

# Introduction to Zynq Hardware

## Lab 7

### Adding Custom IP to Vivado IP Catalog



June 2017  
Version 12

## Lab 7 Overview

Xilinx offers a diverse IP catalog which makes it very easy to connect many of the common interfaces to your design. This lab will provide step-by-step instructions on how to create custom IP, add it to the IP catalog, and then connect it into our design.

## Lab 7 Objectives

When you have completed Lab 7, you will know how to do the following:

- Create a new IP project
- Customize the IP
- Add it to the Vivado IP Catalog
- Add this custom IP to your project
- Add an interrupt to the Zynq PS and connect to the custom IP
- Test the custom IP with a custom software application

## Experiment 1: Create A New IP Project

This experiment shows how to create a new IP project, as well as define the AXI interface and register settings for the IP.

### Experiment 1 General Instruction:

Create a new AXI peripheral.

### Experiment 1 Step-by-Step Instructions:

1. <Optional> If you did not complete Lab 6 or wish to start with a clean copy, delete the ZynqDesign, IP and SDK\_Workspace folders in the ZynqHW/2017\_1 folder. Then unzip **Solutions\ZynqHW\_Lab6\_Solution.zip** to the 2017\_1 folder. If you have 7-zip installed, you can do this by right-clicking and dragging **ZynqHW\_Lab6\_Solution.zip** to the 2017\_1 folder. Select **7-Zip → Extract Here**.
2. Open **Vivado** with the existing project.
3. Select **Tools → Create and Package IP New IP....**

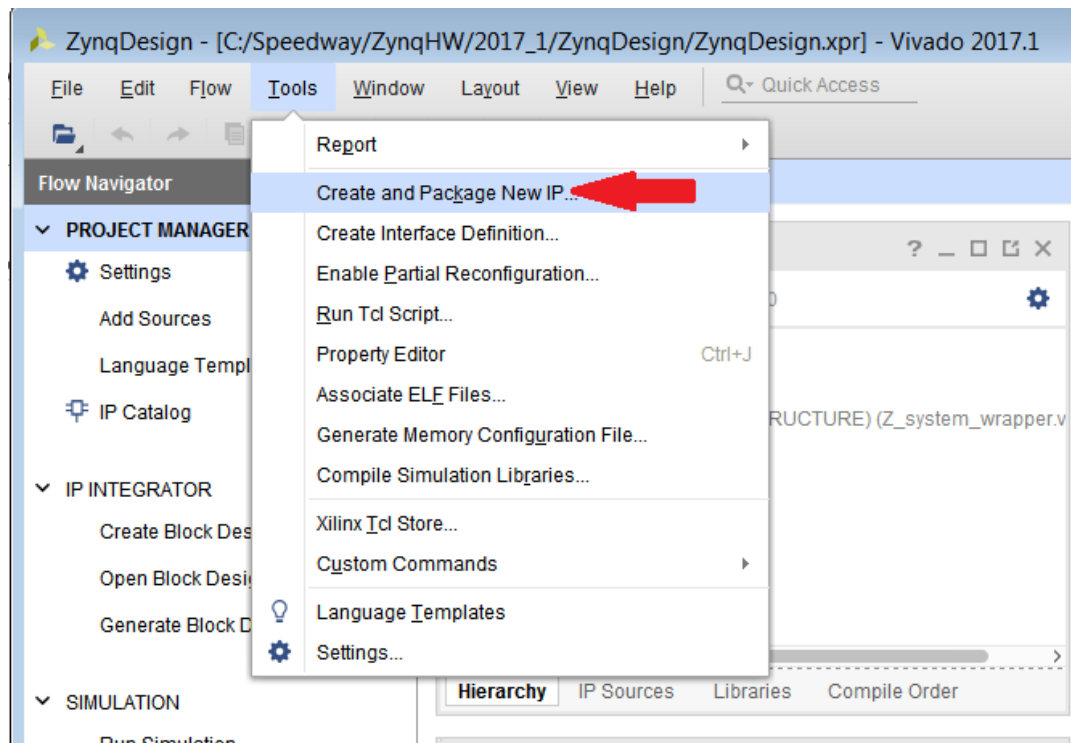


Figure 1 - Create New IP

4. Click **Next >** on Welcome Screen to Create and Package IP.

5. We will be **creating a new AXI peripheral**, check the corresponding box and hit **Next >**.

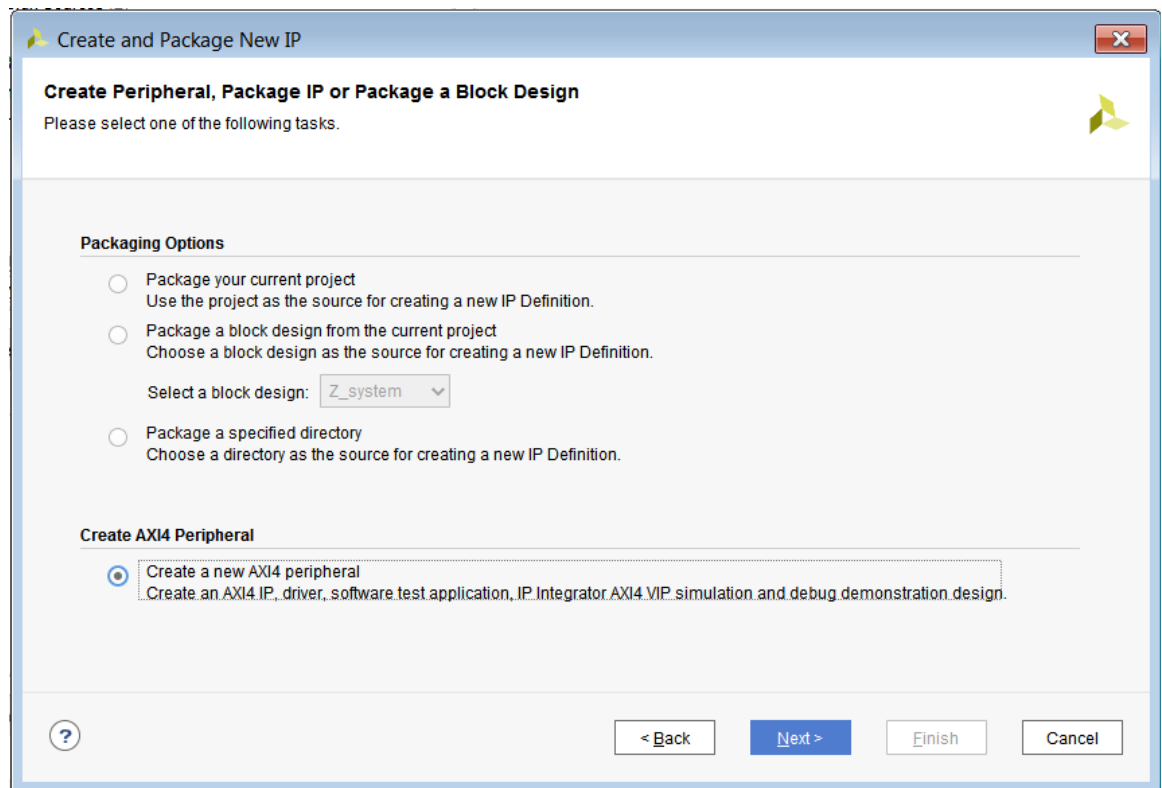


Figure 2 - Create a New AXI Peripheral

6. **\*\*\*Very Important\*\*\* Please use the same naming conventions outlined in step 7.**
7. Enter the following fields as shown here. The only critical fields are **Library**, as we will create this in our USER library, and **Name** and **Display Name**. When done, click **Next >**.

Name:	PWM_w_Int
Version:	1.0
Display name:	PWM_w_Int_v1.0
Description:	PWM with Interrupt option
IP location:	C:/Speedway/ZynqHW/2017_1/ip_repo

**Create and Package New IP**

**Peripheral Details**  
Specify name, version and description for the new peripheral

Name: PWM\_w\_Int

Version: 1.0

Display name: PWM\_w\_Int\_v1.0

Description: PWM with Interrupt option

IP location: C:/Speedway/ZynqHW/2017\_1/ip\_repo

☐ Overwrite existing

< Back Next > Finish Cancel

**Figure 3 - IP Details**

8. This peripheral is simple and does not require much bandwidth, thus an AXI-Lite interface is acceptable. This will also be a slave device with the PS as the Master. A 32-bit interface is good and we only really need 1 register, but four is the lowest setting. Thus the default parameters are acceptable. Click **Next >**.

