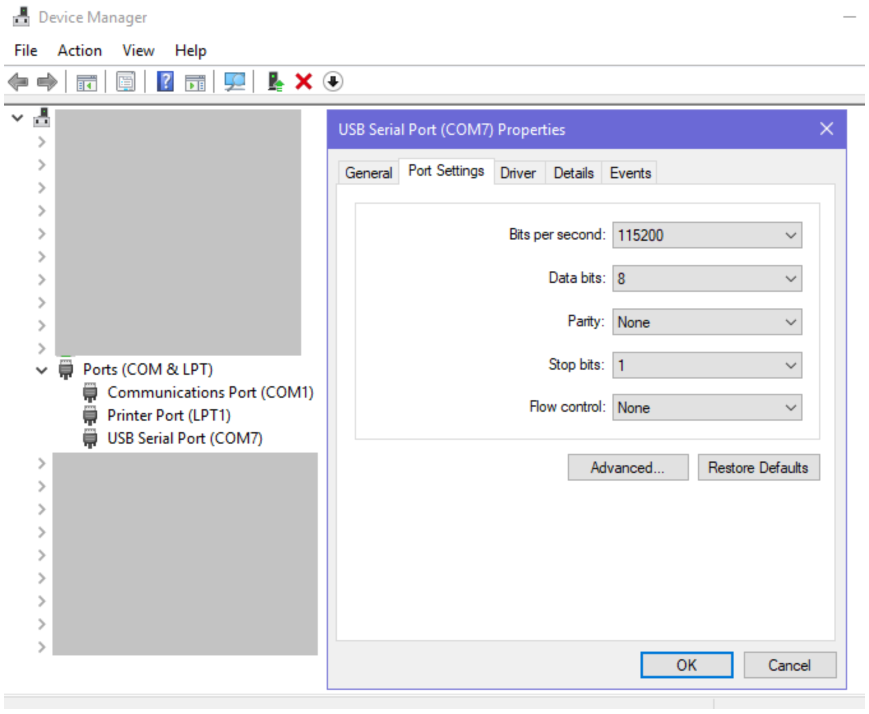
3.7 Test the design on the hardware

1. Grab a Arty Z7 board and connect the USB cable to your PC and to the *PROG UART* port.
2. Set the *JP4* jumper to *JTAG* mode to enable chip configuration via the JTAG.
3. Set the *JP5* jumper to *USB* mode to enable power supply via the USB.
4. You should see the LED *LD13* shine red.

Now you need to set up the UART connection between the PC and the board.

- In Windows 10, right-click on the Start button and go to Device Manager.
- Under Ports, one of the available COM ports, probably the latest ones, is the Arty Z7 board. Right click on it, and set the Port Settings as shown below.

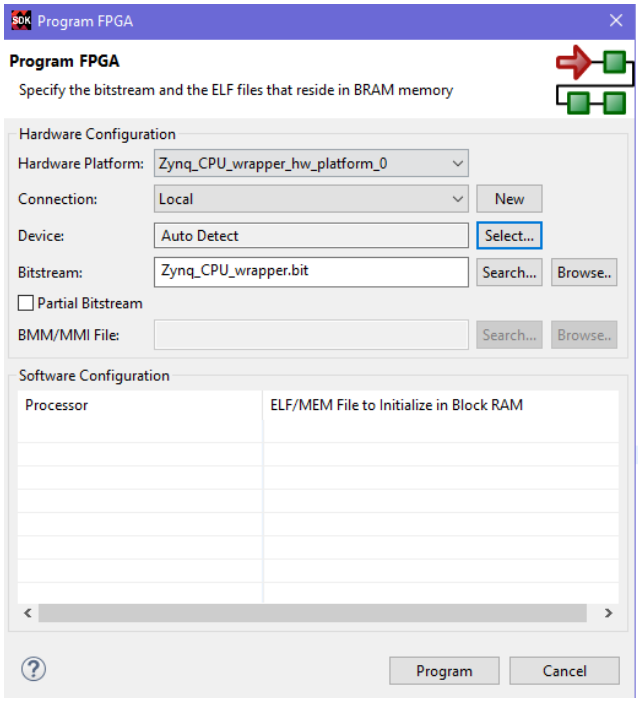
- Note: If you can't set the Port Settings in Windows due to lack of administrator rights, do not worry. Setting the correct settings in the SDK Terminal should be sufficient.


Device Manager
File Action View Help
Ports (COM & LPT)
Communications Port (COM1)
Printer Port (LPT1)
USB Serial Port (COM7)
USB Serial Port (COM7) Properties
General Port Settings Driver Details Events
Advanced... Restore Defaults
OK Cancel

Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

Now that your PC is ready to receive the Hello World message, we are ready to send our bitstream and software application to the hardware. In the SDK, from the menu, select Xilinx Tools -> Program FPGA. Ensure that the settings in the Program FPGA dialog window are reasonable and click Program. If successful, there will be no error messages, the SDK Log will output the message FPGA configured successfully ... and the LED LD12 on the Arty Z7 board will shine yellow.

- Click on the SDK Terminal tab and then on the green plus button to connect the SDK to a serial port. Then select the settings matching the ones set in Device Manager and clock OK. You should then see the message Connected to COM_nr at 115200 in the SDK Terminal.
- Lastly, select the project folder hello_world, then select the menu Run -> Run and select Lunch on Hardware (GDB) in the Run As dialog window. The program will run on the board and you will see the message Hello World in the SDK Terminal.



