EE3220: System On-Chip (SoC) Design

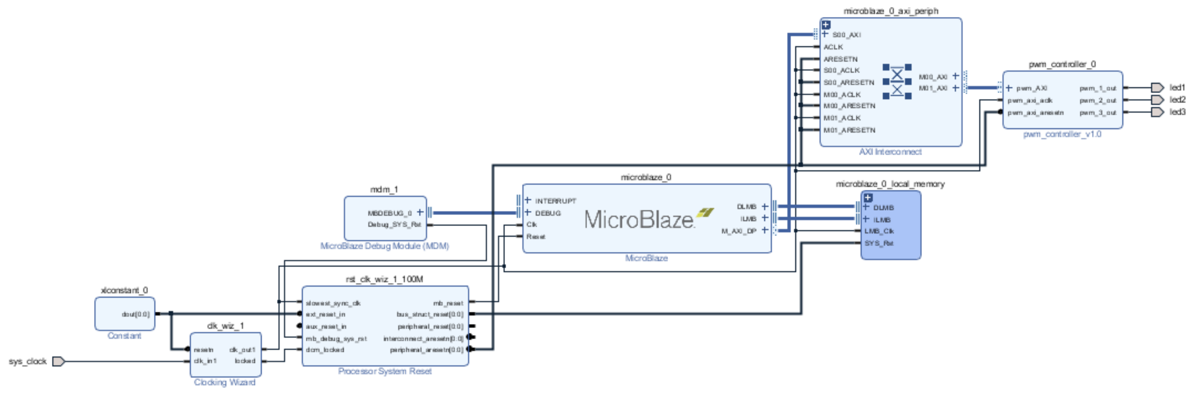
Laboratory 4: Creating custom IP for MicroBlaze and PWM

Objectives:

By the end of this lab, student should be able to:

- Create a custom IP in Vivado
- Integrate the IP with MicroBlaze soft processor
- Generate pulse width modulation (PWM) signal with FPGA
- Use PWM to display different colors on RGB LED

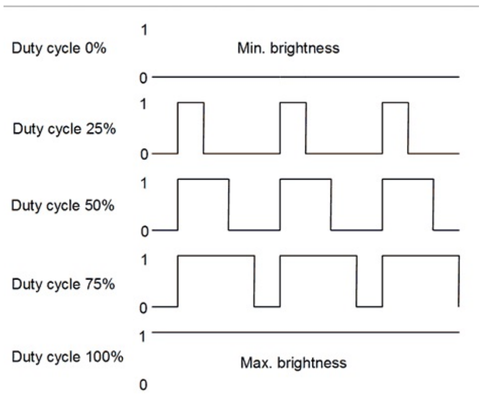
System overview:



In this exercise, we are going to create and build a custom FPGA PWM IP in Vivado. The IP will be connected to MicroBlaze microcontroller. The system will be synthesized and implemented after which the bitstream will be generated. Using the Vitis IDE, we will write a C program to display different colors with the RGB LED on the board. There will be three check points total, take enough snapshots as evidence of the checkpoints to be attached in the lab report.

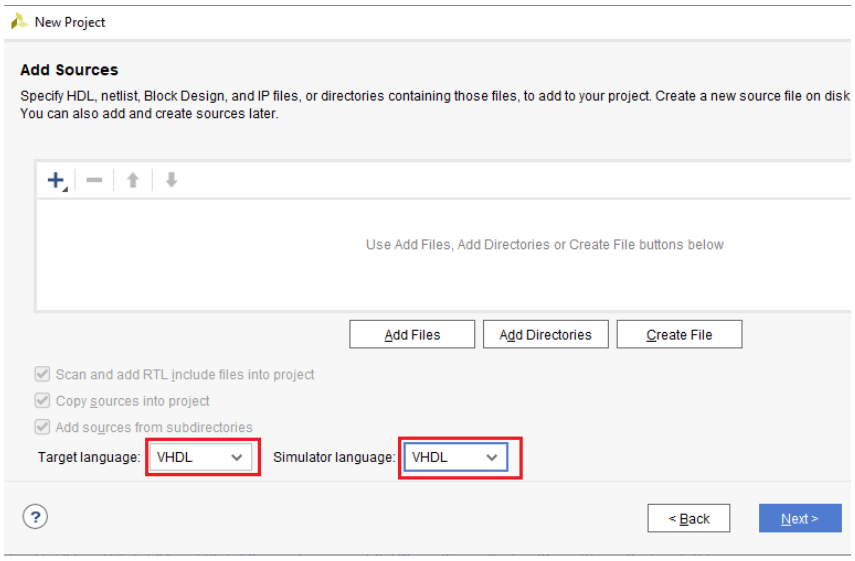
Part 1: Create and Package a custom PWM Controller IP in Vivado

In this section, student will use Vivado hlx to create a custom pwm controller IP as an AXI peripheral in Vivado. Pulse width modulation (PWM) is a method of reducing the average power delivered to an electronic load by using an on-off digital signal. It plays a vital role in many applications such as motor control and inverters. In PWM signal generator, the duty cycle can be defined as the portion of the time period where the device or system is operating, i.e. Duty cycle= T_{on}/T. In this equation, T_{on}=the time length the pulse is high (1) and T is the period, i.e. the time length of a pulse. By controlling the T_{on} or equivalently changing the Duty cycle, we can thus change the average power delivery to a load, like LED. Using PWM different colors could be displayed on an RGB LED.



Now follow the following steps to achieve part 1:

1. Create new project and name it **lab4**. At the **Add Sources** stage, set the **Target language** and the **Simulator language** to **VHDL** as indicated below.



New Project

Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk
You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

Add Files | Add Directories | Create File

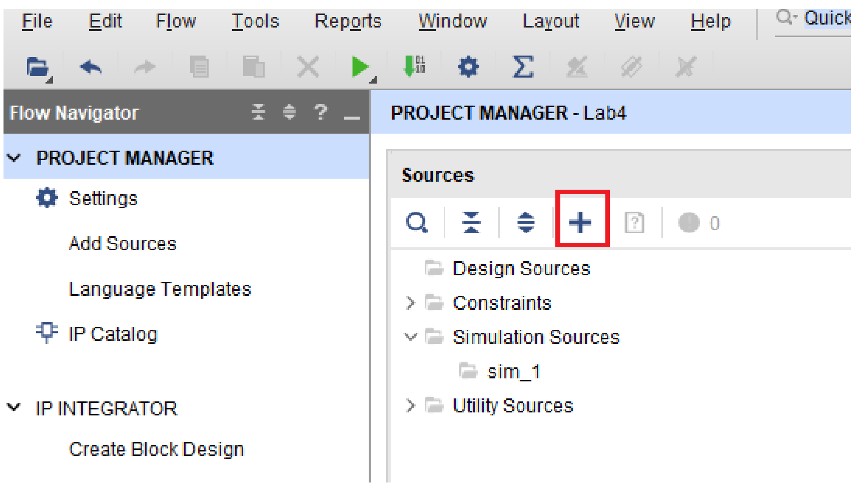
☑ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

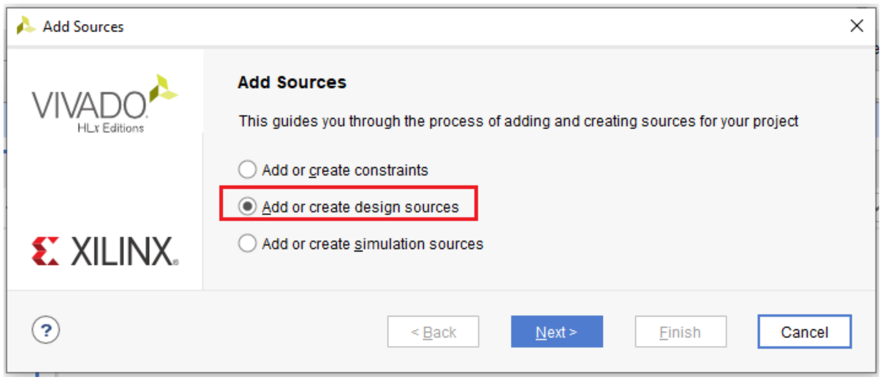
Target language: VHDL   Simulator language: VHDL

< Back   Next >

2. Create new source file

- Select the **'+'** to add new source under source window
- Select **Add or create design sources**, click Next



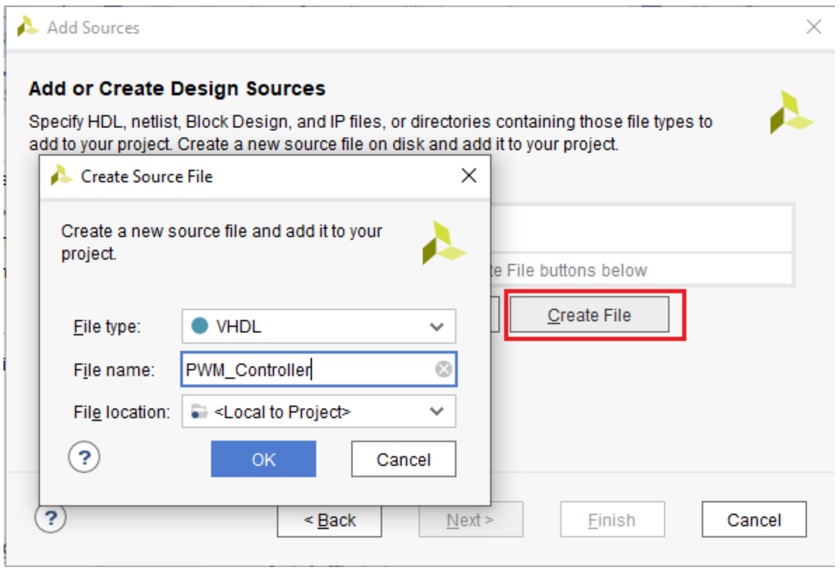

Add Sources

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

- Click **Create File**, select **VHDL** as the File Type and enter the File name as **“PWM_Controller”**. Click **OK** and then **Finish**.


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.