**Create and Package New IP**

**Add Interfaces**  
Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Interfaces: S00\_AXI

Name: S00\_AXI

Interface Type: Lite

Interface Mode: Slave

Data Width (Bits): 32

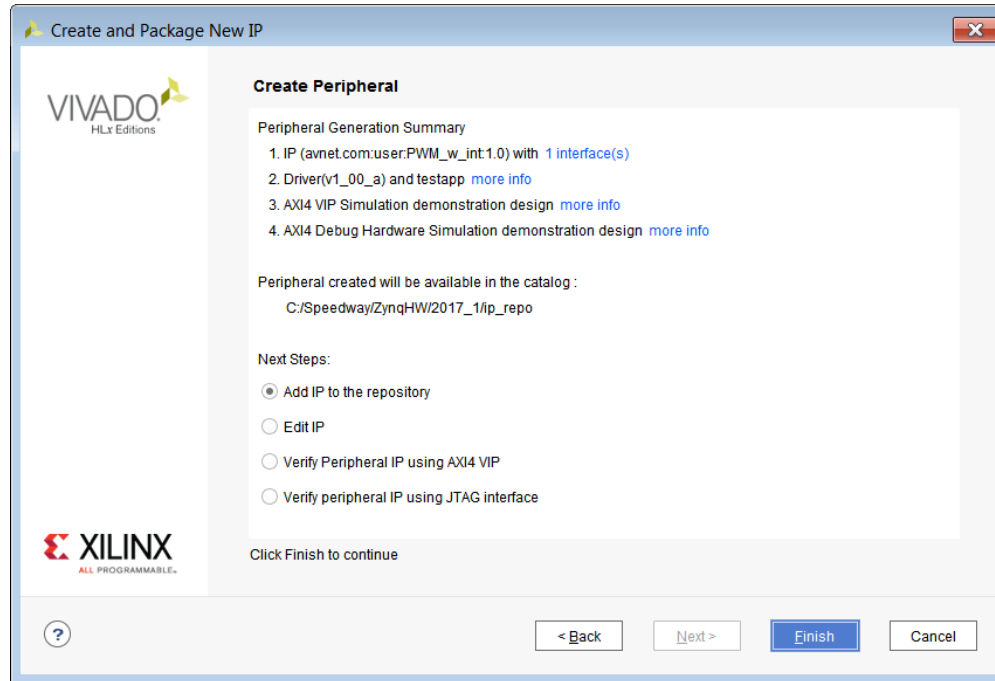
Memory Size (Bytes): 64

Number of Registers: 4 [4..512]

< Back Next > Finish Cancel

**Figure 4 - IP Interface Settings**

9. Select the **Add IP to the repository** radio button. Verify the *Catalog* path. If OK, click **Finish**.



**Figure 5 - IP Creation Summary**

**Questions:**

**Answer the following questions:**

- Where is the IP project located?

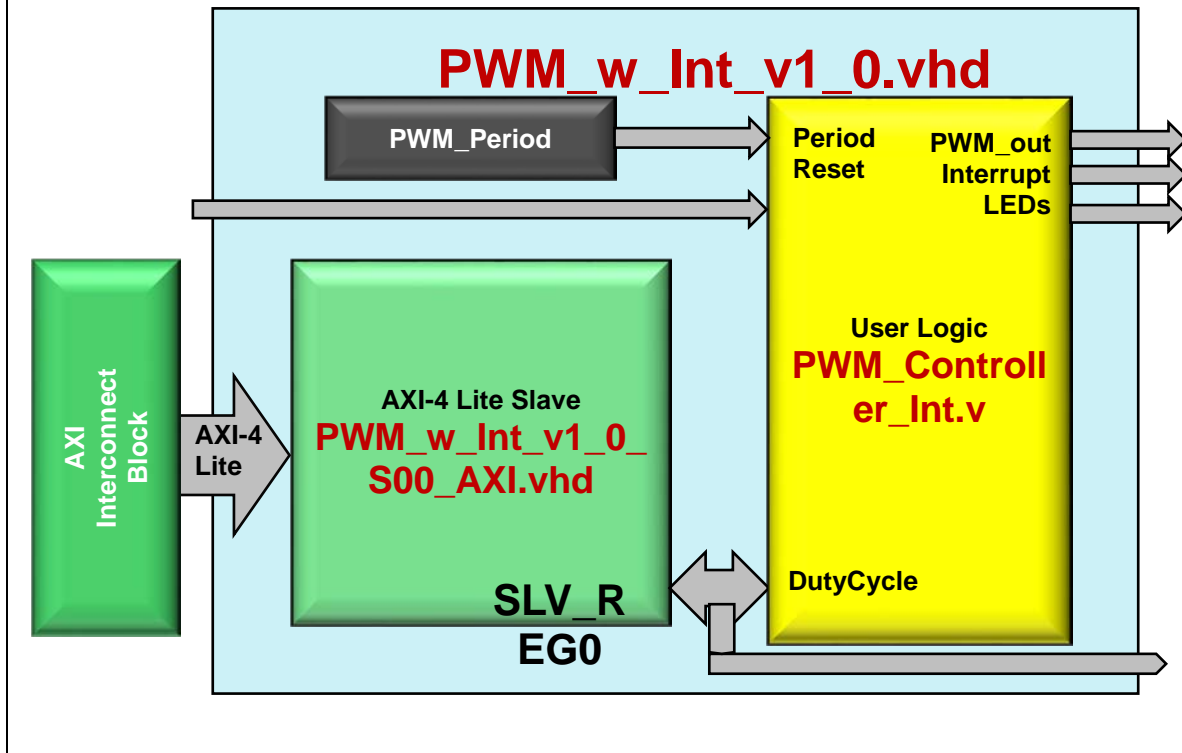
---

## Experiment 2: Customize the New IP Project

This experiment shows how to bring in custom user logic and instantiate it into the new IP project. Additionally, we'll customize the IP project adding a new port as well as parameter inputs.

### Experiment 2 General Instruction:

Import user logic HDL, PWM\_Controller\_Int.v, and instantiate it into the project. Connect to slv\_reg0 from the AXI interface. Add PWM outputs to top-level.



### Experiment 2 Step-by-Step Instructions:

1. In Vivado, select **Window** → **IP Catalog**.
2. Select the IP in the IP catalog by searching for "PWM" in the search window. Right click and select **Edit in IP Packager**.

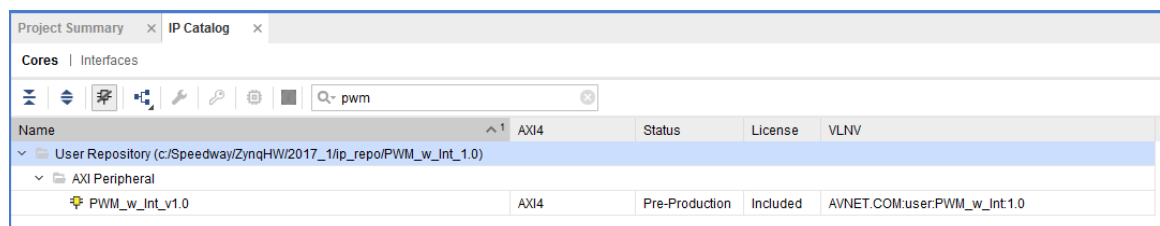
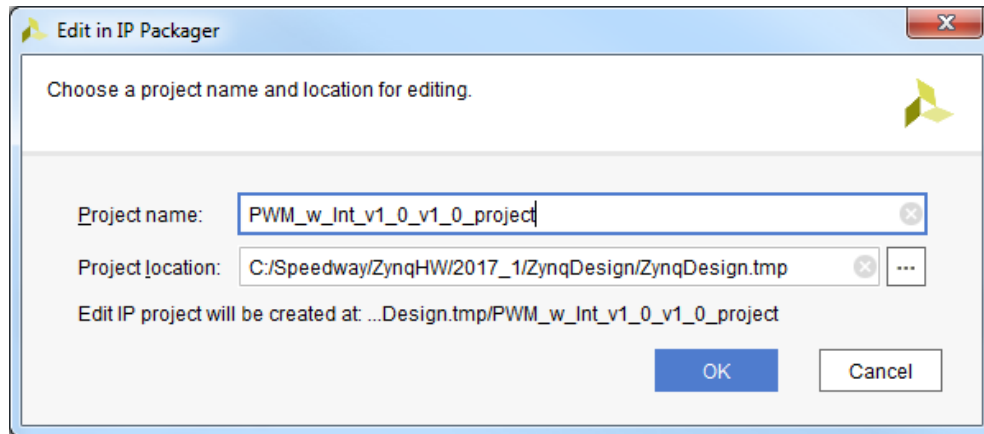


Figure 6 – Edit IP Project

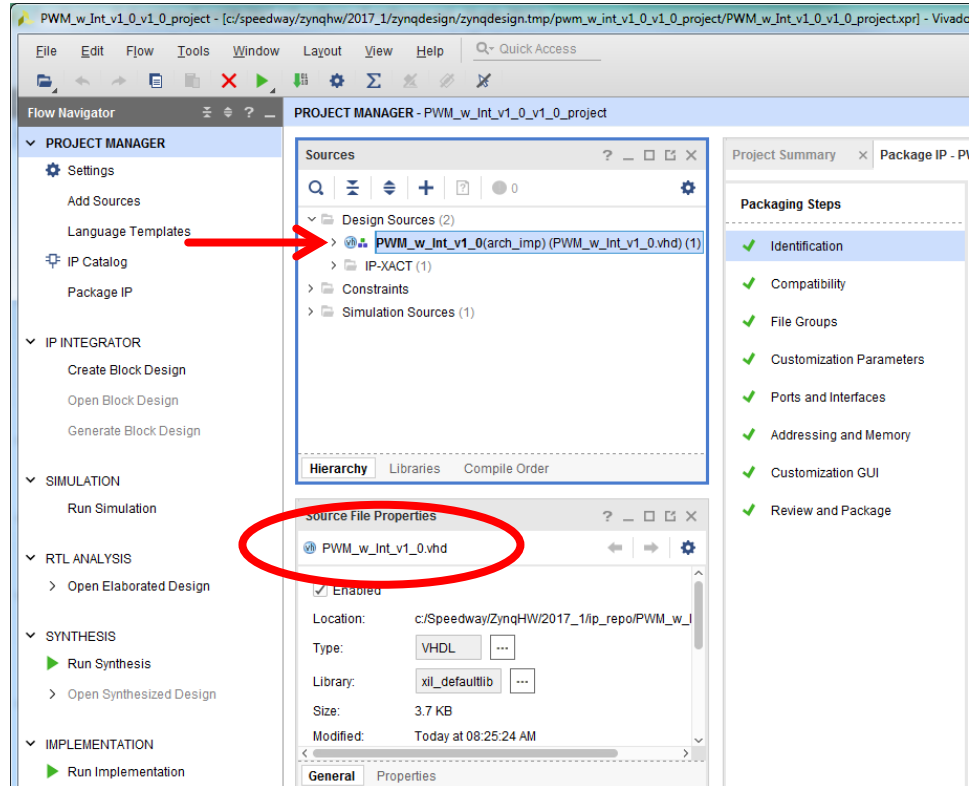
3. Accept the project defaults and click **OK**.