Program FPGA

Program FPGA

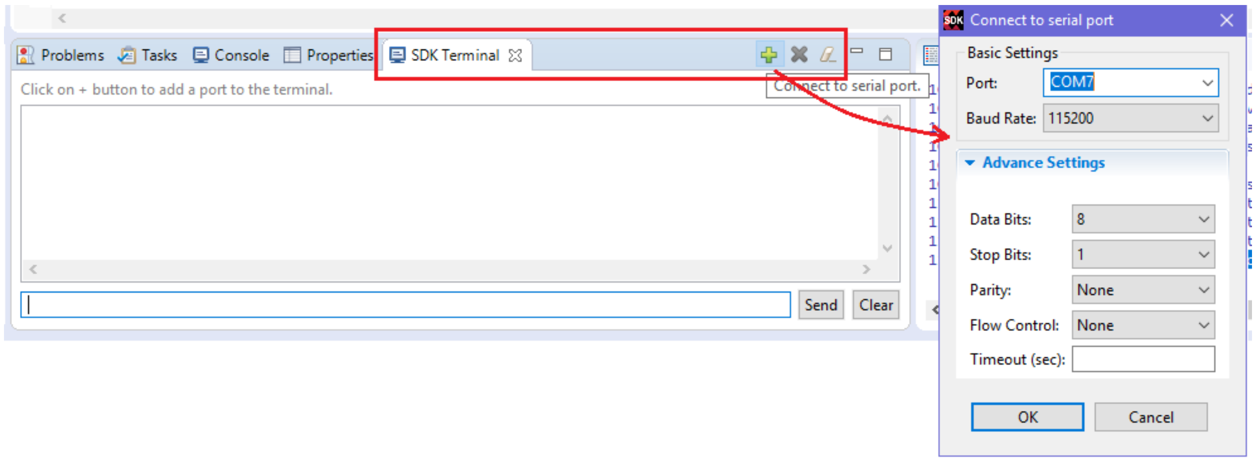
Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Platform:	Zynq_CPU_wrapper_hw_platform_0		
Connection:	Local	New	
Device:	Auto Detect	Select...	
Bitstream:	Zynq_CPU_wrapper.bit	Search...	Browse..
☐ Partial Bitstream			
BMM/MMI File:		Search...	Browse..

Software Configuration

Processor	ELF/MEM File to Initialize in Block RAM



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

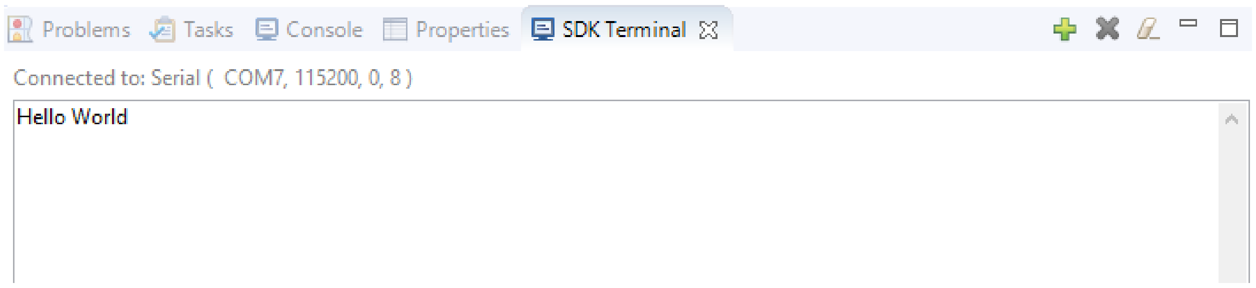
Connect to serial port

Basic Settings

Port:	COM7
Baud Rate:	115200

Advance Settings

Data Bits:	8
Stop Bits:	1
Parity:	None
Flow Control:	None
Timeout (sec):	



Connected to: Serial (COM7, 115200, 0, 8)

```
Hello World
```

3.8 Follow-up

Now you have gone through the whole process of building a ZYNQ system on Arty Z7 board, as a followup, please change the initial C code so that your group name and group members' names are the output of the SDK terminal.

Note: Please refer to this site if anything is unclear https://digilent.com/reference/vivado/getting-started-with-ipi/2018.2

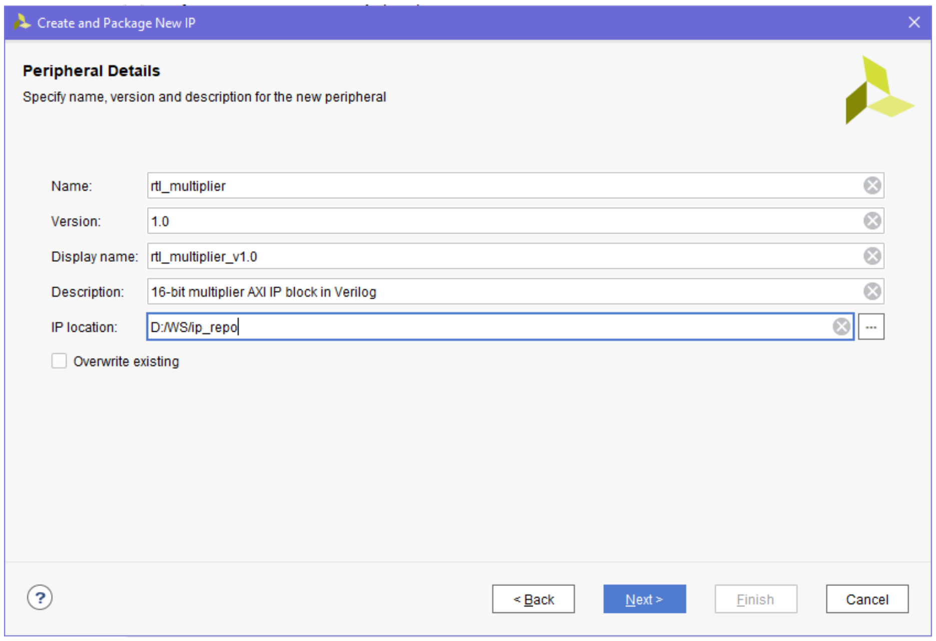
4. Creating and using custom IP blocks in Verilog

A system on a chip consisting of both a Hard Processor System and FPGA fabric, such as the Zynq-7000, offers the opportunity of offloading computation to the FPGA. Parallelizable algorithms can thus be accelerated, or more computations can be executed in parallel. In this section, we will create a simple custom AXI IP block that multiplies two numbers and will connect it to the Zynq PS. The multiplier will take as input two 16-bit unsigned numbers and will output the product as one 32-bit unsigned number. A single 32-bit write to the IP block will contain the two 16-bit inputs, separated into the lower and higher 16 bits. A single 32-bit read from the peripheral will contain the result from the multiplication of the two 16-bit inputs. Although this design doesn't really make much sense as an accelerator, it is a good learning example.

4.1 Create and Package the IP block

Feel free to start with any previously made Vivado project that contains a Zynq system. For example, you can start with the system you created in the previous section. To make your work easier, you can copy-paste the project ZynqComputer to ZynqComputerExtended and open it in Vivado too. Now that you have a project with a Zynq PS System open in Vivado, follow the instructions below.

- Start by going to menu Tools -> Create and Package New IP....
    - Read the overview of the Create and Package New IP wizard and then click Next.
- We are interested in a new AXI4 peripheral, therefore select the Create a new AXI4 peripheral and click Next. To read more about AXI, please go to Canvas>File>Reading Materials>Zynq> ug761_axi_reference_guide.pdf.