Create Source File

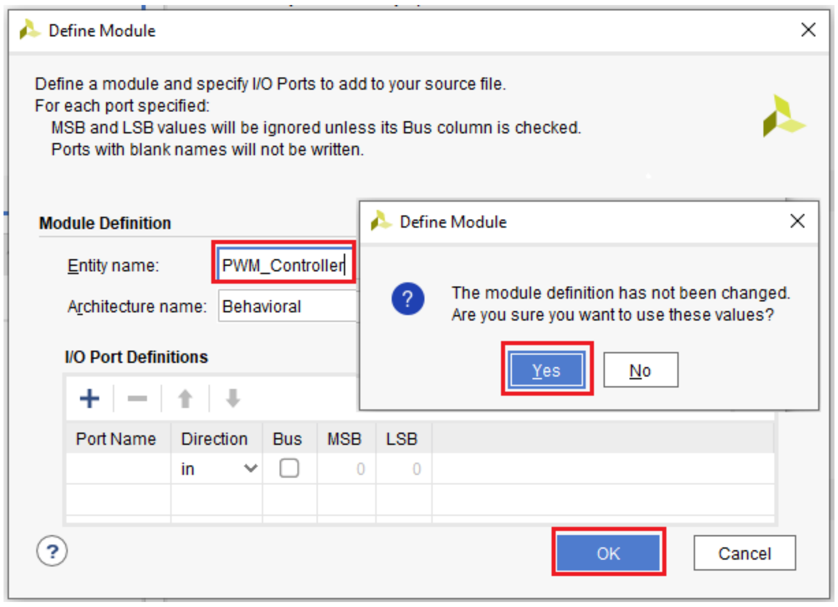
Create a new source file and add it to your project.

File type: VHDL

File name: PWM_Controller

File location: <Local to Project>

- Click **OK**, a dialogue appears, click **OK** then **Yes** then **Finish**. This creates a new source file


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name: PWM_Controller

Architecture name: Behavioral

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	in	☐	0	0

Define Module

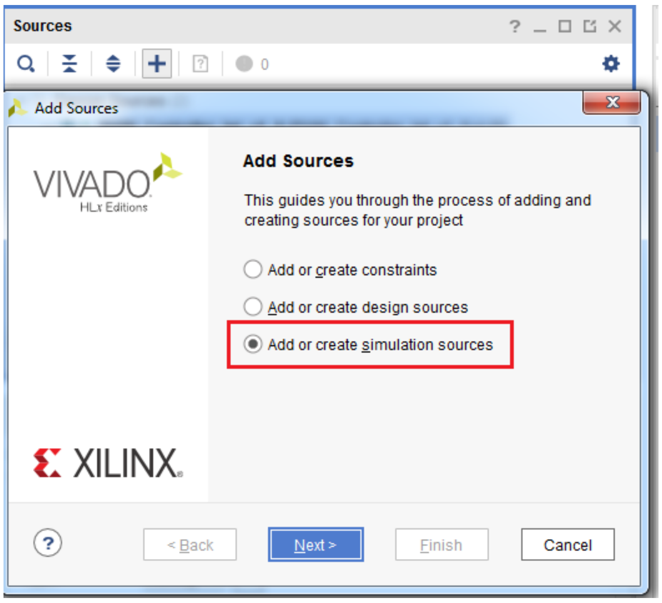
The module definition has not been changed.
Are you sure you want to use these values?

- Open the source file, copy and paste the VHDL program provided below and save.

[Note: In this design, the input signals include, clk, reset, duty cycle counter and output pwm signals]

```vhdl
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity PWM_Controller is
-- Port ( );
port(
  clk:      in std_logic;
  rst:      in std_logic;

  pwm_1_enable:  in std_logic;
  pwm_2_enable:  in std_logic;
  pwm_3_enable:  in std_logic;

  pwm_1_dc_counter: in unsigned(7 downto 0);
  pwm_2_dc_counter: in unsigned(7 downto 0);
  pwm_3_dc_counter: in unsigned(7 downto 0);

  pwm_1_out:    out std_logic;
  pwm_2_out:    out std_logic;
  pwm_3_out:    out std_logic
 );

end PWM_Controller;

architecture Behavioral of PWM_Controller is

  signal fr_counter : unsigned(7 downto 0) := "00000000";

-------- Increment register from 0 to 255 and reset ---------

 begin
    Counter_proc: process(clk)begin

      if( rst ='1') then
        fr_counter <= "00000000";
      elsif(rising_edge(clk)) then
        fr_counter <= fr_counter + "00000001";
      end if;
    end process;

 pwm_1_out <= '1' when((fr_counter < pwm_1_dc_counter) and (pwm_1_enable='1'))
else '0';
 pwm_2_out <= '1' when((fr_counter < pwm_2_dc_counter) and (pwm_2_enable='1'))
else '0';
 pwm_3_out <= '1' when((fr_counter < pwm_3_dc_counter) and (pwm_3_enable='1'))
else '0';

end Behavioral;
```

- We can first simulate the PWM_Controller design to verify its functionality.
- Click **'+'** to add new source and select **Add or create simulation sources**. Click Next.

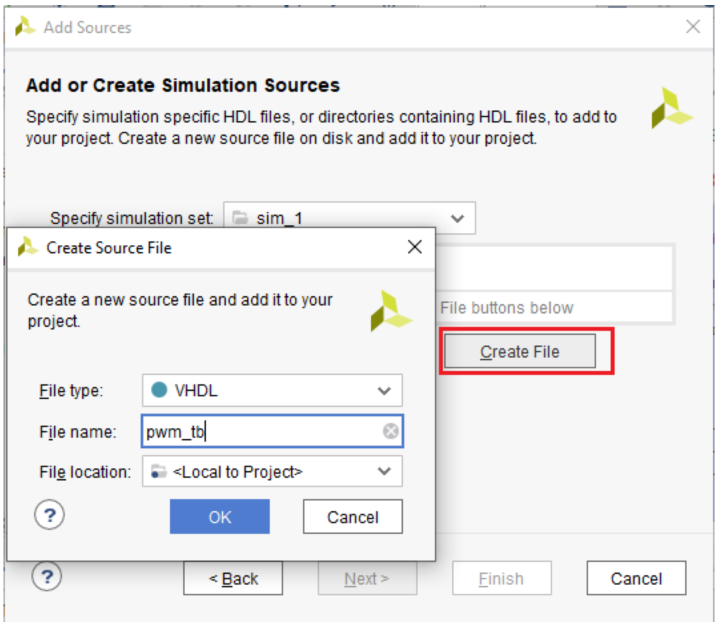


Add Sources

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

- Click **Create File**, specify the file type as **VHDL**, file name, and location. Then click **OK** and then **Finish**.



Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Create Source File

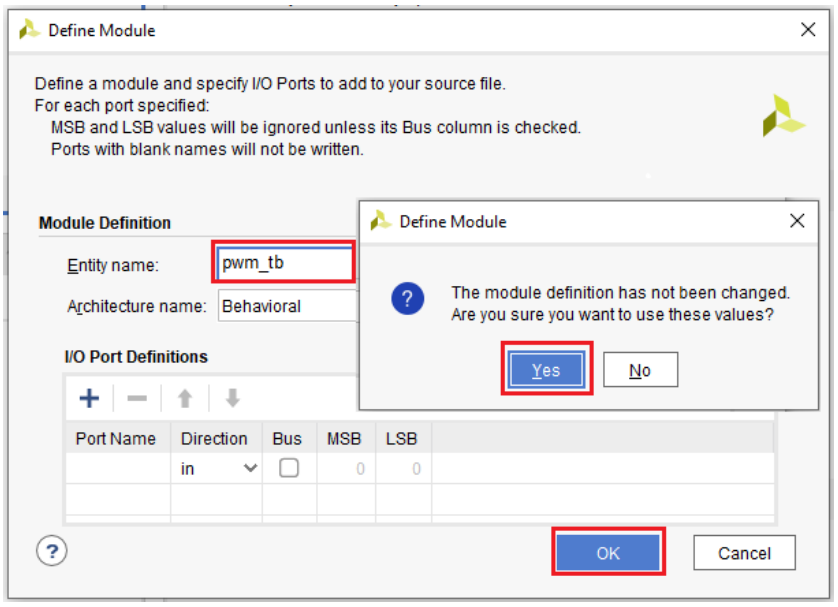
Create a new source file and add it to your project.

File type: VHDL

File name: pwm_tb

File location: <Local to Project>

- Click **OK** and **Yes**


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name: pwm_tb
Architecture name: Behavioral

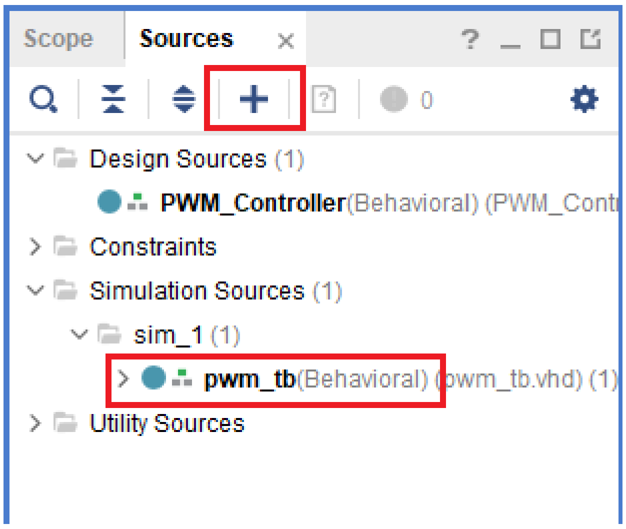
I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	in	☐	0	0

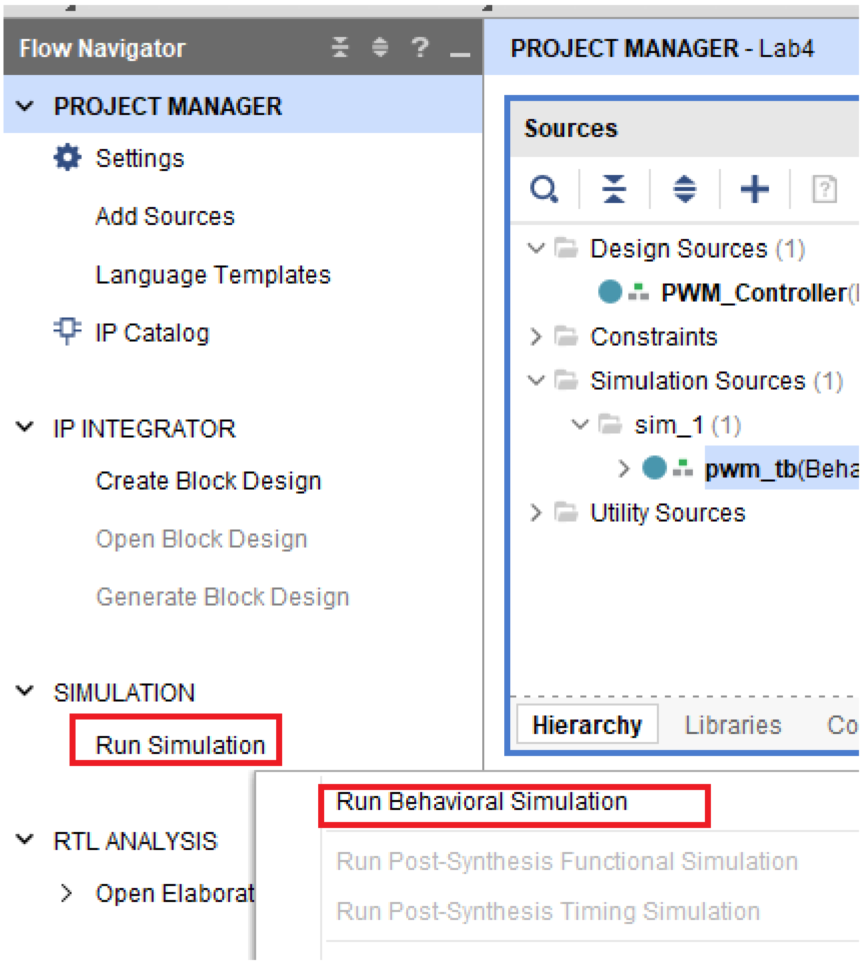
Define Module

The module definition has not been changed.
Are you sure you want to use these values?