**Figure 7 – IP Project Name and Location**

This will open a new Vivado Project when completed. In this new project we will edit our IP. Take note of the Root Directory and Project Name. This is important for when you want to edit this IP at a later date. The **<IP Name>\_project.xpr** project is the project you will open to edit this IP. In this case, our IP project is named, **PWM\_w\_Int\_v1\_0\_project.xpr**.

4. Expand the Design Sources in the Sources window. Notice, when a file is selected in the sources window, its properties show up in the *Source File Properties* window. This is helpful in determining where your HDL files exist on your PC.



**Figure 8 - Design Sources and Properties**



5. There are two VHDL files. One, **PWM\_w\_Int\_v1\_0.vhd**, is the top-level wrapper file. **Open** this file by double-clicking on it. It instantiates the AXI interface, **PWM\_w\_Int\_v1\_0\_S00\_AXI** around line 84, which is defined by the 2<sup>nd</sup> HDL file **PWM\_w\_Int\_v1\_0\_S00\_AXI.vhd**. A place holder exists just below this to add user logic, which is where we will add our custom IP HDL.

```
84  -- Instantiation of Axi Bus Interface S00_AXI
85  PWM_w_Int_v1_0_S00_AXI_inst : PWM_w_Int_v1_0_S00_AXI
86      generic map (
87          C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
88          C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
89      )
90      port map (
91          S_AXI_ACLK  => s00_axi_aclk,
92          S_AXI_ARESETN  => s00_axi_aresetn,
93          S_AXI_AWADDR  => s00_axi_awaddr,
94          S_AXI_AWPROT  => s00_axi_awprot,
95          S_AXI_AWVALID  => s00_axi_awvalid,
96          S_AXI_AWREADY  => s00_axi_awready,
97          S_AXI_WDATA  => s00_axi_wdata,
98          S_AXI_WSTRB  => s00_axi_wstrb,
99          S_AXI_WVALID  => s00_axi_wvalid,
100         S_AXI_WREADY  => s00_axi_wready,
101         S_AXI_BRESP  => s00_axi_bresp,
102         S_AXI_BVALID  => s00_axi_bvalid,
103         S_AXI_BREADY  => s00_axi_bready,
104         S_AXI_ARADDR  => s00_axi_araddr,
105         S_AXI_ARPROT  => s00_axi_arprot,
106         S_AXI_ARVALID  => s00_axi_arvalid,
107         S_AXI_ARREADY  => s00_axi_arready,
108         S_AXI_RDATA  => s00_axi_rdata,
109         S_AXI_RRESP  => s00_axi_rresp,
110         S_AXI_RVALID  => s00_axi_rvalid,
111         S_AXI_RREADY  => s00_axi_rready
112     );
113
114     -- Add user logic here
115
116     -- User logic ends
```

**Figure 9 – User Logic Location**

6. Now we want to copy PWM\_Controller\_Int.v from the

**C:\Speedway\ZynqHW2017\_1\Support\_documents**

directory and paste it into the

**C:\Speedway\ZynqHW2017\_1\ip\_repo\PWM\_w\_Int\_1.0\hdl** directory

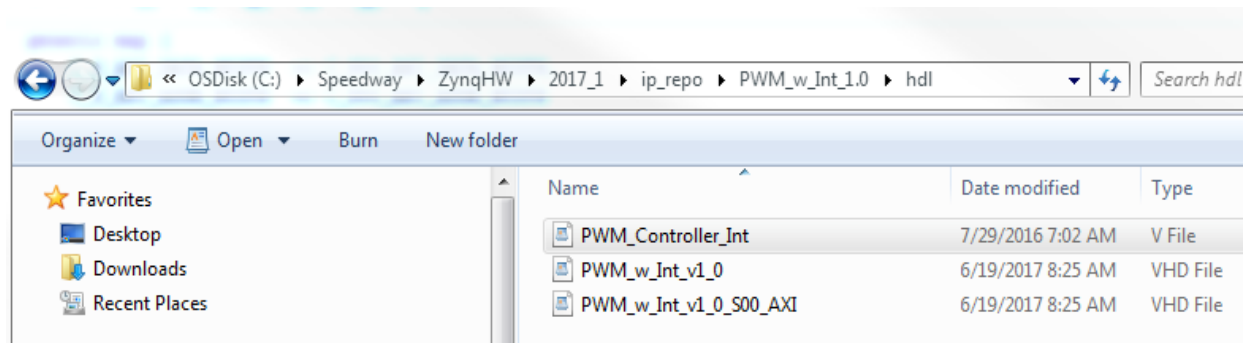


Figure 10 - Copy HDL Files into IP HDL Directory

7. Click **Add Sources** from the Flow Navigator window, and then select **Add or create design source**. When done, click **Next >**.

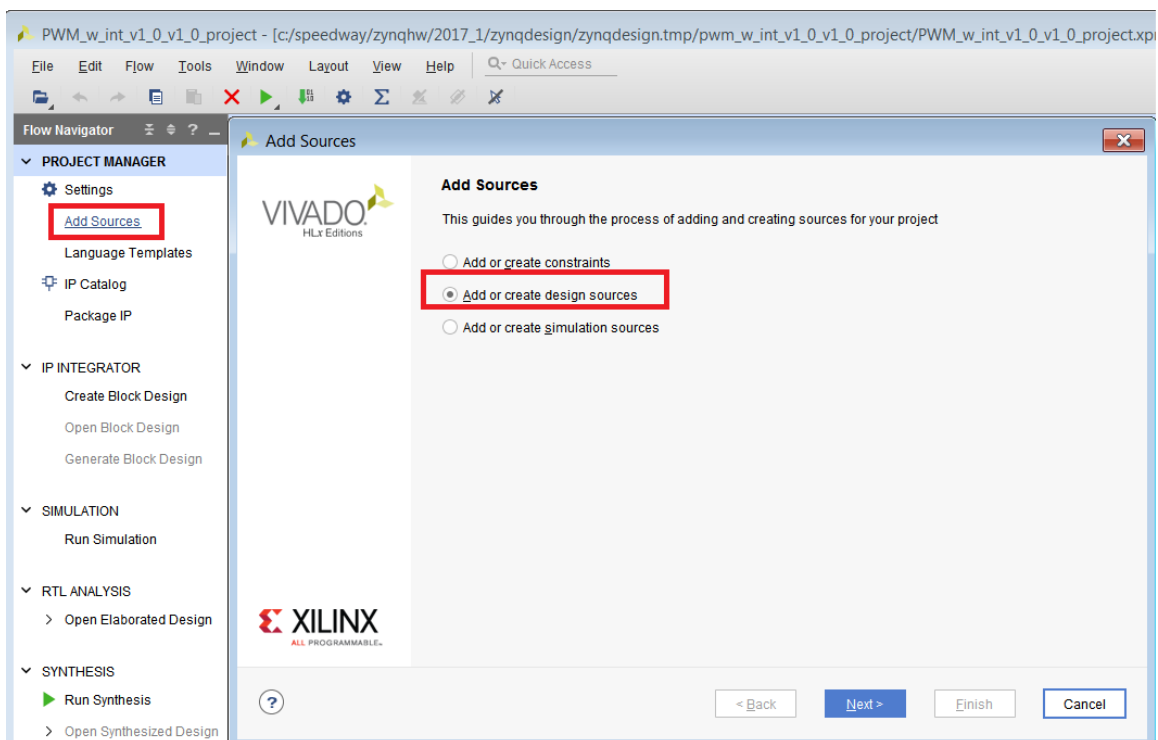


Figure 11 - Add Design Sources

8. In the Add Source Files window, Click **Add Files**

9. Change the directory to:

**C:\Speedway\ZynqHW\2017\_1\ip\_repo\PWM\_w\_Int\_1.0\hdl**

Double-click **PWM\_Controller\_Int.v** to add it to the project.

10. Verify *copy sources into project* is not checked. Click **Finish**.

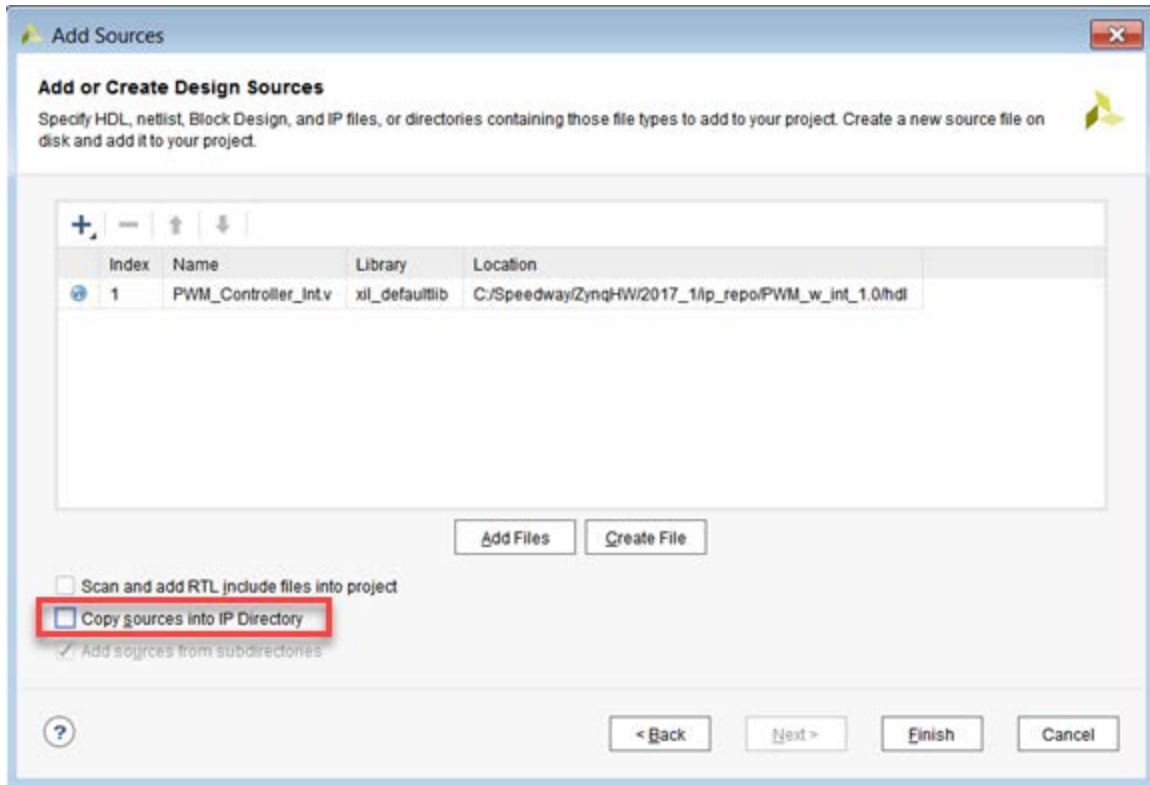


Figure 12 - Add Files to Project

**\*\*\*Note\*\*\*** In the directory specified below, there are the three files that require modification in the following steps leading up to Step 22. If you encounter an issue writing the HDL code any synthesizing the design, feel free to copy and paste the file text into your Vivado project.