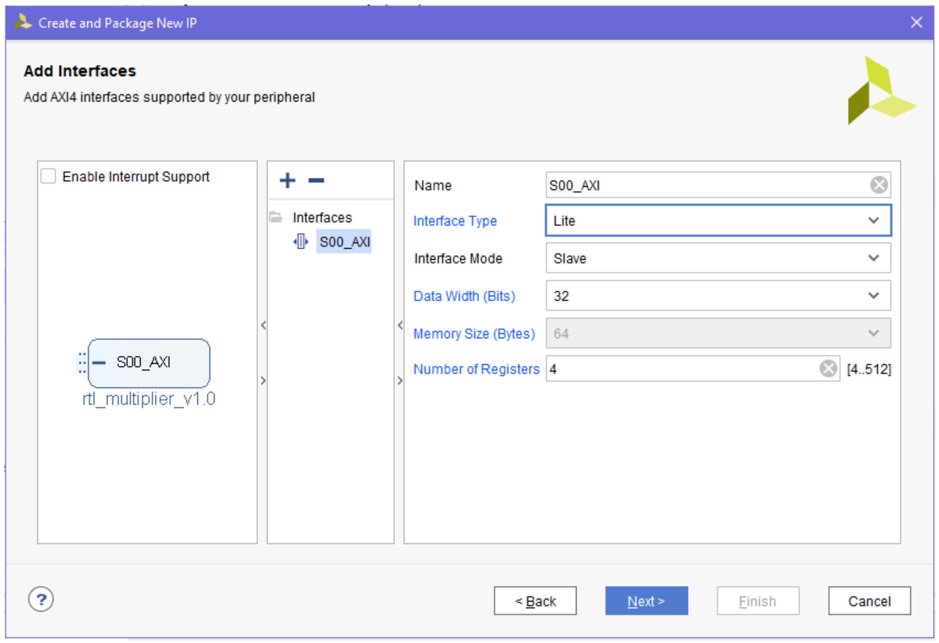
Peripheral Details

Specify name, version and description for the new peripheral

Name:	rtl_multiplier
Version:	1.0
Display name:	rtl_multiplier_v1.0
Description:	16-bit multiplier AXI IP block in Verilog
IP location:	D:/WS/ip_repo

☐ Overwrite existing

- On the *Add Interfaces* page, use the default 32-bit AXI4 Lite Slave interface.



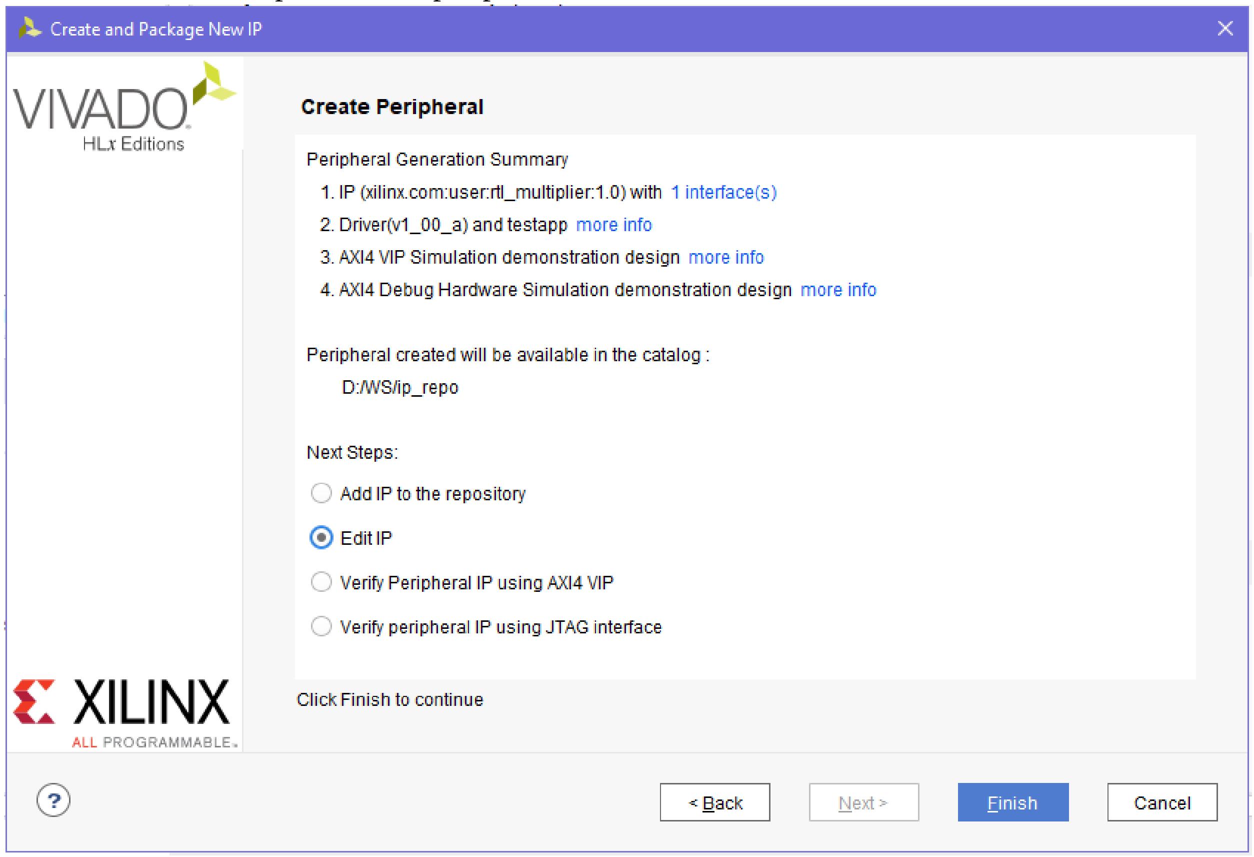
Add Interfaces

Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

- On the last page, select Edit IP and click Finish. This will open another Vivado window in which we will implement the peripheral.