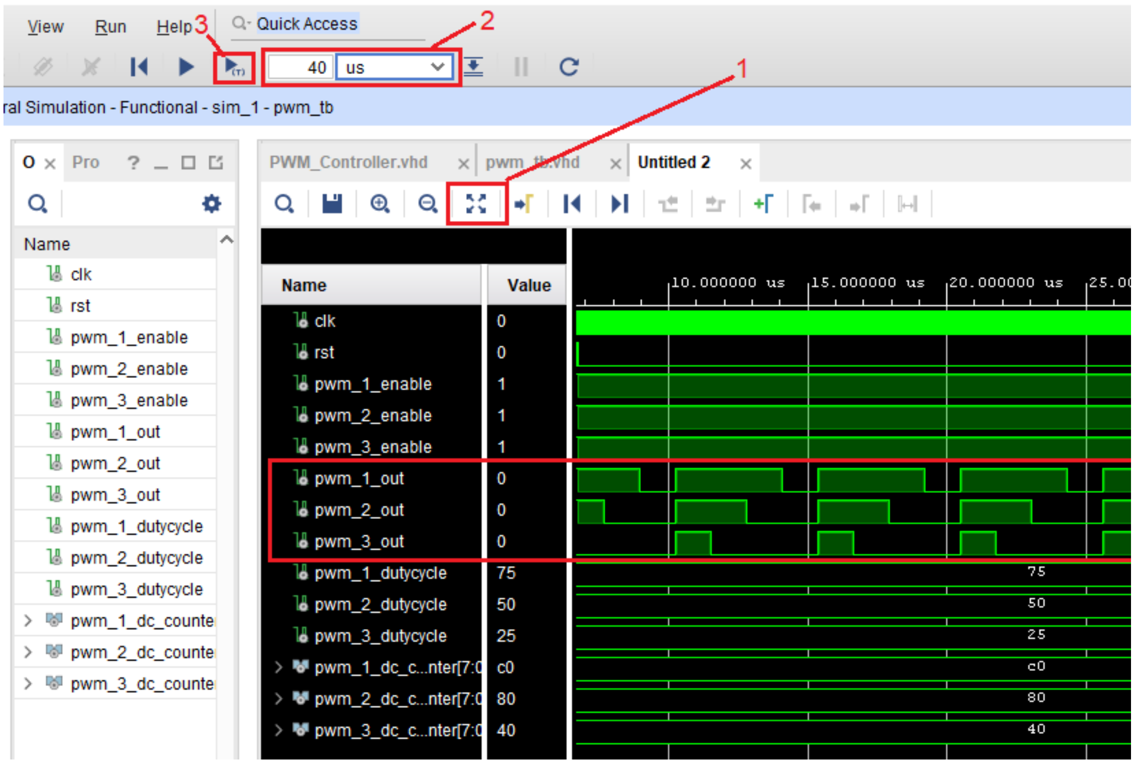
- In sources panel, under **Simulation sources/sim_1/**, double click the **pwm_tb** file created to open it.



- Copy and paste the test bench code provided to verify the PWM controller module.
- In the flow navigator, click **run simulation** and **Run Behavioral Simulation**



- In the simulation window, click the tab labelled as **1**, then set **tab 2** to 40 us and click **tab 3** to run the simulation



Check Point 1:

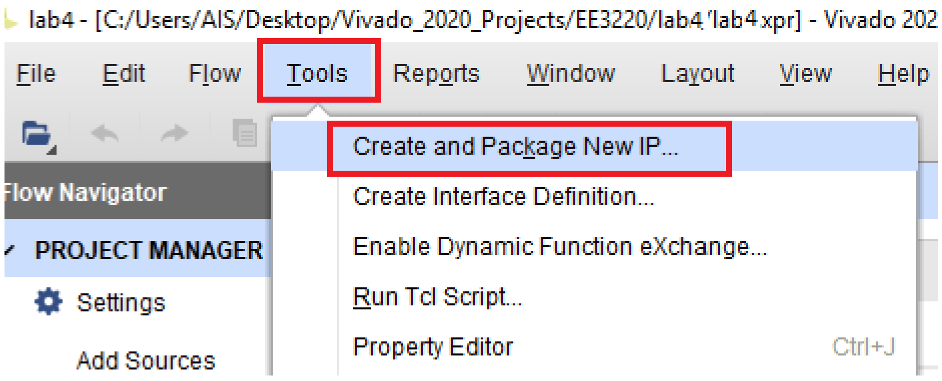
Modify the duty cycles to 20%, 40% and 60% and observe the change in the pulse widths. Take snapshots as evidence of the checkpoint achievement. Proceed to the next task.

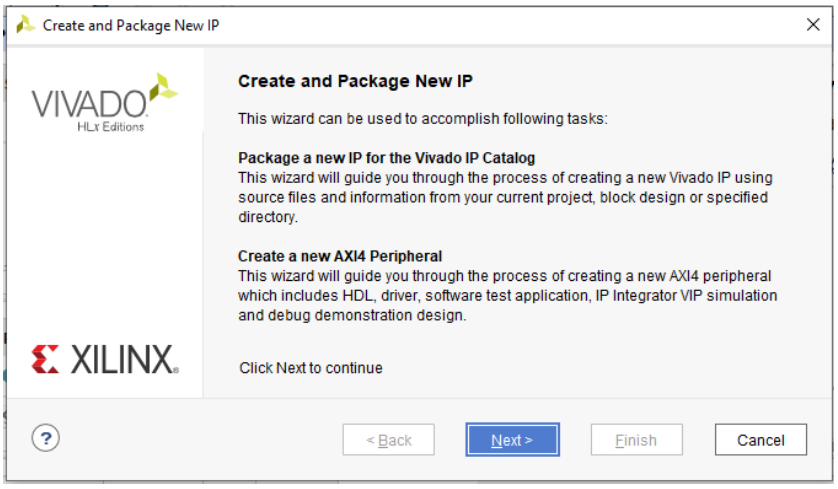
The test bench code:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity pwm_tb is
-- Port ( );
end pwm_tb;

architecture Behavioral of pwm_tb is

signal clk:       std_logic :='0';
signal rst:       std_logic :='1';

signal pwm_1_enable:   std_logic ;
signal pwm_2_enable:   std_logic ;
signal pwm_3_enable:   std_logic ;

signal pwm_1_out:    std_logic;
signal pwm_2_out:    std_logic;
signal pwm_3_out:    std_logic;

signal pwm_1_dutycycle: integer := 75;     --75%  duty cycle
signal pwm_2_dutycycle: integer := 50;     --50%  duty cycle
signal pwm_3_dutycycle: integer := 25;     --25%  duty cycle

signal pwm_1_dc_counter: unsigned(7 downto 0) := to_unsigned((pwm_1_dutycycle *
256 /100),8);
signal pwm_2_dc_counter: unsigned(7 downto 0) := to_unsigned((pwm_2_dutycycle *
256 /100),8);
signal pwm_3_dc_counter: unsigned(7 downto 0) := to_unsigned((pwm_3_dutycycle *
256 /100),8);

component PWM_Controller is

port(
  clk:      in std_logic;
  rst:      in std_logic;

  pwm_1_enable:  in std_logic;
  pwm_2_enable:  in std_logic;
  pwm_3_enable:  in std_logic;

  pwm_1_dc_counter: in unsigned(7 downto 0);
  pwm_2_dc_counter: in unsigned(7 downto 0);
  pwm_3_dc_counter: in unsigned(7 downto 0);

  pwm_1_out:    out std_logic;
  pwm_2_out:    out std_logic;
  pwm_3_out:    out std_logic
);
end component;

begin

-- instantiate pwm controller

pwm_contr: PWM_Controller

 port map
 (
  --/**** input ports ************/

  clk        =>(clk),
  rst        =>(rst),

  pwm_1_enable   =>(pwm_1_enable),
  pwm_2_enable   =>(pwm_2_enable),
  pwm_3_enable   =>(pwm_3_enable),

  pwm_1_dc_counter =>(pwm_1_dc_counter),
  pwm_2_dc_counter =>(pwm_2_dc_counter),
  pwm_3_dc_counter =>(pwm_3_dc_counter),

--/*** output ports ************/

  pwm_1_out     =>(pwm_1_out),
  pwm_2_out     =>(pwm_2_out),
  pwm_3_out     =>(pwm_3_out)

 );