C:\Speedway\ZynqHW\2017\_1\Support\_documents\PWM\_Controller\_Int\_Modified

11. The file will now show up in our Sources window. But not under our top-level file. This is expected as we have not instantiated this code in the top-level file yet.

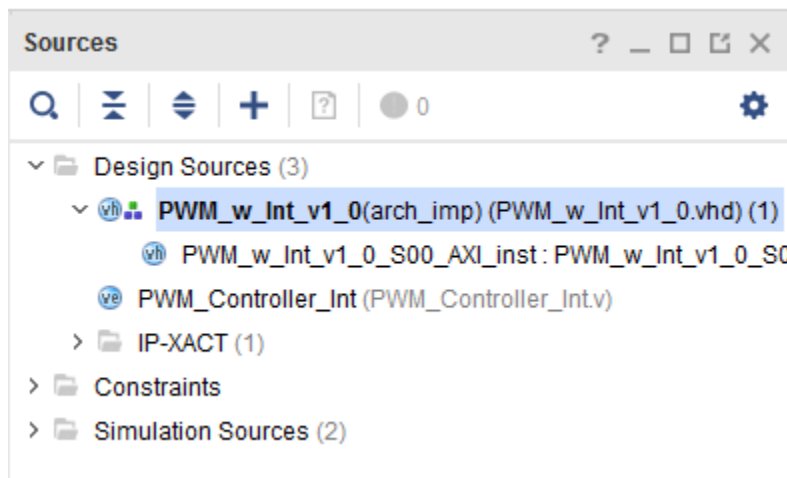


Figure 13 - New Source in Project

12. Open PWM\_Controller\_Int by double-clicking it. Look over the file and then instantiate it by double clicking on PWM\_w\_Int\_v1\_0.vhd. Then declare the component **PWM\_Controller\_Int** in the component declaration section of **PWM\_w\_Int\_v1\_0.vhd** around line 55.

```
54      -- component declaration
55      component PWM_Controller_Int is
56      generic (
57          period : integer := 20
58      );
59      port (
60          Clk : in std_logic;
61          DutyCycle : in std_logic_vector(31 downto 0);
62          Reset : in std_logic;
63          PWM_out : out std_logic_vector(7 downto 0);
64          Interrupt : out std_logic;
65          count : out std_logic_vector(period-1 downto 0)
66      );
67      end component PWM_Controller_Int;
```

Figure 14 – Add Component Declaration

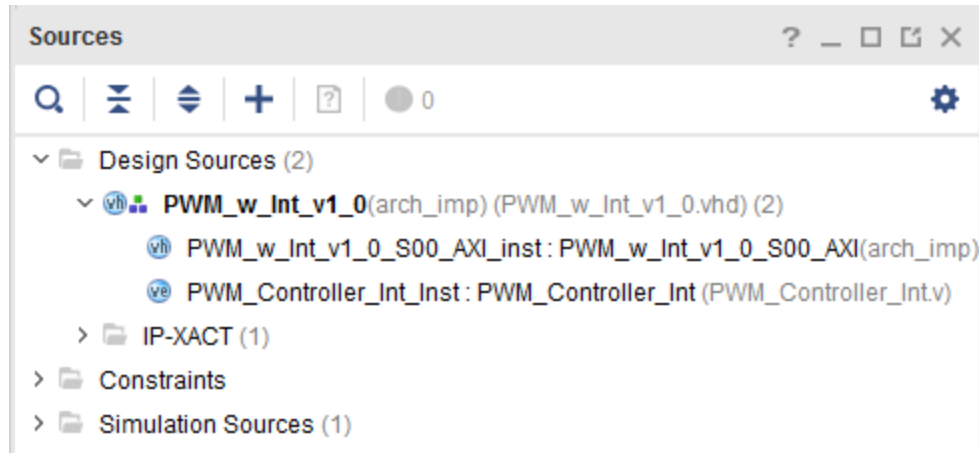
13. Scroll to bottom of **PWM\_w\_Int\_v1\_0.vhd** and enter the following in the user logic section (around line 139) :

```
129      -- Add user logic here
130      PWM_Controller_Int_Inst : PWM_Controller_Int
131      generic map (
132          period =>
133      )
134      port map (
135          Clk =>,
136          DutyCycle =>,
137          Reset =>,
138          PWM_out =>,
139          Interrupt =>,
140          count =>
141      );
142      -- User logic ends
```

Figure 15 – Add User Logic

Click **Save**, or Control-S.

14. Once entered, this HDL file will now be incorporated into the top level.



**Figure 16 - User Logic now in Project Hierarchy**

The next steps guide you through the methodology in connecting the User Logic HDL into the IP Core. If you are comfortable with this process, you can copy the final HDL files from *ZynqHW\_Lab7\_Solution.zip* in the *Solutions* folder into your IP/PWM\_w\_Int1.0/hdl folder. If you choose this, skip to step 21, else continue.

15. Now we need to connect the signals of our user logic, PWM\_Controller\_Int. Start by connecting the obvious signals in the PWM\_Controller\_Int instantiation. The clock and reset have the following associations between the Controller signal and the Slave AXI.

s00\_axi\_aclk to Clk  
s00\_axi\_aresetn to Reset

Edit **PWM\_w\_Int\_v1\_0.vhd** so the code looks like this (around line 135):

```

129      -- Add user logic here
130  PWM_Controller_Int_Inst : PWM_Controller_Int
131      generic map (
132          period =>
133      )
134      port map (
135          Clk => s00_axi_aclk,
136          DutyCycle =>,
137          Reset => s00_axi_aresetn,
138          PWM_out =>,
139          Interrupt =>,
140          count =>
141      );
142      -- User logic ends

```

**Figure 17 – Add in Clock and Reset**

Next we'll connect the outputs. These signals need to be defined in the top-level port list. This IP will bring out four outputs;

- **LEDs:** The actual PWM output to be connected to the LEDs
- **Interrupt\_out:** an Interrupt that indicates an invalid PWM setting
- **PWM\_Counter:** (for debug) the free running PWM counter
- **DutyCycle:** (for debug) the value passed down from the PS via the AXI interface to control the duty cycle of the PWM.
- Also add a configurable parameter, **PWM\_PERIOD**, that sets the counter depth of the PWM counter.

16. Add these ports at the top of the code as well as the instantiation of our user logic HDL(around line 8):

```

5  entity PWM_w_Int_v1_0 is
6      generic (
7          -- Users to add parameters here
8          PWM_PERIOD : integer := 20;
9          -- User parameters ends
10         -- Do not modify the parameters beyond this line
11
12
13         -- Parameters of Axi Slave Bus Interface S00_AXI
14         C_S00_AXI_DATA_WIDTH  : integer := 32;
15         C_S00_AXI_ADDR_WIDTH  : integer := 4
16     );
17     port (
18         -- Users to add ports here
19         LEDs : out std_logic_vector(7 downto 0);
20         Interrupt_Out : out std_logic;
21         PWM_Counter: out std_logic_vector(PWM_PERIOD-1 downto 0);
22         DutyCycle : out std_logic_vector(31 downto 0);
23         -- User ports ends

```

Figure 18 – Add Outputs

(Add Signal around line 99)

```

99  signal DutyCycle_int : std_logic_vector(31 downto 0);
100
101  begin
102
103  DutyCycle <= DutyCycle_int;

```

Figure 19 – Add Signals

(Complete user logic around line 138)

```

135      -- Add user logic here
136  PWM_Controller_Int_Inst : PWM_Controller_Int
137      generic map (
138          period => PWM_PERIOD
139      )
140      port map (
141          Clk => s00_axi_aclk,
142          DutyCycle => DutyCycle_int,
143          Reset => s00_axi_aresetn,
144          PWM_out => LEDs,
145          Interrupt => Interrupt_Out,
146          count => PWM_Counter
147      );
148      -- User logic ends

```

Figure 20 - Add User Logic

That fully connects our IP; however we have not connected to the source of DutyCycle. Again, this comes from the processor. Conveniently, when we went through the Create IP Peripheral wizard, Vivado created an AXI proxy for us.

17. Open **PWM\_w\_Int\_v1\_0\_S00\_AXI.vhd**. Look over the file, scroll down to line 100 and 202. Here you will see a commented explanation and the definitions for the four registers created by Vivado through this AXI proxy. We only need one of these registers, slv\_reg0.
18. Make slv\_reg0 a module output. Around line 19, add an output (**slave\_reg0** for example) under User Ports. Then connect it to slv\_reg0 (around line 123) :

```

17  port (
18      -- Users to add ports here
19      slave_reg0: out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
20      -- User ports ends
21      -- Do not modify the ports beyond this line
22  )

```

Figure 21 - Add Slave Module Output

```

120  begin
121      -- I/O Connections assignments
122      slave_reg0 <= slv_reg0;
123      S_AXI_AWREADY <= axi_awready;

```

Figure 22 – Connect Slave Register

19. **Save** the file, CTRL-S, and return to the top-level file, **PWM\_w\_Int\_v1\_0.vhd**.