Create Peripheral

Peripheral Generation Summary

1. IP (xilinx.com:user:rtl_multiplier:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 VIP Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

Peripheral created will be available in the catalog :
D:/WS/ip_repo

Next Steps:

- Add IP to the repository
- [✓ Edit IP]
- Verify Peripheral IP using AXI4 VIP
- Verify peripheral IP using JTAG interface

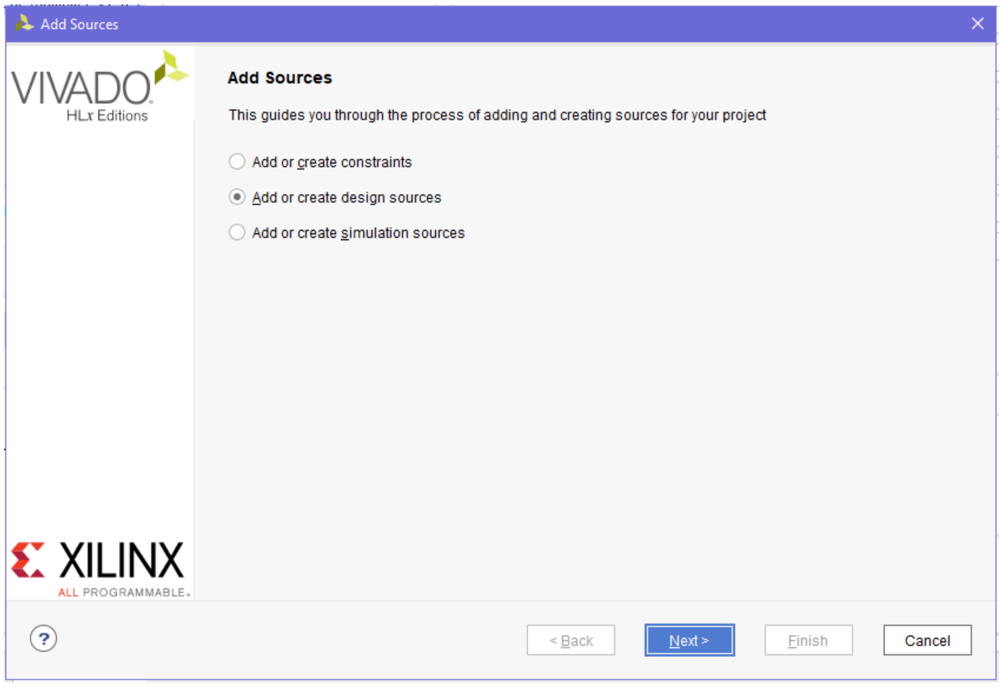
Click Finish to continue

4.2 Edit the IP block

- The multiplier Verilog code is simple since it only multiplies two numbers. For example, this code will do:

```
module rtl_multiplier(
    input clk,
    input [15:0] a,
    input [15:0] b,
    output [31:0] product
    );

    reg [31:0] productReg;
    assign product = productReg;
    always @(posedge clk) begin
        productReg <= a * b;
    end
endmodule
```

- Save this file as rtl_multiplier.v in a directory such as ip_repo\src\ for later access.
- From the pannel Flow Navigator on the left, click Add Sources and, in the new Add Sources window, select Add or create design sources.

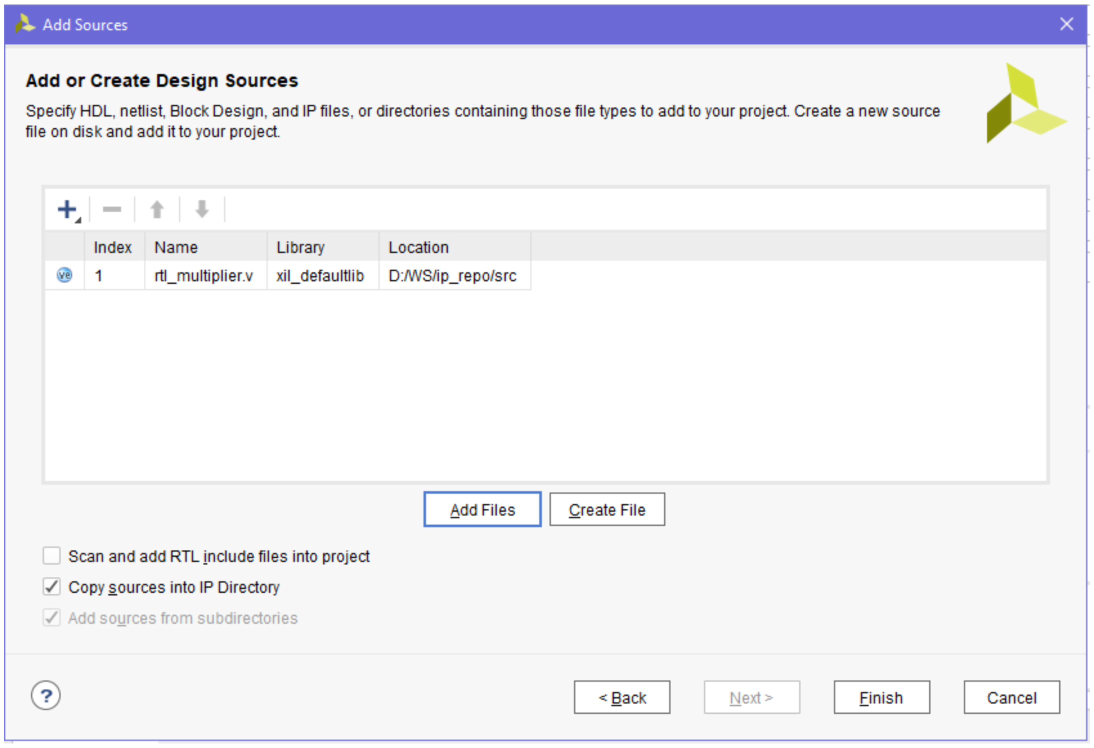


Add Sources

This guides you through the process of adding and creating sources for your project

Add or create constraints [✓ Add or create design sources] Add or create simulation sources

- Next, select the previously saved ip_repo\src\rtl_multiplier.v file and have the option Copy sources into IP Directory checked. Click Finish and the file will be added to the Design Sources.


Add Sources

Add or Create Design Sources

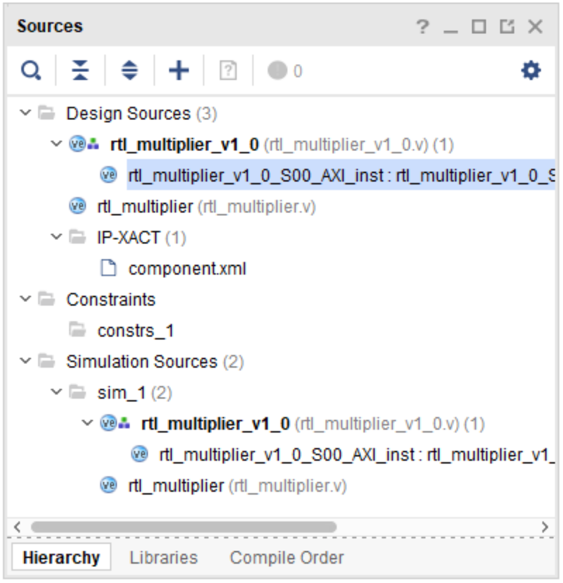
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	Index	Name	Library	Location
	1	rtl_multiplier.v	xil_defaultlib	D:/WS/ip_repo/src

☐ Scan and add RTL include files into project
☑ Copy sources into IP Directory
☑ Add sources from subdirectories

At this point the *rtl_multiplier.v* file is separately part of the *Design Sources*. Let's connect it to the AXI IP block.