--// routines

clk      <= not clk after 10ns; --half_period
rst      <= '0' after   20ns;

 pwm_1_enable <= '1';      -- enable pwm 1
 pwm_2_enable <= '1';
 pwm_3_enable <= '1';

end Behavioral;
```

3. Create an empty Axi4 peripheral IP

- In Vivado, click on **Tools->Create and Package New IP**. Click Next


lab4 - [C:/Users/AIS/Desktop/Vivado_2020_Projects/EE3220/lab4/lab4.xpr] - Vivado 202
File
Edit
Flow
Tools
Reports
Window
Layout
View
Help
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Create and Package New IP...
Create Interface Definition...
Enable Dynamic Function eXchange...
Run Tcl Script...
Property Editor Ctrl+J


Create and Package New IP

Create and Package New IP

This wizard can be used to accomplish following tasks:

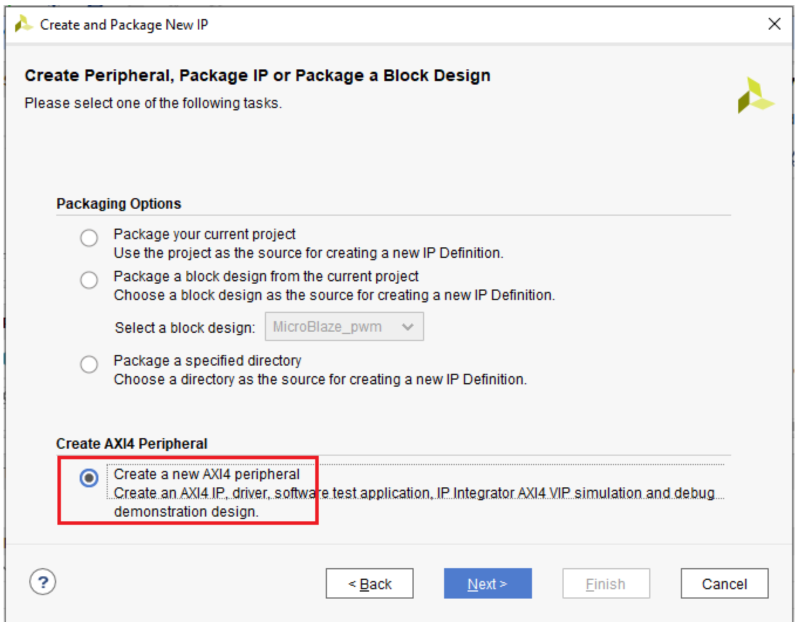
Package a new IP for the Vivado IP Catalog
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.

Create a new AXI4 Peripheral
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.

Click Next to continue

< Back | Next > | Finish | Cancel

- Choose **Create a new AXI4 peripheral** and click **Next**


Create and Package New IP

Create Peripheral, Package IP or Package a Block Design
Please select one of the following tasks.

Packaging Options

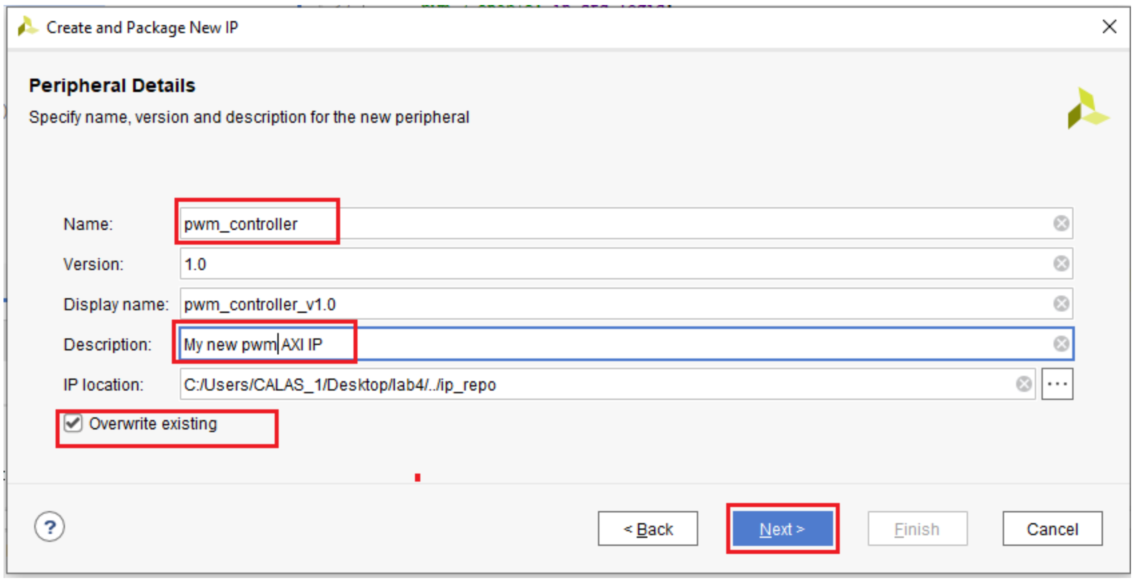
- ○ Package your current project
  Use the project as the source for creating a new IP Definition.
- ○ Package a block design from the current project
  Choose a block design as the source for creating a new IP Definition.
  Select a block design: MicroBlaze_pwm
- ○ Package a specified directory
  Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

- [✓ Create a new AXI4 peripheral]
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

< Back | Next > | Finish | Cancel

- Specify the IP name as **pwm_controller** and click **Next**



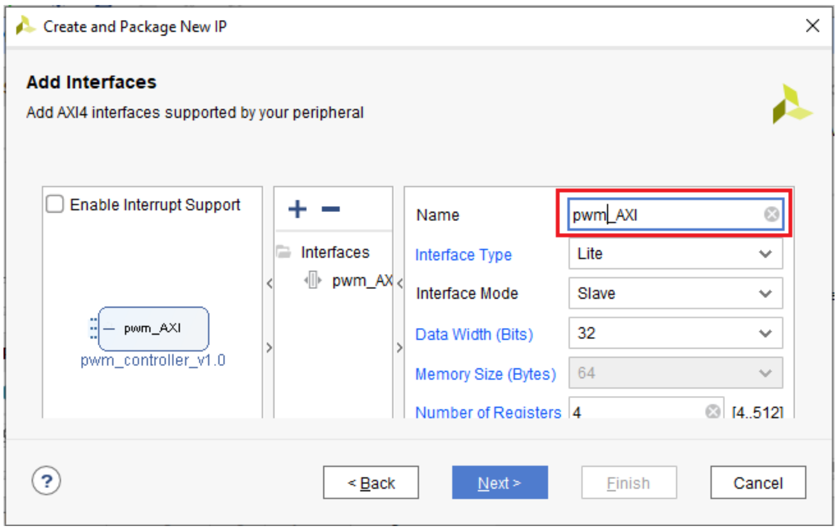
Peripheral Details

Specify name, version and description for the new peripheral

Name:	pwm_controller
Version:	1.0
Display name:	pwm_controller_v1.0
Description:	My new pwm AXI IP
IP location:	C:/Users/CALAS_1/Desktop/lab4/../ip_repo

[✓] Overwrite existing

- Name interface as **pwm_AXI**. Leave the rest as default and click **Next**



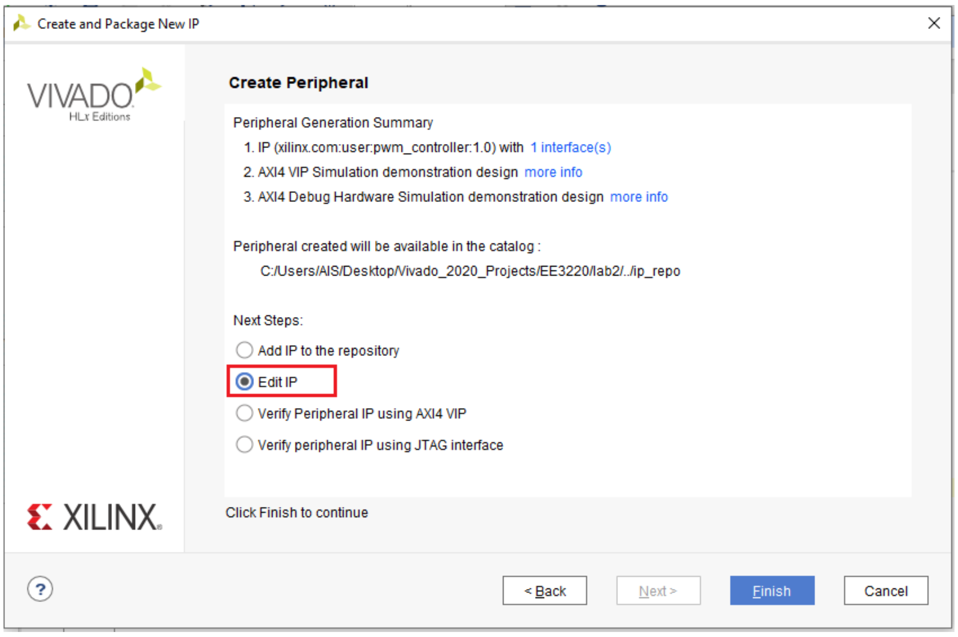
Add Interfaces

Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

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

- Select **Edit IP** and click **Finish**, wait for Vivado to create and open a new project for the IP.



Create Peripheral

Peripheral Generation Summary

1. IP (xilinx.com:user:pwm_controller:1.0) with 1 interface(s)
2. AXI4 VIP Simulation demonstration design more info
3. AXI4 Debug Hardware Simulation demonstration design more info

Peripheral created will be available in the catalog :
C:/Users/AIS/Desktop/Vivado_2020_Projects/EE3220/lab2/../ip_repo

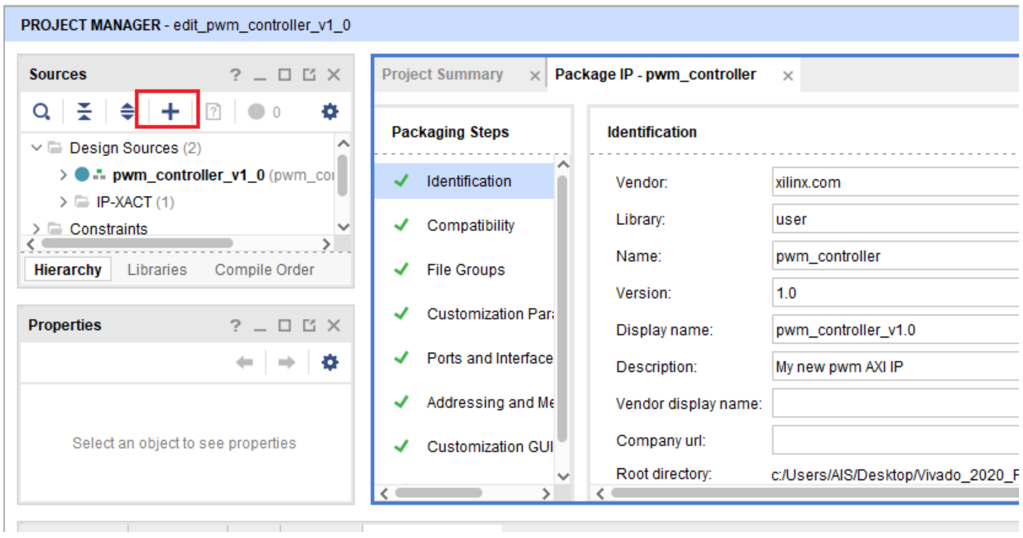
Next Steps:

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

Click Finish to continue

4. Edit the IP by adding our previously created vhdl source code.

- Click add sources **(+)** button.

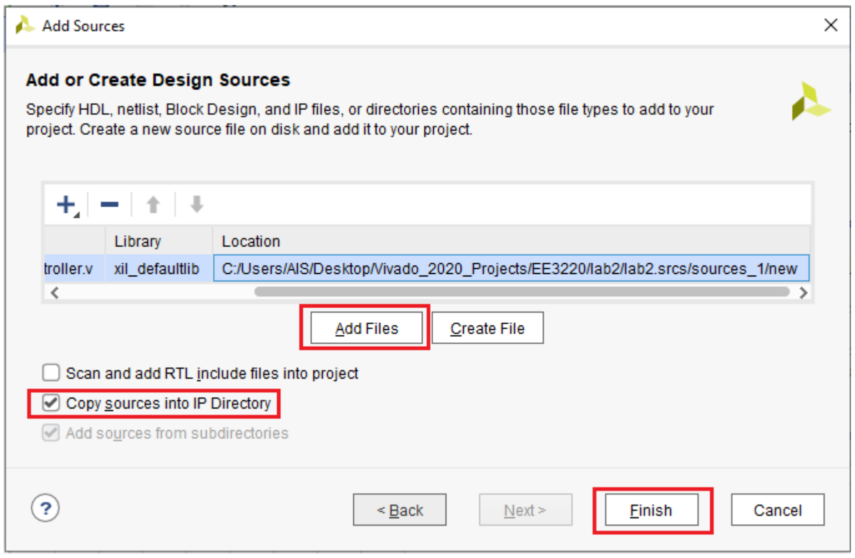


Identification

Vendor: xilinx.com
Library: user
Name: pwm_controller
Version: 1.0
Display name: pwm_controller_v1.0
Description: My new pwm AXI IP
Vendor display name:
Company url:
Root directory: c:/Users/AIS/Desktop/Vivado_2020_F

- Then, choose **Add Files**, add the PWM generator design source you previously created and click Finish. Remember to click, copy sources into IP directory. The path to file is:

  <Your Project folder>\lab4.srcs\sources_1\new\PWM_Controller.vhd



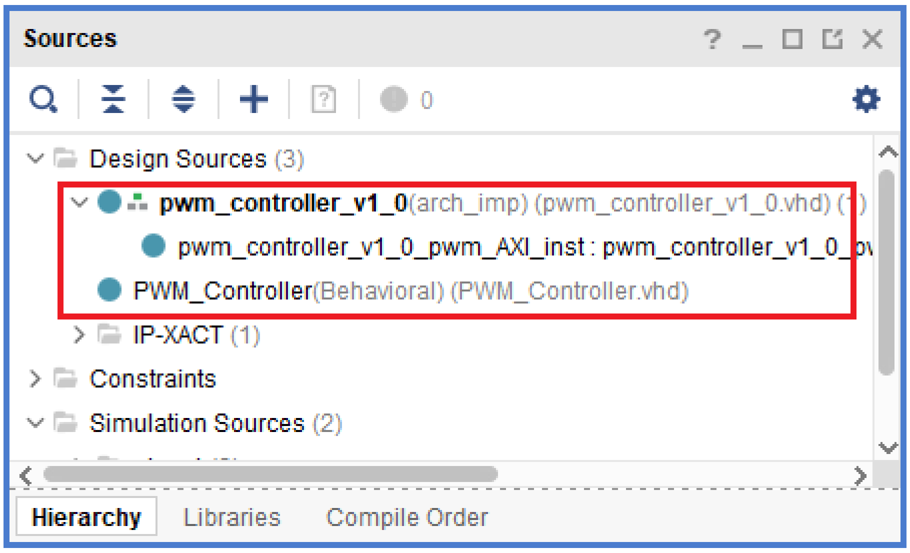
Add or Create Design Sources

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.

	Library	Location
troller.v	xil_defaultlib	C:/Users/AIS/Desktop/Vivado_2020_Projects/EE3220/lab2/lab2.srcs/sources_1/new

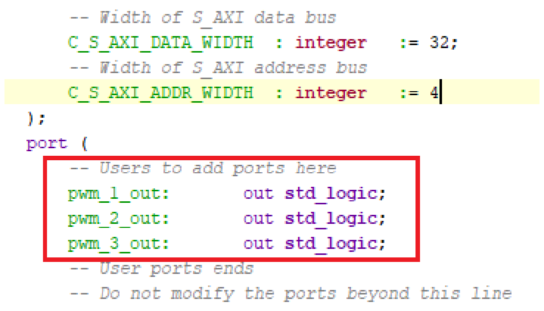
☐ Scan and add RTL include files into project
☑ Copy sources into IP Directory
☑ Add sources from subdirectories

- After you added the file, ensure you have 3 source files in the sources panel.


Sources
Design Sources (3)
pwm_controller_v1_0(arch_imp) (pwm_controller_v1_0.vhd) (1)
pwm_controller_v1_0_pwm_AXI_inst : pwm_controller_v1_0_p
PWM_Controller(Behavioral) (PWM_Controller.vhd)
IP-XACT (1)
Constraints
Simulation Sources (2)
Hierarchy
Libraries
Compile Order

- "**pwm_controller_v1_0**" is the wrapper. It contains Instantiation of AXI Bus Interface, and the user logic, which implement the function of the IP.
- "**PWM_Controller_v1_0_pwm_AXI_inst**" contains the axi lite bus interface.
- "**PWM_Controller**" contains the logic of a PWM controller.
- Open the **PWM_Controller_v1_0_pwm_AXI_inst.vhd**, in the port declaration part, we need to add our own input/output ports as well as instantiate our created source file module. Now find **-- Users to add ports here**. Add our pwm outputs as follows:

```
    pwm_1_out:     out std_logic;
    pwm_2_out:     out std_logic;
    pwm_3_out:     out std_logic;
```

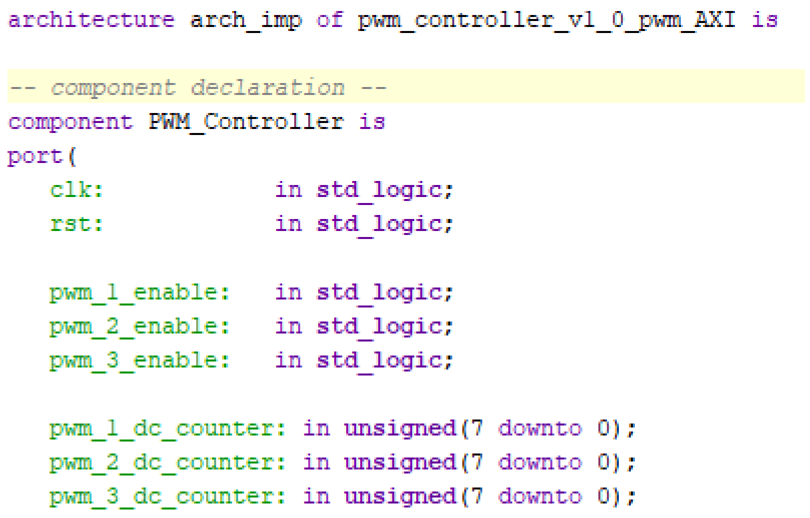


```
        -- Width of S_AXI data bus
        C_S_AXI_DATA_WIDTH  : integer   := 32;
        -- Width of S_AXI address bus
        C_S_AXI_ADDR_WIDTH  : integer   := 4
    );
    port (
        -- Users to add ports here
        pwm_1_out:          out std_logic;
        pwm_2_out:          out std_logic;
        pwm_3_out:          out std_logic;
        -- User ports ends
        -- Do not modify the ports beyond this line
```

For the rest of the inputs, we will utilize the IP's registers to get the input data so no need to set them as inputs here.

- Scroll down and find **architecture arch_imp of pwm_controller_v1_0_pwm_AXI is**. Under this line declare the pwm component as below:

```
-- component declaration --

component PWM_Controller is
port(

  clk:           in std_logic;
  rst:           in std_logic;

  pwm_1_enable:  in std_logic;
  pwm_2_enable:  in std_logic;
  pwm_3_enable:  in std_logic;
  pwm_1_dc_counter: in unsigned(7 downto 0);
  pwm_2_dc_counter: in unsigned(7 downto 0);
  pwm_3_dc_counter: in unsigned(7 downto 0);

  pwm_1_out:     out std_logic;
  pwm_2_out:     out std_logic;
  pwm_3_out:     out std_logic

 );
end component;
```



```
architecture arch_imp of pwm_controller_v1_0_pwm_AXI is

-- component declaration --
component PWM_Controller is
port(
   clk:            in std_logic;
   rst:            in std_logic;

   pwm_1_enable:   in std_logic;
   pwm_2_enable:   in std_logic;
   pwm_3_enable:   in std_logic;

   pwm_1_dc_counter: in unsigned(7 downto 0);
   pwm_2_dc_counter: in unsigned(7 downto 0);
   pwm_3_dc_counter: in unsigned(7 downto 0);
```

- Find **-- Add user logic here**, instantiate the **pwm_controller** module here as follows:

```
-- instantiate pwm controller
-- Add user logic here
 pwm_contr: PWM_Controller
 port map
 (
  --/**** input ports ************/--

  clk         =>(S_AXI_ACLK),
  rst         =>(S_AXI_ARESETN),

  pwm_1_enable    =>(slv_reg0(0)),
  pwm_2_enable    =>(slv_reg0(1)),
  pwm_3_enable    =>(slv_reg0(2)),

  pwm_1_dc_counter =>unsigned(slv_reg1(7 downto 0)),
  pwm_2_dc_counter =>unsigned(slv_reg2(7 downto 0)),
  pwm_3_dc_counter =>unsigned(slv_reg3(7 downto 0)),

--/*** output ports ************/--

  pwm_1_out      =>(pwm_1_out),
  pwm_2_out      =>(pwm_2_out),
  pwm_3_out      =>(pwm_3_out)
 );
```

Finally, we need to modify the top module too as follows:

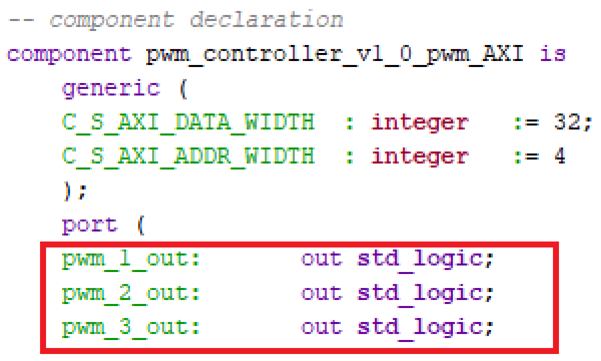
- Open the top module design source **"PWM_Controller_v1_0"**. In the port declaration part, find **-- Users to add ports here** and add our pwm outputs as previously done:

```
      pwm_1_out:     out std_logic;
      pwm_2_out:     out std_logic;
      pwm_3_out:     out std_logic;
```

- Add the pwm output port to the pwm_AXI component declaration.
- Find **component pwm_controller_v1_0_pwm_AXI is** " under port, add the below:

```
        pwm_1_out:     out std_logic;
        pwm_2_out:     out std_logic;
        pwm_3_out:     out std_logic;
```

[Note: Compare with the picture below and ensure it is correct]

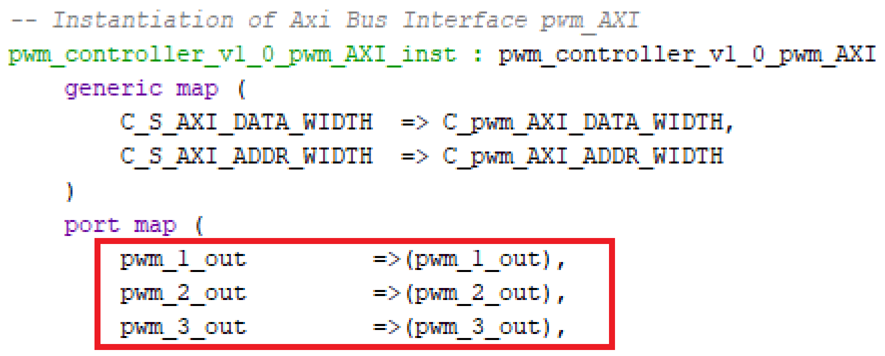


```
-- component declaration
component pwm_controller_v1_0_pwm_AXI is
    generic (
    C_S_AXI_DATA_WIDTH  : integer   := 32;
    C_S_AXI_ADDR_WIDTH  : integer   := 4
    );
    port (
    pwm_1_out:        out std_logic;
    pwm_2_out:        out std_logic;
    pwm_3_out:        out std_logic;
```

- Add the pwm output ports to the pwm_AXI instance. Find **-- Instantiation of Axi Bus Interface pwm_AXI**, under "port map" write :

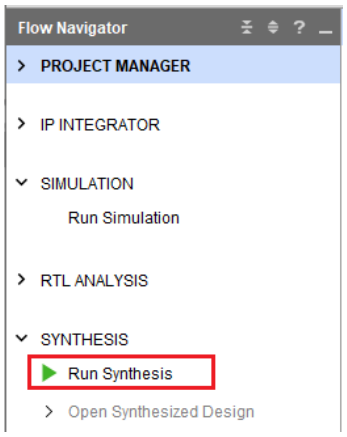
```
  -- the added pwm output ports
        pwm_1_out      =>(pwm_1_out),
        pwm_2_out      =>(pwm_2_out),
        pwm_3_out      =>(pwm_3_out),
```

[Note: Compare with the picture below and ensure it is correct]

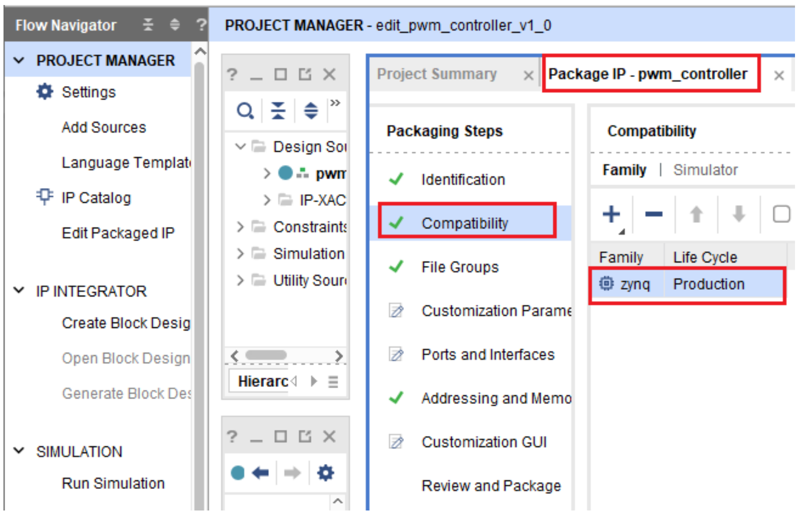


```
-- Instantiation of Axi Bus Interface pwm_AXI
pwm_controller_v1_0_pwm_AXI_inst : pwm_controller_v1_0_pwm_AXI
    generic map (
        C_S_AXI_DATA_WIDTH  => C_pwm_AXI_DATA_WIDTH,
        C_S_AXI_ADDR_WIDTH  => C_pwm_AXI_ADDR_WIDTH
    )
    port map (
        pwm_1_out           =>(pwm_1_out),
        pwm_2_out           =>(pwm_2_out),
        pwm_3_out           =>(pwm_3_out),
```

- After these steps, we can synthesize the custom IP. Click run synthesis. Then wait.

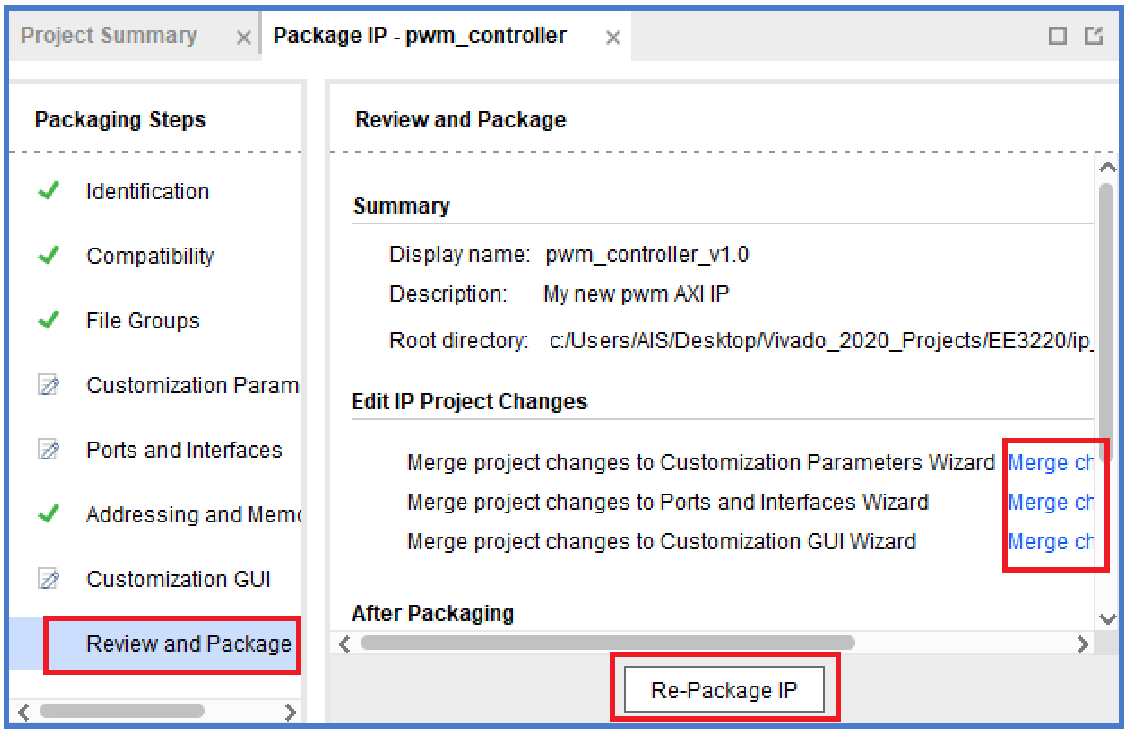


- Click **Package IP** tab
- In compatibility page, life cycle tab, select production.



Family	Life Cycle
zynq	Production

- Select **review and package**. Click all the **Merge project changes** highlighted in blue.
- Click **Re-Package IP**, then click OK and close the project when finished.



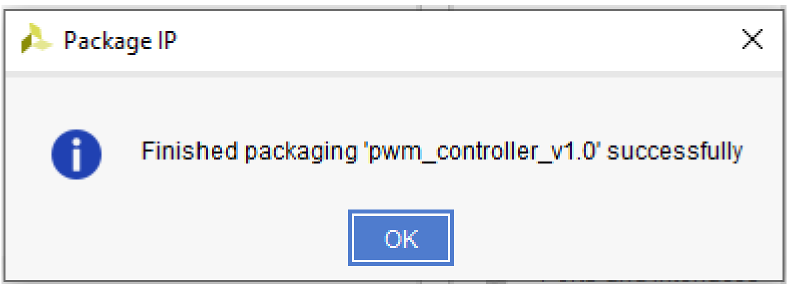
Summary

Display name: pwm_controller_v1.0
Description: My new pwm AXI IP
Root directory: c:/Users/AIS/Desktop/Vivado_2020_Projects/EE3220/ip_

Edit IP Project Changes

Merge project changes to Customization Parameters Wizard Merge ch
Merge project changes to Ports and Interfaces Wizard Merge ch
Merge project changes to Customization GUI Wizard Merge ch

After Packaging


Package IP

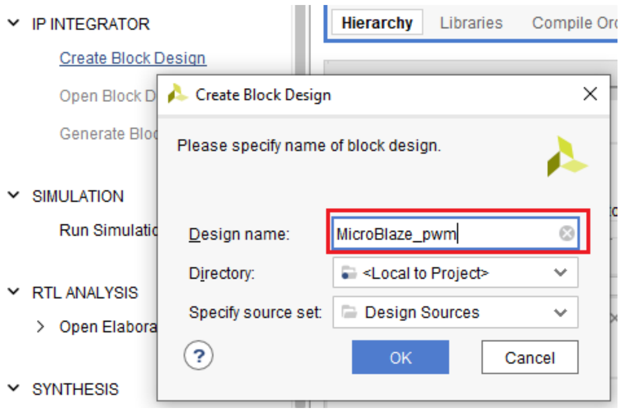
Finished packaging 'pwm_controller_v1.0' successfully

Part2: Build the embedded system with MicroBlaze and the custom PWM IP

In this part, the pwm custom IP created in part1 will be integrated with a MicroBlaze soft processor core. We will use the block design approach in the Vivado hlx and add the IPs as we did in laboratory 3. The whole system will be synthesized, implemented and bitstream will finally be created. The design will be exported to Vitis for the software development and programming of the FPGA.

1. Create a new block design and add the MicroBlaze and the pwm IPs

- In the original project, click create block design. Name the design and click **OK**

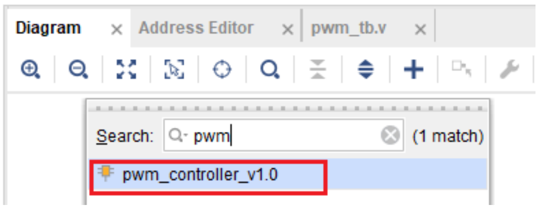

Please specify name of block design.

Design name: MicroBlaze_pwm

Directory: <Local to Project>

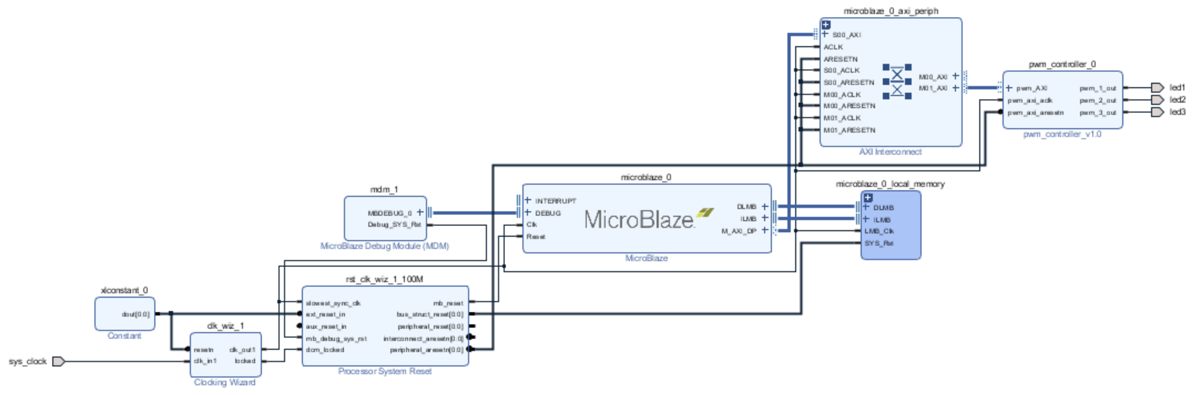
Specify source set: Design Sources

- Click add IP **'+'** sign in the block design and add the MicroBlaze IP as you did in lab 3
- Search **pwm** and select our new pwm_controller IP to add the our pwm IP

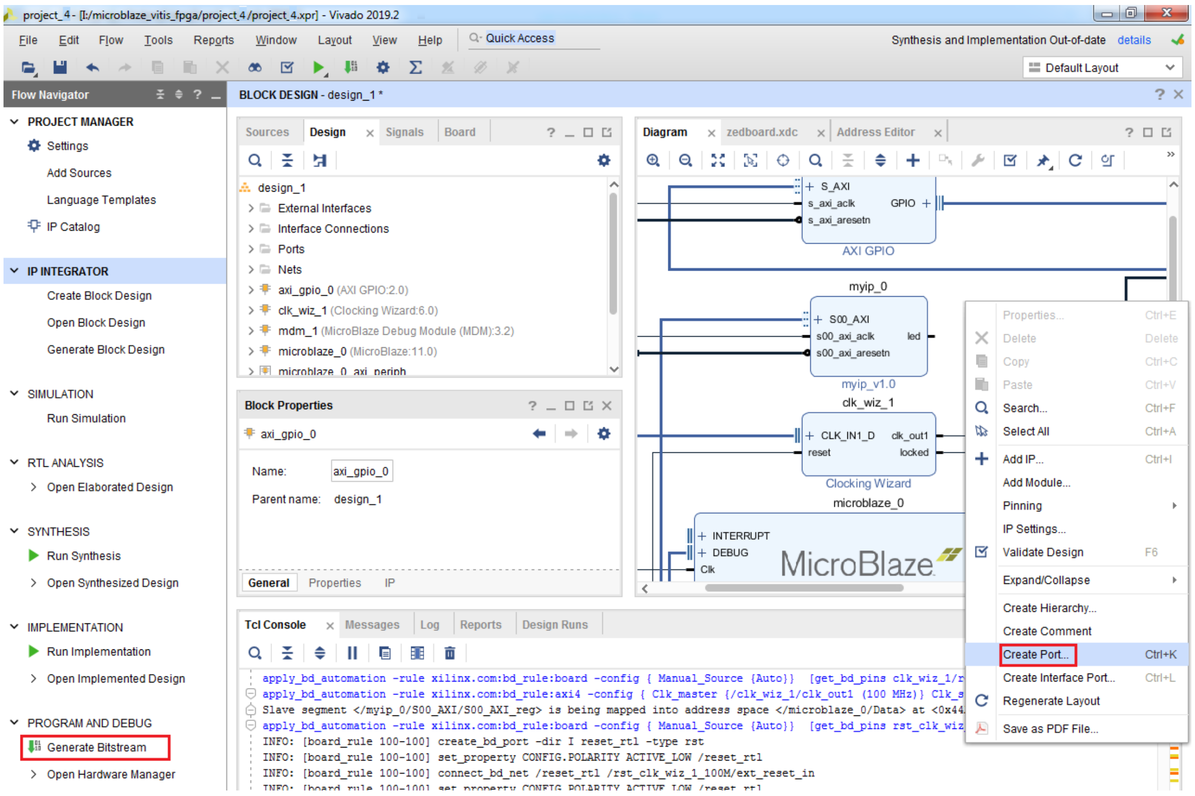

Search: pwm (1 match)

pwm_controller_v1.0

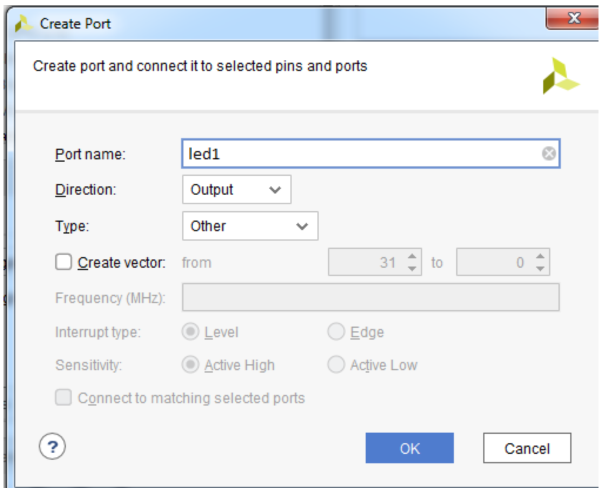
- Click **run block automation** and the **Regenerate layout**.
- Configure the clock wizard to use **sys_clk** and have **active low reset**. Also add a constant 1 to the **resetn** of the clock and the **ext_reset_in** of the Processor Systems Reset as we did in lab 3.
- Click **run connection automation** and Regenerate layout.



- We need to add an output ports (led1, led2 and led3) to the pwm output. Right click on the design and choose **Create Port**. Connect the ports to the outputs of **PWM_Controller IP**