Since we've added a new port from slave AXI interface, we need to connect it in the top-level file. Again, this register contains the **DutyCycle** setting passed down from the processor.

20. Add the slave\_reg0 port to the slave AXI interface component declaration (around line 77).

```

70     component PWM_w_Int_v1_0_S00_AXI is
71     generic (
72         C_S_AXI_DATA_WIDTH  : integer    := 32;
73         C_S_AXI_ADDR_WIDTH  : integer    := 4
74     );
75
76     port (
77         slave_reg0: out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
78         S_AXI_ACLK : in std_logic;

```

Figure 23 - Add Files to Project

Connect it to **DutyCycle\_int** signal, Around line 114. **Save** when done.

```

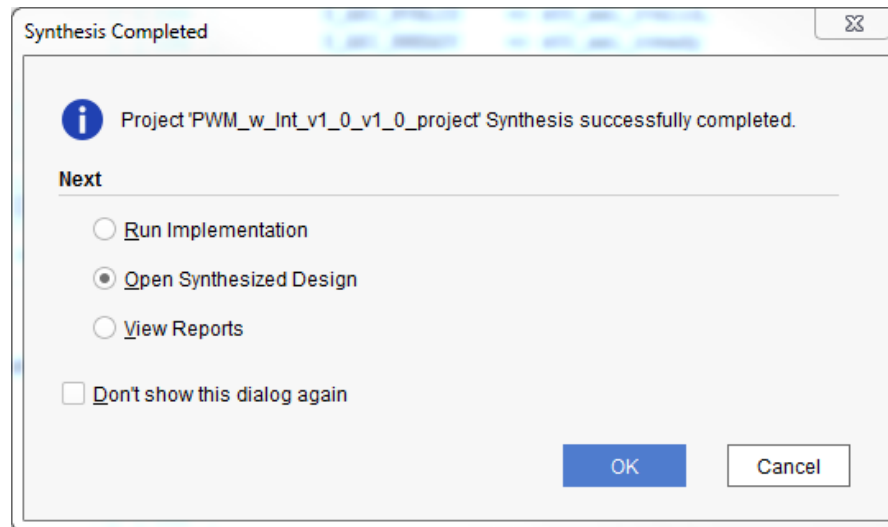
108 PWM_w_Int_v1_0_S00_AXI_inst : PWM_w_Int_v1_0_S00_AXI
109     generic map (
110         C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
111         C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
112     )
113     port map (
114         slave_reg0 => DutyCycle_int,
115
116         S_AXI_ACLK  => s00_axi_aclk,

```

Figure 24 - Add Files to Project

21. That completes all connections. Select the top-level HDL file, PWM\_w\_Int\_v1\_0.vhd, then click **Run Synthesis** to validate the design. Click yes to save is prompted then OK on the launch runs window.
22. Do not run implementation. Open the synthesized design from the *Synthesis Completed* dialog box.





**Figure 25 – Synthesis Completed**

23. Once the synthesized design opens, open the schematic view by clicking on **Schematic** under *Synthesis* in the *Flow Navigator*. Check that all the expected I/O are present. If it looks correct the design should be good. In a real world design further verification would be in order, but for now we can package the IP.

## Experiment 3: Package the IP Project

This experiment shows how to package the IP for the IP Catalog.

### Experiment 3 General Instruction:

Package the IP.

### Experiment 3 Step-by-Step Instructions:

1. Click **Package IP** in the Flow Navigator Window.

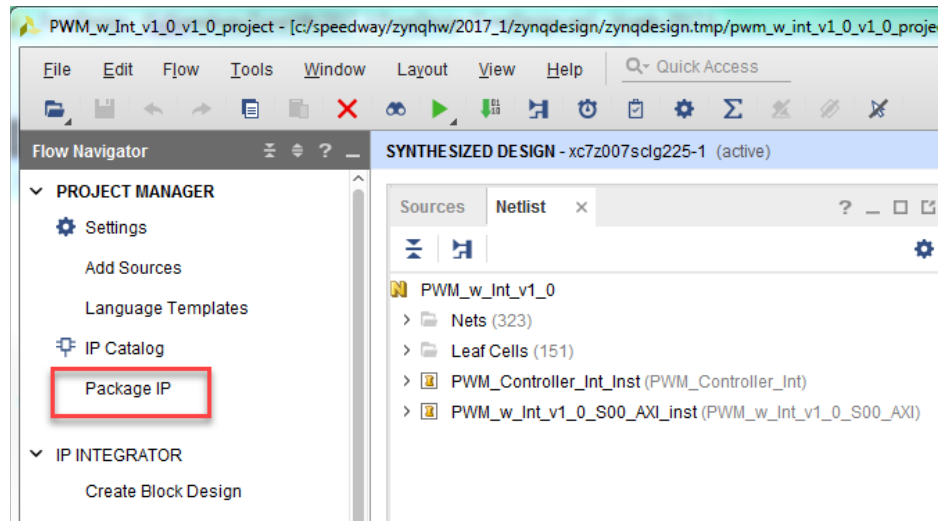


Figure 26 - Package IP

2. This GUI will build our IP step-by-step for the IP Catalog. In the first window, *IP Identification*, it has many of the parameters we entered when we created this project. **Verify** these parameters. Note the version of the IP is in this window. If updates or edits are made to this IP, the version number can be updated here.

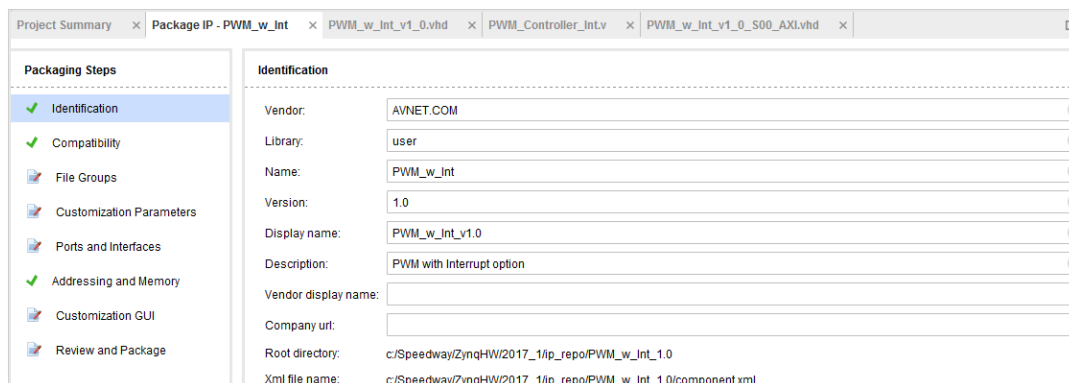


Figure 27 - IP Identification

- Switch to the next setting, *Compatibility*. Here you can set the compatible device families. Zynq is included as that was the target part when we created this IP. More families can be added by right-clicking in the box. Change the *Life Cycle* to **Production** by typing in the field shown in figure 28, then press **Enter**.

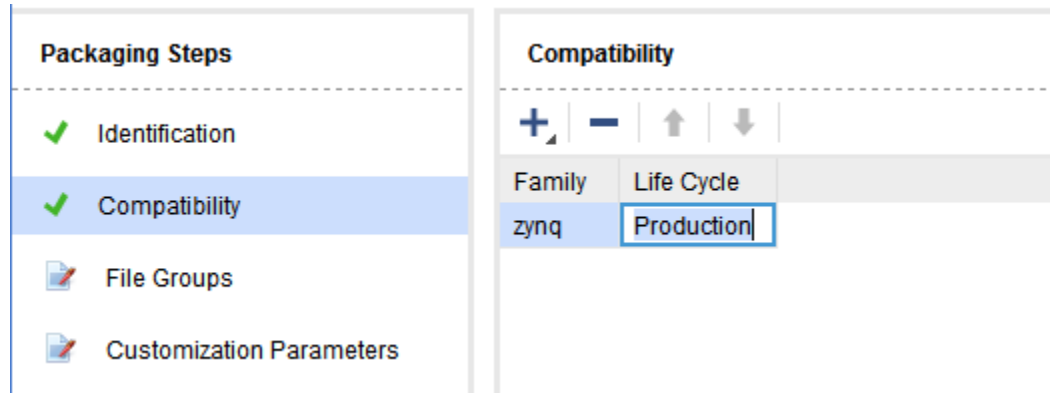


Figure 28 - IP Compatibility

- Next is *File Groups*, notice at the top of this window a notification is highlighted in blue. This indicates that Vivado has detected our changes we made in the HDL. Click **Merge changes from File Groups Wizard**.