- Expand the top branch (rtl_multiplier_v1_0.v) and open the file rtl_multiplier_v1_0_S00_AXI_inst by double-clicking on it.

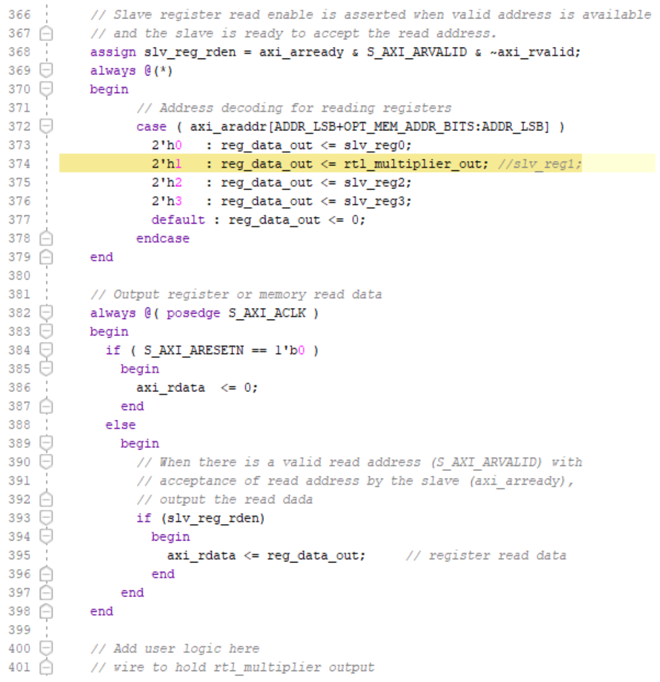

Sources

```
Design Sources (3)
  rtl_multiplier_v1_0 (rtl_multiplier_v1_0.v) (1)
    rtl_multiplier_v1_0_S00_AXI_inst : rtl_multiplier_v1_0_S
  rtl_multiplier (rtl_multiplier.v)
  IP-XACT (1)
    component.xml
Constraints
  constrs_1
Simulation Sources (2)
  sim_1 (2)
    rtl_multiplier_v1_0 (rtl_multiplier_v1_0.v) (1)
      rtl_multiplier_v1_0_S00_AXI_inst : rtl_multiplier_v1_
    rtl_multiplier (rtl_multiplier.v)
```

- Scroll down to the end of the file where the comment Add user logic here is and insert the code below. The code below instantiates the rtl_multiplier module inside the AXI IP block and connects the clock to the AXI clock, the inputs a and b to the 16 MSB and LSB of the first register (slv_reg0) and the output product to a created wire rtl_multiplier_out.
- Take a pulse here and try to understand this code before you move forward.

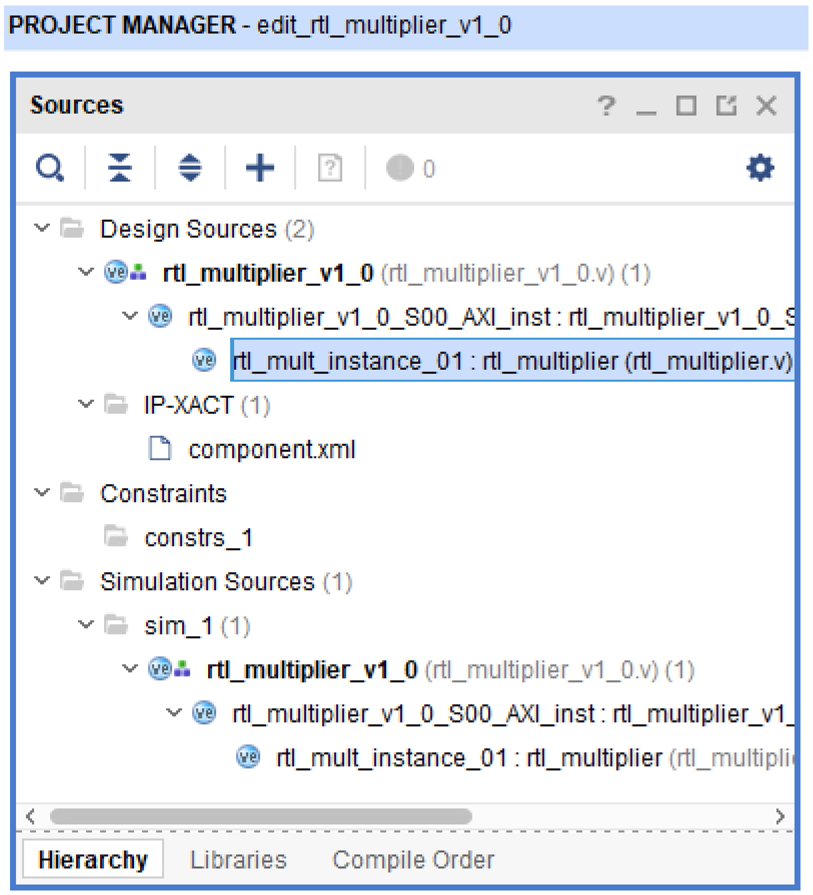
```
// Add user logic here
// wire to hold rtl_multiplier output
wire [31:0] rtl_multiplier_out;
// instantiate the rtl_multiplier
rtl_multiplier rtl_mult_instance_01(
    .clk(S_AXI_ACLK),
    .a(slv_reg0[31:16]),
    .b(slv_reg0[15:0]),
    .product(rtl_multiplier_out)
);
// User logic ends
```