```
apply_bd_automation -rule xilinx.com:bd_rule:board -config { Manual_Source {Auto}}  [get_bd_pins clk_wiz_1/r
apply_bd_automation -rule xilinx.com:bd_rule:axi4 -config { Clk_master {/clk_wiz_1/clk_out1 (100 MHz)} Clk_s
Slave segment </myip_0/S00_AXI/S00_AXI_reg> is being mapped into address space </microblaze_0/Data> at <0x44
apply_bd_automation -rule xilinx.com:bd_rule:board -config { Manual_Source {Auto}}  [get_bd_pins rst_clk_wiz
INFO: [board_rule 100-100] create_bd_port -dir I reset_rtl -type rst
INFO: [board_rule 100-100] set_property CONFIG.POLARITY ACTIVE_LOW /reset_rtl
INFO: [board_rule 100-100] connect_bd_net /reset_rtl /rst_clk_wiz_1_100M/ext_reset_in
INFO: [board_rule 100-100] set_property CONFIG.POLARITY ACTIVE_LOW /reset_rtl
```



Create port and connect it to selected pins and ports

Port name: led1

Direction: Output

Type: Other

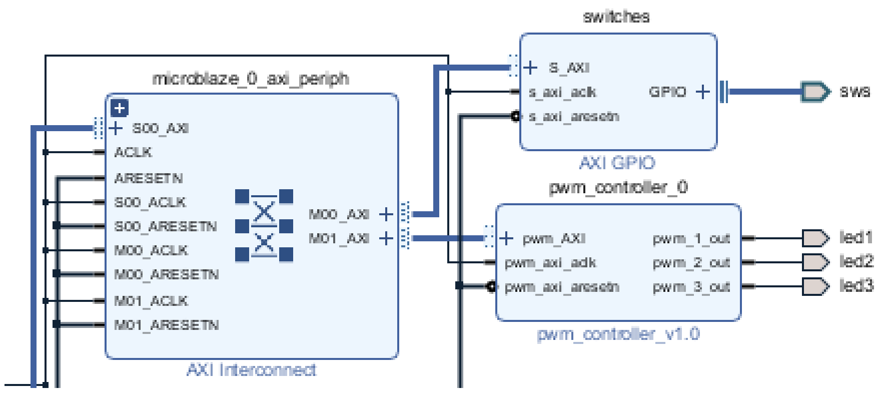
☐ Create vector: from 31 to 0

Frequency (MHz):

Interrupt type: ● Level ○ Edge

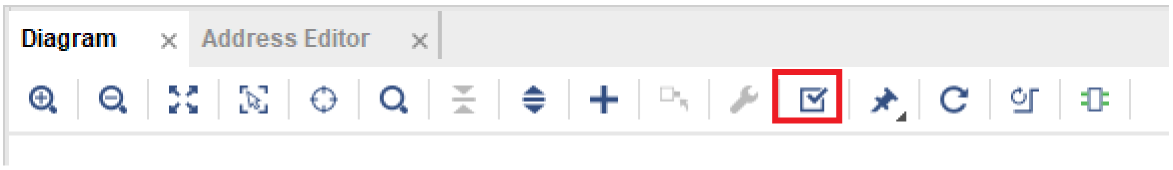
Sensitivity: ● Active High ○ Active Low

☐ Connect to matching selected ports



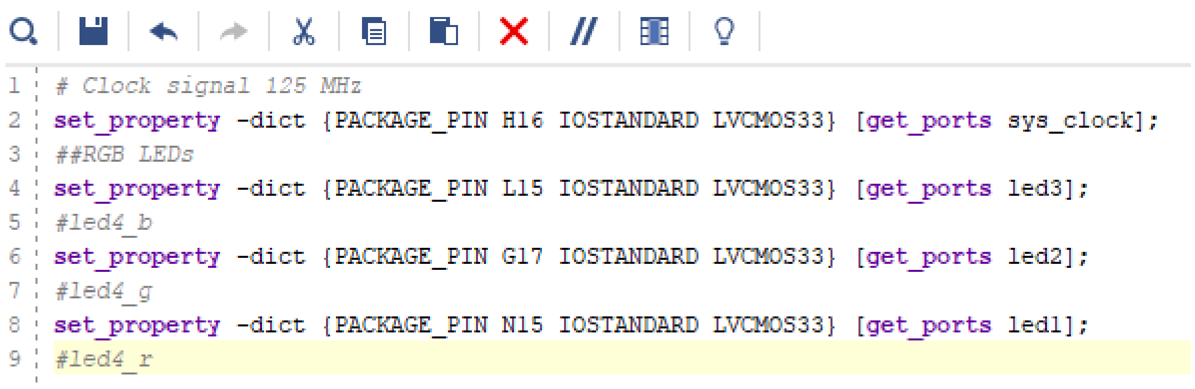
2. Validate the design and create bitstream.

- Click on validate design (a tick icon). After a dialog box shows validation successful. Click **OK**



- On the sources panel, right the block design, choose **Create HDL wrapper**, click OK.
- Choose add sources **(+)** sign under source to create a constraint file, click **Create File** tab, specify constraint file name and click OK
- Double click and open the constraint file you just created , i.e. pynq_z2_constr.xdc
- Enter the constraints we provide you below by copying or typing.
- In the Flow navigator, click **generate bitstream**