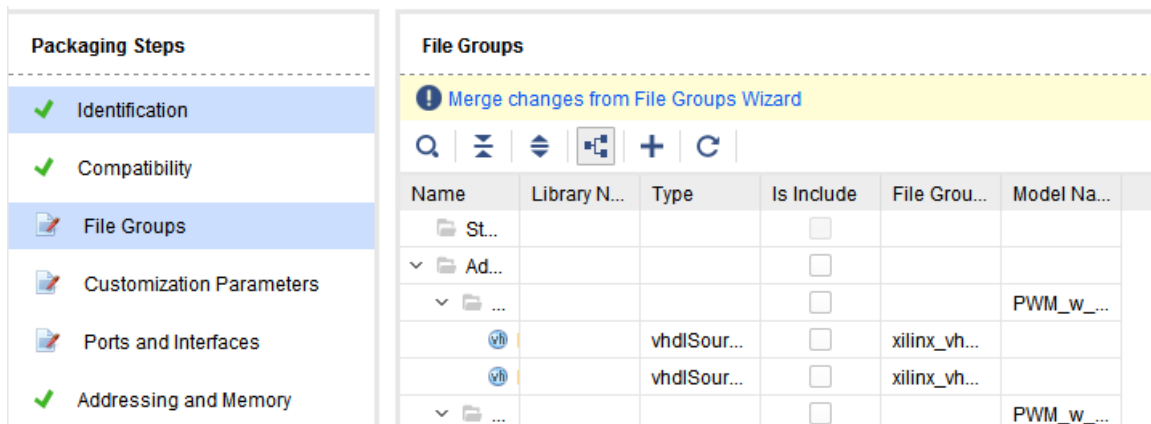


Figure 29 - IP File Groups

- Once the changes have merged, expand all files. Notice it's added our **PWM\_Controller\_Int** user logic HDL. However this Verilog module is within VHDL groups, hence the critical warnings.

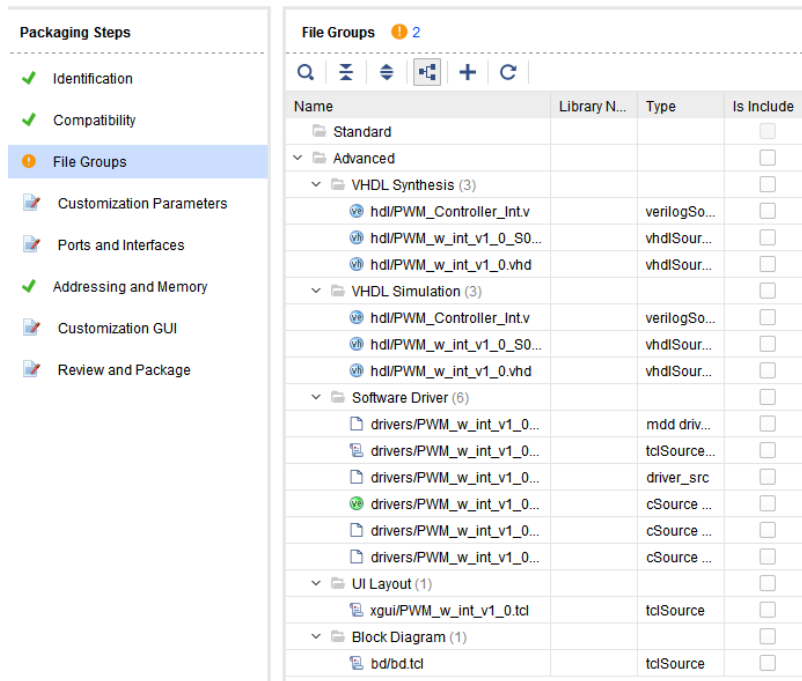


Figure 30 – File Groups

6. Select **Standard**, right-click and **Add File Group...** To use both VHDL and Verilog modules, select **Synthesis** and **Simulation** groups and click **OK**.

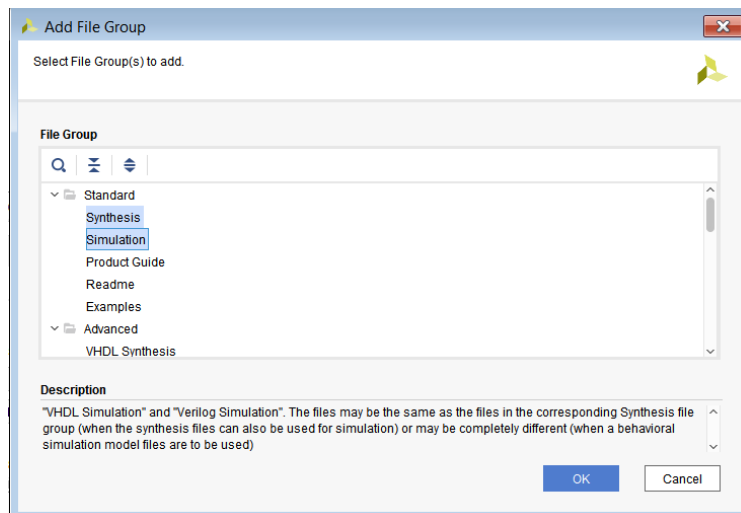


Figure 31 - Add File Group

7. Now select **Synthesis**, right-click and select **Add Files**.
8. When the Add IP Files window opens, click **Add Files**, Vivado should immediately open the HDL directory. If not, navigate to:

C:\Speedway\ZynqHW\2017\_1\IP\PWM\_w\_Int\_1.0\hdl

Change 'Files of type' option to **All Files**. Select **PWM\_Controller\_Int.v**, **PWM\_w\_Int\_v1\_0\_S00\_AXI.vhd** and **PWM\_w\_Int\_v1\_0.vhd**. Click **OK**. Make sure to not copy sources into project.

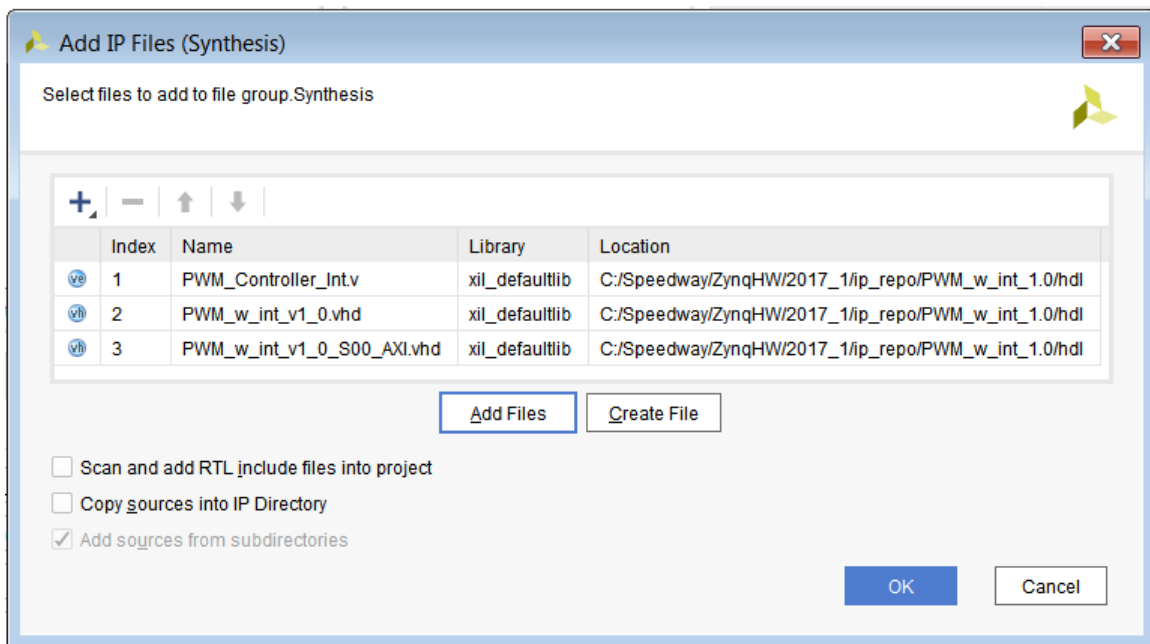


Figure 32 - Add HDL Files to IP File Groups

9. **Repeat** this process for *Simulation* group.
10. The IP Packager will give **2 errors**.

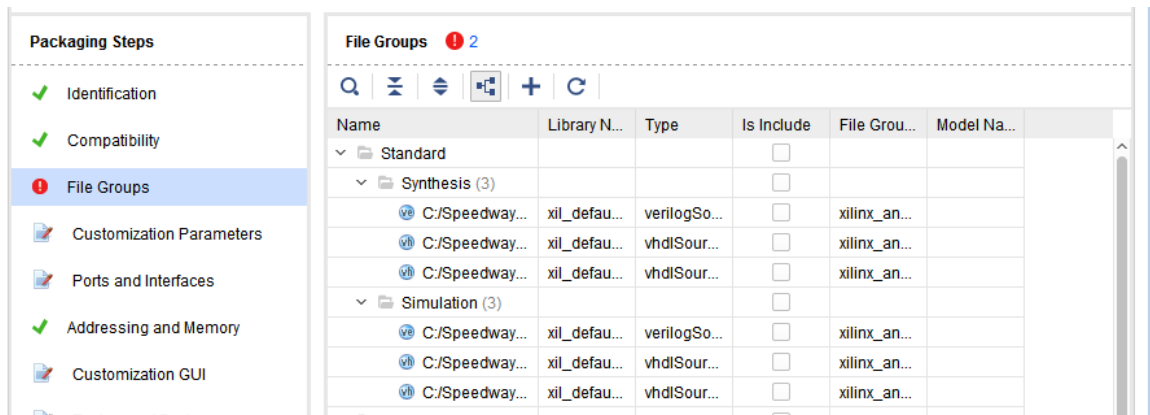


Figure 33 – Two Errors and Warnings

11. Specify **PWM\_w\_Int\_v1\_0** in **Model Name** for **Synthesis** and **Simulation** groups under the Standard folder. The IP File Groups should appear as follows now, and the 2 errors will disappear :