- Finally, set the *rtl_multiplier_out* as the output of one of the AXI registers, as shown below.



```
	// Slave register read enable is asserted when valid address is available
	// and the slave is ready to accept the read address.
	assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
	always @(*)
	begin
	      // Address decoding for reading registers
	      case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
	        2'h0   : reg_data_out <= slv_reg0;
	        2'h1   : reg_data_out <= rtl_multiplier_out; //slv_reg1;
	        2'h2   : reg_data_out <= slv_reg2;
	        2'h3   : reg_data_out <= slv_reg3;
	        default : reg_data_out <= 0;
	      endcase
	end

	// Output register or memory read data
	always @( posedge S_AXI_ACLK )
	begin
	  if ( S_AXI_ARESETN == 1'b0 )
	    begin
	      axi_rdata  <= 0;
	    end
	  else
	    begin
	      // When there is a valid read address (S_AXI_ARVALID) with
	      // acceptance of read address by the slave (axi_arready),
	      // output the read dada
	      if (slv_reg_rden)
	        begin
	          axi_rdata <= reg_data_out;     // register read data
	        end
	    end
	end

	// Add user logic here
	// wire to hold rtl_multiplier output
```

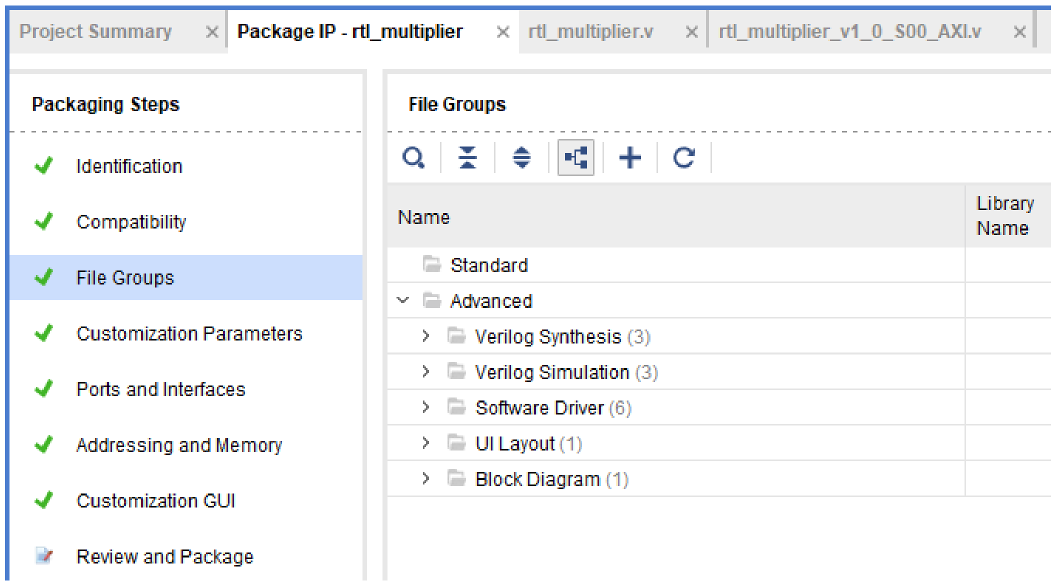
After saving the file(s) you will notice that the rtl_multiplier.v file has been integrated under the AXI file in the hierarchy. Good work so far! Almost there - just a few more rudimentary steps.


PROJECT MANAGER - edit_rtl_multiplier_v1_0

Sources

- Design Sources (2)
  - rtl_multiplier_v1_0 (rtl_multiplier_v1_0.v) (1)
    - rtl_multiplier_v1_0_S00_AXI_inst : rtl_multiplier_v1_0_S
      - rtl_mult_instance_01 : rtl_multiplier (rtl_multiplier.v)
  - IP-XACT (1)
    - component.xml
- Constraints
  - constrs_1
- Simulation Sources (1)
  - sim_1 (1)
    - rtl_multiplier_v1_0 (rtl_multiplier_v1_0.v) (1)
      - rtl_multiplier_v1_0_S00_AXI_inst : rtl_multiplier_v1_
        - rtl_mult_instance_01 : rtl_multiplier (rtl_multipli

- In the Package IP tab, click on File Groups and then on Merge changes from the File Groups Wizard. This will OK the File Groups step with a flattering green tick badge of great success. :)


Project Summary | Package IP - rtl_multiplier | rtl_multiplier.v | rtl_multiplier_v1_0_S00_AXI.v

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- Review and Package

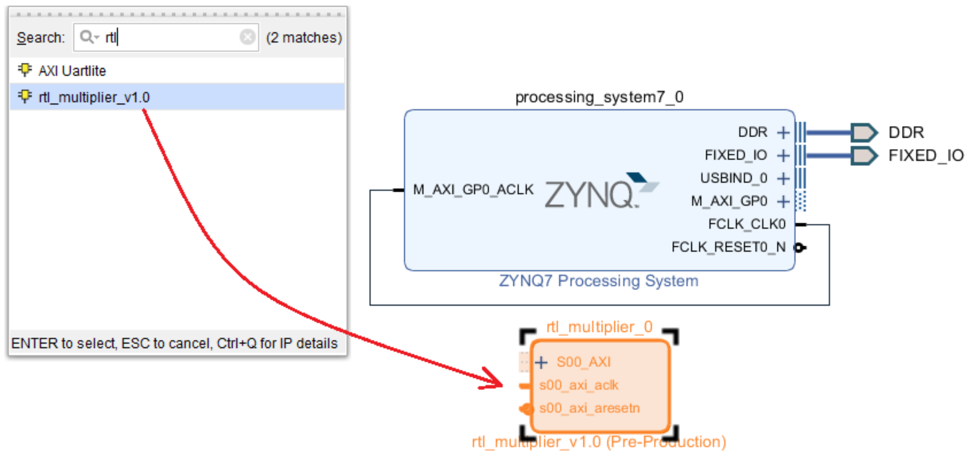
File Groups

Name	Library Name
Standard	
Advanced	
Verilog Synthesis (3)	
Verilog Simulation (3)	
Software Driver (6)	
UI Layout (1)	
Block Diagram (1)	

- Next, click on *Review and Package* and proceed with *Re-Package IP*. The IP block will be packaged and you can safely close the project.