```
# Clock signal 125 MHz
set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports sys_clock];
##RGB LEDs
set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports led3]; #
led4_b
set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports led2]; #
led4_g
set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports led1]; #
led4_r
```



```
# Clock signal 125 MHz
set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports sys_clock];
##RGB LEDs
set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports led3];
#led4_b
set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports led2];
#led4_g
set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports led1];
#led4_r
```

3. Export hardware.
   After bitstream file is successfully generated, then export the hardware.

- Go to File, under Export, click **Export Hardware**
- Select **Fixed** option and click next
- Choose **include bitstream** and click next
- Choose the lab4 project directory where the XSA file will be stored.
- Click Next and then Finish

Check Point 2:

Take snapshots to be attached in the lab report as the evidence of achieving this checkpoint.

Part 3: Software Development using Vitis IDE

1. Create new Vitis application project

- Open Vitis IDE, choose a folder as your workspace location and click launch.
- Choose **Create application project** and click Next
- Choose **create a new platform from hardware(xsa)**, browse and go to the **lab4** project folder and select the **MicroBlaze_pwm_wrapper.xsa**. Click **Next** and leave other setting as default, then, **Finish**.
- Specify the project name as **lab4_vitis**, click next.
- Click **Next**
- Choose **Empty Application** and click Finish. Our new application project will be created

2. Create new source code file (**pwm.c**).

- Right click src, select **New ->File** to create a new source code file.
- Name the source file as **pwm.c**, click finish. A new source file will be created

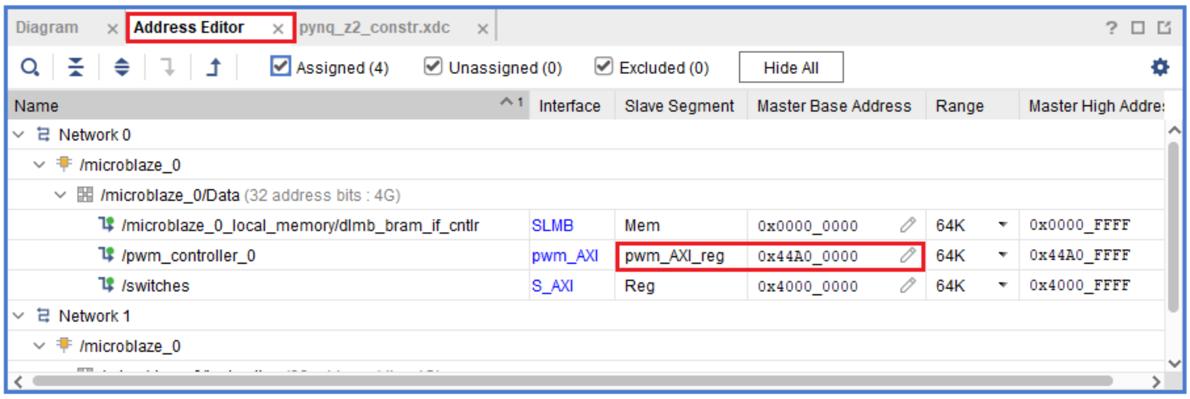
3. Write a program to display different colors on the RGB LEDS using pwm, build project and program the FPGA.

- Open the **pwm.c** file, copy and paste the source code provided. You can Use **Paint**, the Microsoft software to identify the rgb values of different colors. Here we give the value for some colors.

Color	RGB values
Purple	Red =128; Green = 0; Blue=128
Yellow	Red =255; Green = 255; Blue = 0
Orange	Red =255; Green = 128; Blue = 0

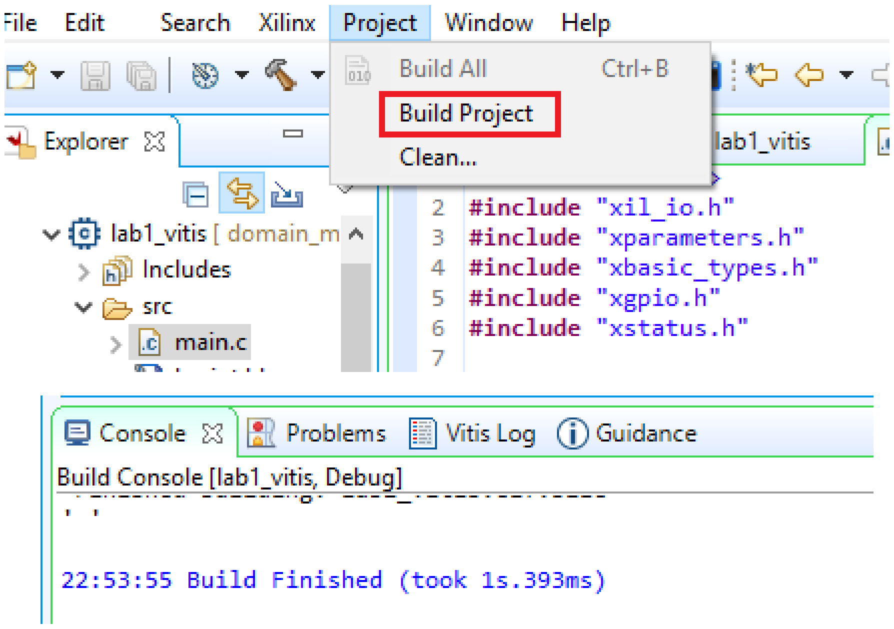
```
#include <stdio.h>
#include "xil_io.h"
#include "xparameters.h"
#include "xbasic_types.h"
#include "xstatus.h"

#define slv_reg0 0x44A00000  //BASEADDR of pwm_controller registers. This value
is //obtained from xparameters.h or the Address Editor in Vivado

int main()
{
    Xuint32      *pwm_1_dc_count; // declare pointers;
    Xuint32      *pwm_2_dc_count;
    Xuint32      *pwm_3_dc_count;
    Xuint32      *enable_reg;

// initialize pointers to the address of the axi peripheral slave registers

    enable_reg = (u32 *)(slv_reg0);
    pwm_1_dc_count = (u32 *) (slv_reg0 + 4);
    pwm_2_dc_count = (u32 *)(slv_reg0 + 8);
    pwm_3_dc_count = (u32 *)(slv_reg0 + 12);

// enable the PWM outputs and set the duty cycle counter values for the
registers
    *(enable_reg) = 0x07; // enable all the 3 pwm outputs
    *(pwm_1_dc_count) = (u32) 0;     //red color value
    *(pwm_2_dc_count) = (u32) 255;   //green color value
    *(pwm_3_dc_count) = (u32) 128;   //blue color value

    while(1);

    return 0;
 }
```

For the actual value of the pwm_controller **BASEADDR** refer to the Address Editor of the block design in Vivado. The value is **0x44A0_0000** for our design.



Name	Interface	Slave Segment	Master Base Address	Range	Master High Addre
Network 0					
/microblaze_0					
/microblaze_0/Data (32 address bits : 4G)					
/microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
/pwm_controller_0	pwm_AXI	pwm_AXI_reg	0x44A0_0000	64K	0x44A0_FFFF
/switches	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
Network 1					
/microblaze_0					

- Open the project menu, choose **Build Project**

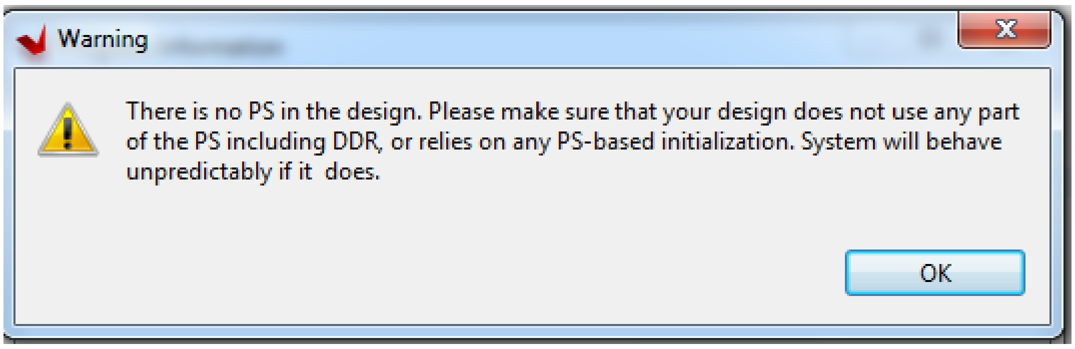


```
#include "xil_io.h"
#include "xparameters.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xstatus.h"
```

Build Console [lab1_vitis, Debug]

```
22:53:55 Build Finished (took 1s.393ms)
```

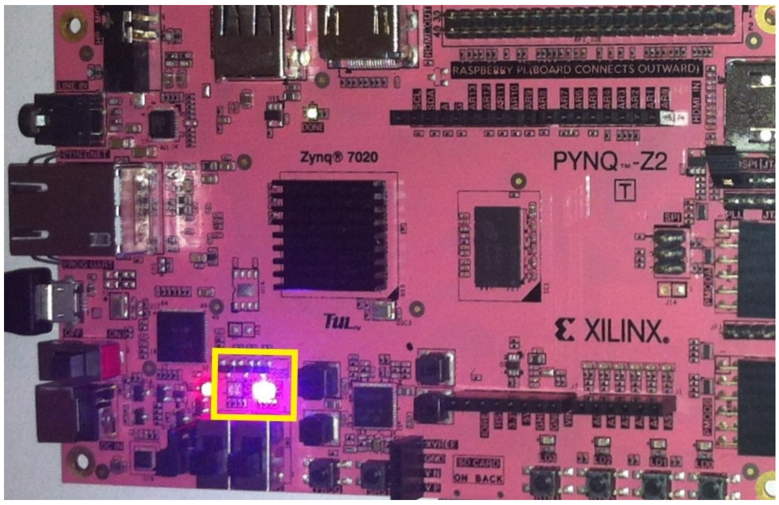
- Open the Xilinx menu, choose **Program FPGA**. Remember to connect the board to your computer and turn it on before you program it.
- Browse the bitstream, mmi and the .elf for the project as done in lab3.
- Click program.
- Wait and click OK



Warning

There is no PS in the design. Please make sure that your design does not use any part of the PS including DDR, or relies on any PS-based initialization. System will behave unpredictably if it does.

OK



Check Point 3:

Modify the program and now display pink, and yellow colors. Take snapshots and attached in the lab report as the evidence of achieving this checkpoint.

~ END ~