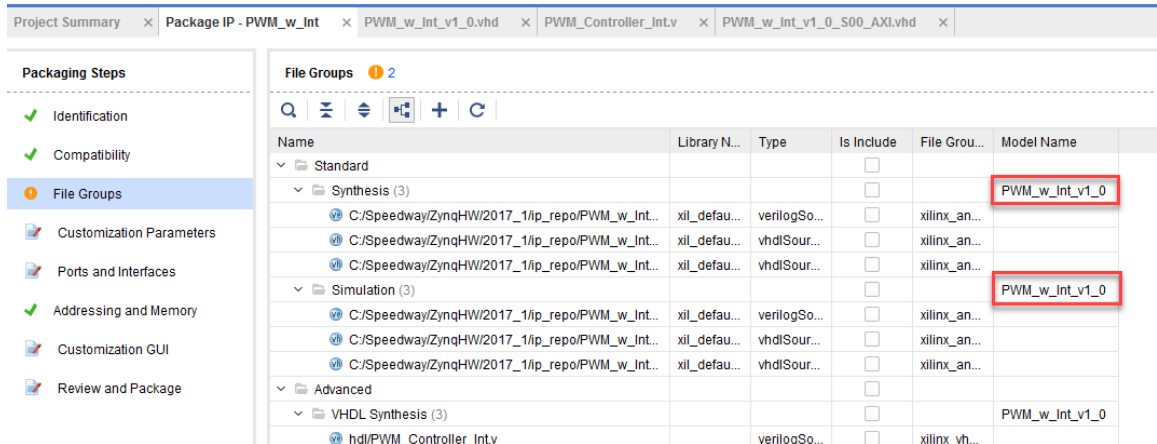


Figure 34 – Errors Disappear

12. Switch to the *Customization Parameters* page. Again, changes are detected and must be merged. Click **Merge changes...**

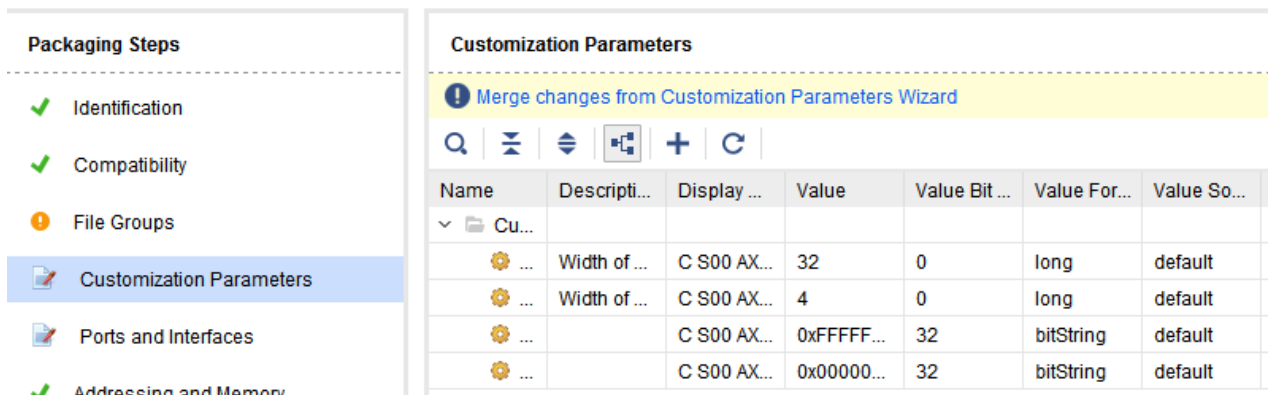


Figure 35 - IP Customization Parameters

Our **PWM\_PERIOD** parameter appears in the **Hidden Parameters** folder.

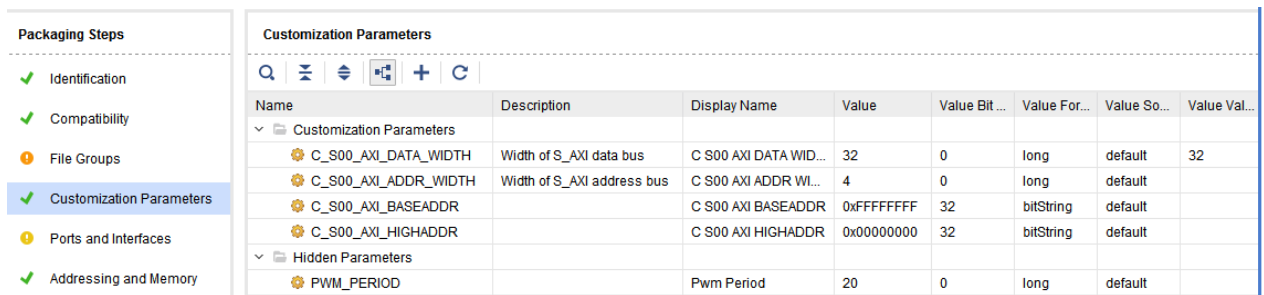


Figure 36 - User Parameters Added

13. Proceed back to File Groups and remove **VHDL Synthesis** and **VHDL Simulation** groups.

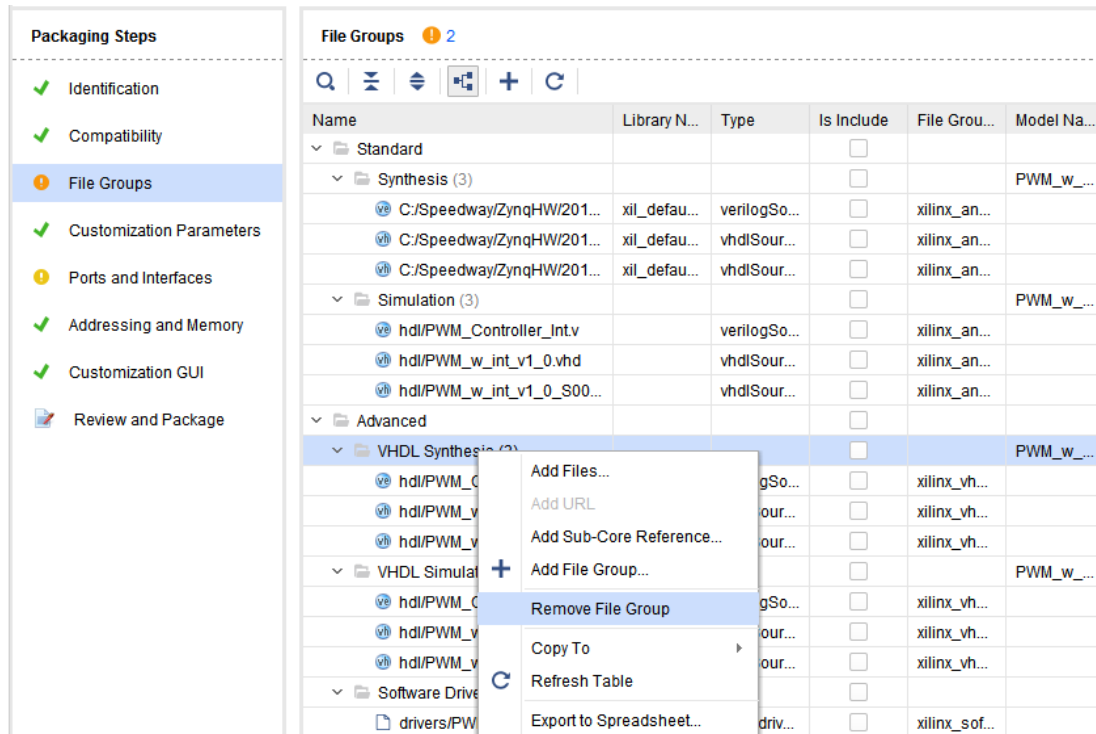


Figure 37 – Remove VHDL Synthesis and Simulation Groups

14. Return to Customization Parameters and double click on PWM\_PERIOD and check the **Visible in Customization GUI**, then click on **OK**.

Use the options below to customize how the parameter will appear in the Customization GUI for users of the IP.

Name: PWM\_PERIOD

☒ Visible in Customization GUI

☒ Show Name

Display Name: Pwm Period

Tooltip:

Format: long

Editable: Yes

Dependency: No

☐ Specify Range

Type: List of values

Press the + button to add a value

Show As: Not Applicable

Layout: Not Applicable

Default Value: 20

Shows how a parameter would appear in the Customization GUI. 'Radio Group' option is applicable if the number of values is more than one and less than five.

OK Cancel

**Figure 38 – Edit IP Parameter Window**

This will allow it to be viewed in the Customization Parameters folder and not labeled as a Hidden parameter.

Customization Parameters							
Name	Description	Display Name	Value	Value Bit ...	Value For...	Value So...	Value Val...
Customization Para...							
C_S00_AXI_DAT...	Width of S_AXI...	C S00 AXI DATA WIDTH	32	0	long	default	32
C_S00_AXI_ADD...	Width of S_AXI...	C S00 AXI ADDR WIDTH	4	0	long	default	
C_S00_AXI_BAS...		C S00 AXI BASEADDR	0xFFFFF...	32	bitString	default	
C_S00_AXI_HIG...		C S00 AXI HIGHADDR	0x00000...	32	bitString	default	
PWM_PERIOD		Pwm Period	20	0	long	default	

**Figure 39 - User Parameters Added to Customization Folder**



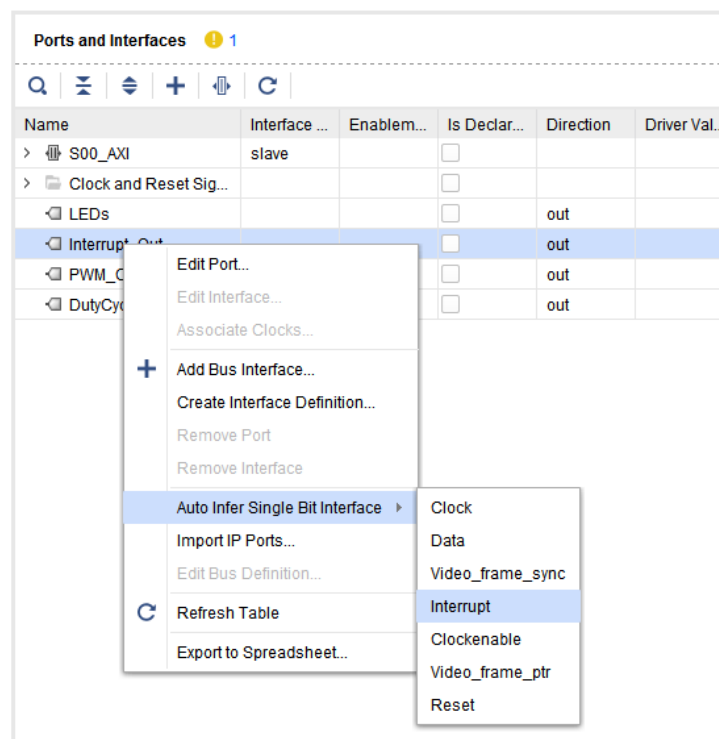
15. Next, select the *Ports and Interfaces* tab.