4.3 Add the IP block to the Zynq PS System

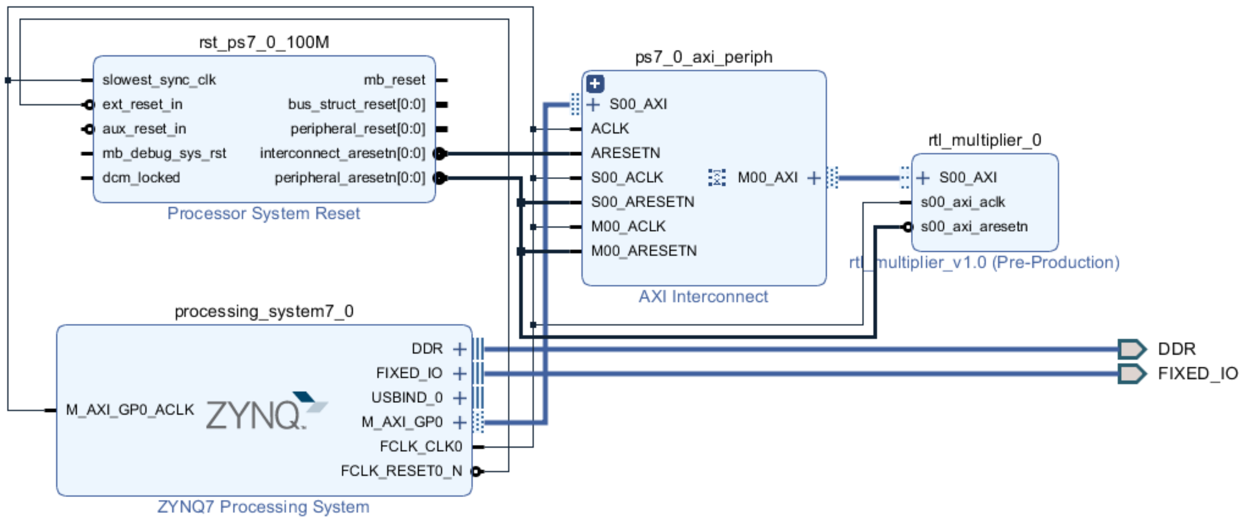
- In the original Vivado project containing the Zynq PS system, click on *Open Block Design* in the *IP Integrator* section to open the design.
- As in the previous lab, to browse for an IP block, click on the *Add IP* (+) button and search for our newly custom created *rtl_multiplier*. Double click it to add it to the design.


Search: rtl (2 matches)

- AXI Uartlite
- rtl_multiplier_v1.0

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

▪ Now let Vivado do the "magic" of connecting it to the ZYNQ7 PS by clicking on Run Connection Automation and use the default settings in the new dialog window.
The Connection Automation will add a few necessary intermediate IP blocks. It might look scary at first, but fear not! It's just a Processor System Reset and an AXI Interconnect here and there, no biggie. To make it more clear (and hopefully less scary), click on the Regenerate Layout (looks like a Refresh) button. You should see a "neat" design as shown below.

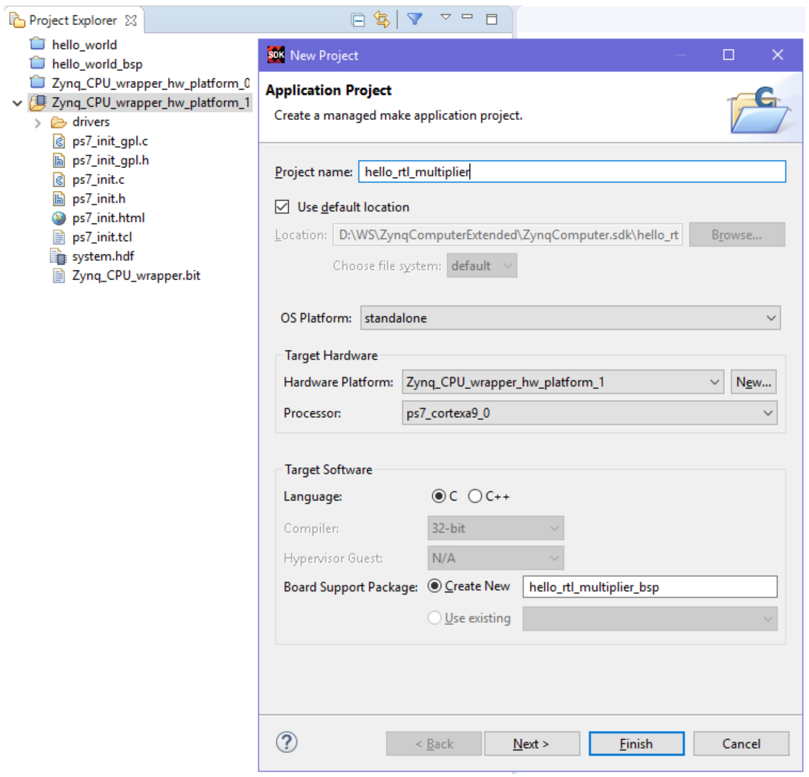


▪ This is all! Now, as in the previous tutorial in Section 3, save all files and click on Generate Bitstream. This will run through Synthesis, Implementation and will generate the bitstream file. Compilation takes a while - see the status in the upper-right corner. When completed, select Open Implemented Design. Next, export the hardware design to SDK including the bitstream and then lunch the SDK. In the Implementation reports, check the resource utilization and take a screenshot, and analyze if it is reasonable?

4.4 Interfacing with the IP Block in Software

So far we have the Zynq PS and the RTL Multiplier as an AXI Lite slave IP block all nicely interconnected, but it is all useless unless we use it in an application - so let's do that.

- Note: In case you have previous projects in the SDK, close them and start with the most recent exported hardware platform.
- Create a File -> New -> Application Project and give it a name as shown below.


Project Explorer

hello_world
hello_world_bsp
Zynq_CPU_wrapper_hw_platform_0
Zynq_CPU_wrapper_hw_platform_1
drivers
ps7_init_gpl.c
ps7_init_gpl.h
ps7_init.c
ps7_init.h
ps7_init.html
ps7_init.tcl
system.hdf
Zynq_CPU_wrapper.bit

New Project

Application Project

Create a managed make application project.

Project name: hello_rtl_multiplier

☑ Use default location

Location: D:\WS\ZynqComputerExtended\ZynqComputer.sdk\hello_rt

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: Zynq_CPU_wrapper_hw_platform_1

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] hello_rtl_multiplier_bsp

Use existing

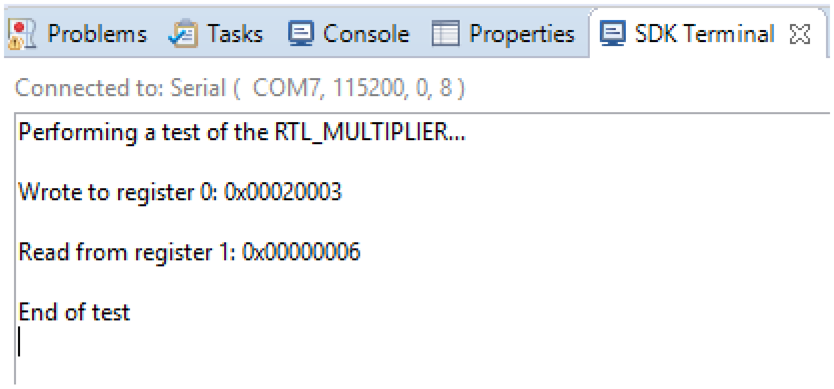
- In the next step, select the *Hello World* template and click *Finish*. The SDK will generate a new application which will appear in the *Project Explorer* as *hello_rtl_multiplier* and *hello_rtl_multiplier_bsp*.
  - *Optional*: Feel free to test that this part works by programming the FPGA, connecting the SDK Terminal and then running the default "Hello World" on the Arty Z7 board.
- Open the *hello_rtl_multiplier/src/helloworld.c* C-source file and update the file with the code provided below. Understand the code well - it's quite simple.

```
#include "platform.h"
#include "xbasic_types.h"
#include "xparameters.h" // Contains definitions for peripheral RTL_MULTIPLIER

// we will use the Base Address of the RTL_MULTIPLIER
Xuint32 *baseaddr_p = (Xuint32 *) XPAR_RTL_MULTIPLIER_0_S00_AXI_BASEADDR;

int main() {
    init_platform();
    xil_printf("Performing a test of the RTL_MULTIPLIER... \n\r");

    // Write multiplier inputs to register 0
    *(baseaddr_p + 0) = 0x00020003;
    xil_printf("Wrote to register 0: 0x%08x \n\r", *(baseaddr_p + 0));

    // Read multiplier output from register 1
    xil_printf("Read from register 1: 0x%08x \n\r", *(baseaddr_p + 1));

    xil_printf("End of test\n\n\r");

    cleanup_platform();
    return 0;
}
```

Save the updated files and run the application on the Arty Z7 board as described in the previous tutorial.

- Program the FPGA
- Connect the SDK Terminal to the serial port
- *Run As -> 4 Launch on Hardware (GDB)*

You should see a number of warnings, but also the successful completion of the test in the SDK Terminal, as shown below.



```
Connected to: Serial ( COM7, 115200, 0, 8 )
Performing a test of the RTL_MULTIPLIER...

Wrote to register 0: 0x00020003

Read from register 1: 0x00000006

End of test
```

4.4 Follow-up assignment

implement an IP block as an AXI Lite Slave that takes in two numbers as arguments and returns the div and mod of them.

References:

1. Arty Z7 Reference Manual https://digilent.com/reference/programmable-logic/arty-z7/reference-manual
2. Get familiar with Arty 7 https://digilent.com/reference/programmable-logic/arty-z7/start
3. Getting Started with the Vivado IP Integrator https://digilent.com/reference/vivado/getting-started-with-ipi/2018.2
4. AXI: Canvas>File>Reading Materials>Zynq> ug761_axi_reference_guide.pdf.

Acknowledgement:

- Sergiu Masanu, Mircea Stan (U of Virginia)
- Andreas Gerstlauer (U of Austin)
- Digilent
- Xilinx University Program

Deliverables:

- Group Deliverables (Compile everything as a single pdf report besides the code):
  - Based on **part 3**, please submit the screenshot of
    - the Eclipse environment
    - the message Hello World in the SDK Terminal.
    - The SDK terminal where it outputs your group name and members' names
    - The modified C code
  - Based on **part 4**, please submit the following
    - Resource utilization report/screenshot of your multiplier
    - Screenshot of your SDK Terminal for the multiplier
    - Resource utilization report/screenshot of your divider
    - SDK Terminal of your SDK Terminal for the divider
    - The divider source code (Verilog or VHDL)
  - Answer the following questions (based on your understanding, feel free to use the internet):
    - What is a bare metal test?
    - How does the Zynq PS communicate with the IP blocks we created in this lab?
    - Briefly read out the AXI, and summarize the main features.
    - Overall, what did you learn from this lab? Any difficulties?
- Individual Deliverables
  - Complete the peer-evaluation form appended in the end of this document

Grading Policy

Factor	**Percentage**
Part 3	40%
Part 4	50%
Questions	10%

JOINT INSTITUTE
交大密西根学院

ECE4810J System-on-Chip (SoC) Design

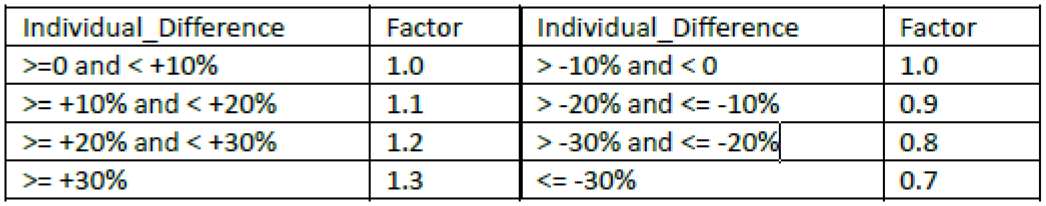
Fall 2021

Lab #1 Peer Evaluation Form

Each team member is required to provide a peer evaluation for the team effort of the lab. The score of the peer evaluation should be integers ranging between 0 to 5, inclusively, with 5 indicating the biggest contribution. A score should be given to each team member including yourself according the team member's contribution based on your observation. A brief description of contribution of each team member should also be provided, as shown in the following table.

Name	Level of contribution (0 – 5)	Description of contribution
(yourself)		
Team member 1		
Team member 2		
Team member 3		

Your lab grade is calculated based on the following:
Individual_Average = (sum of all team member's marks) / (size of the student team)
Group_Average = sum of Individual_Average of all team members / size of student team
Individual_Difference = Individual_Average / Group_Average – 1.0
Using the calculated Individual_Difference, we find a factor from the following lookup table.



Individual_Difference	Factor	Individual_Difference	Factor
>=0 and < +10%	1.0	> -10% and < 0	1.0
>= +10% and < +20%	1.1	> -20% and <= -10%	0.9
>= +20% and < +30%	1.2	> -30% and <= -20%	0.8
>= +30%	1.3	<= -30%	0.7

Your final lab grade is calculated by :
Final grade of lab = points for team effort * factor
Note that the final grade of the lab will not exceed the maximum points assigned to the project. If the peer-evaluation form is not submitted, then your individual_difference will be based on the available data (without yours) only, and your final score will be:
Final grade of lab = points for team effort * factor * 95%