All of our user ports should be on the port list as shown below:

Ports and Interfaces <span>1</span>										
Name	Interface ...	Enablen...	Is Declar...	Direction	Driver Val...	Size Left	Size Right	Size Left ...	Size Righ...	Type Na...
> S00_AXI	slave		<input type="checkbox"/>							
> Clock and Reset Sig...			<input type="checkbox"/>							
LEDs			<input type="checkbox"/>	out		7	0			std_logic...
Interrupt_Out			<input type="checkbox"/>	out						std_logic...
PWM_Counter			<input type="checkbox"/>	out		19	0	(spirit:de...		std_logic...
DutyCycle			<input type="checkbox"/>	out		31	0			std_logic...

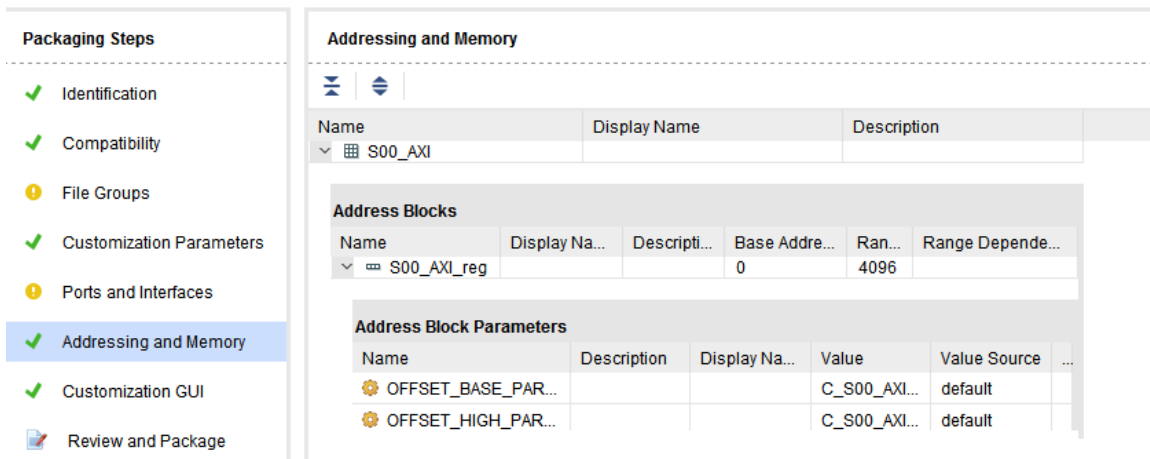
**Figure 40 - User IO Ports**

16. Right click on **Interrupt\_Out** → **Auto Infer Single Bit Interface** → **Interrupt** to define the interrupt



**Figure 41 – Defining Interrupt**

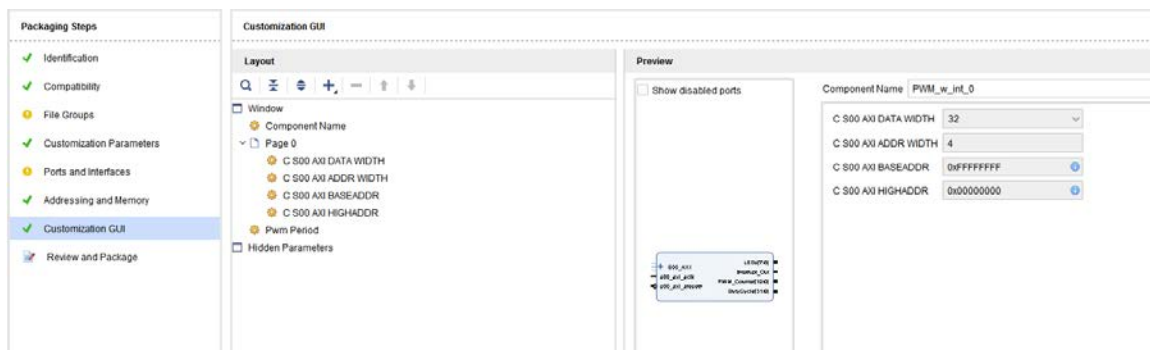
## 17. Move to **Addressing and Memory**.



**Figure 42 - Addressing and Memory Map**

There is no memory or BRAM in this IP. This screen lists the memory map for the Slave AXI interface.

## 18. Next, select the *Customization GUI* tab.



**Figure 43 - IP GUI Preview**

## 19. Feel free to move the **Pwm Period** parameter up under page 0 if you want this location in the GUI

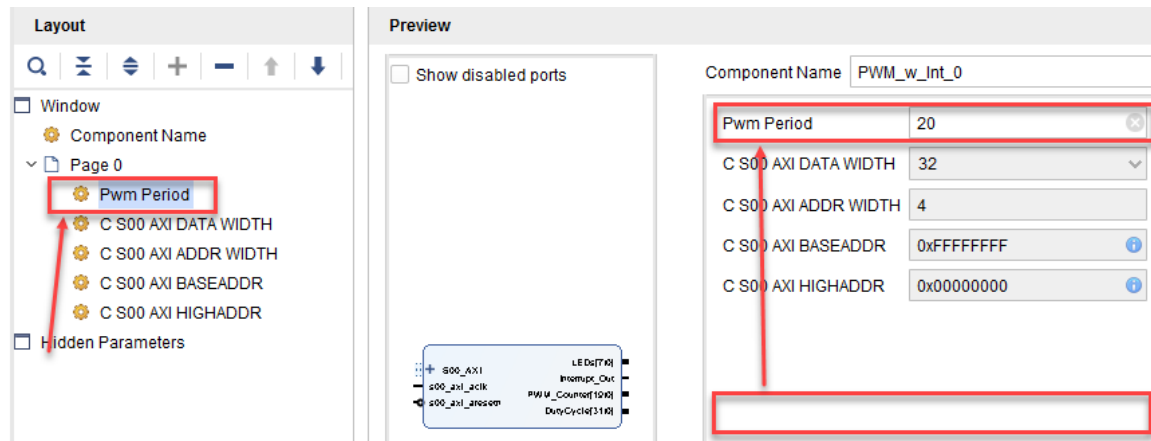


Figure 44 – Customization GUI

20. Select **Review and Package** to complete the IP Packaging.
21. Review the settings and click the **Package IP** button. If you were to edit this IP package later on and re-package the edits you make, you will then see a button labeled **Re-Package IP** here instead.

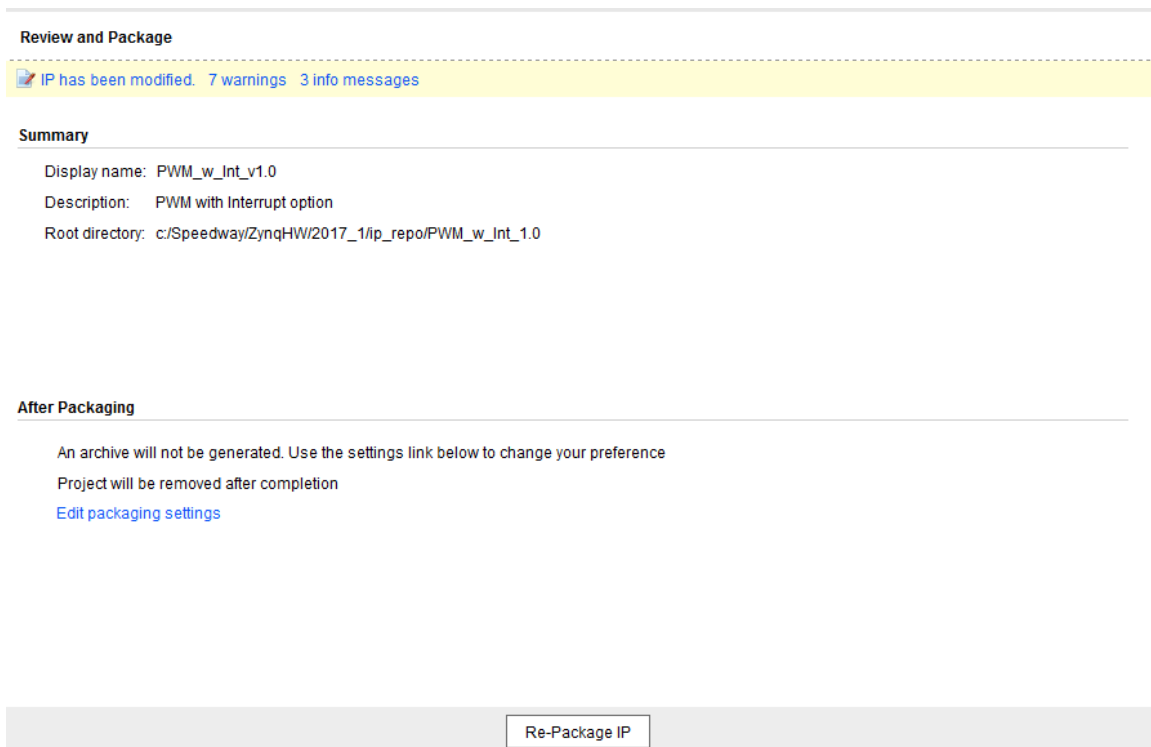


Figure 45 - Package IP

22. When finished, it will ask to close the IP project. Click Yes.

23. Using Windows Explorer, navigate to our IP repository. Notice our IP project is now in the repository:

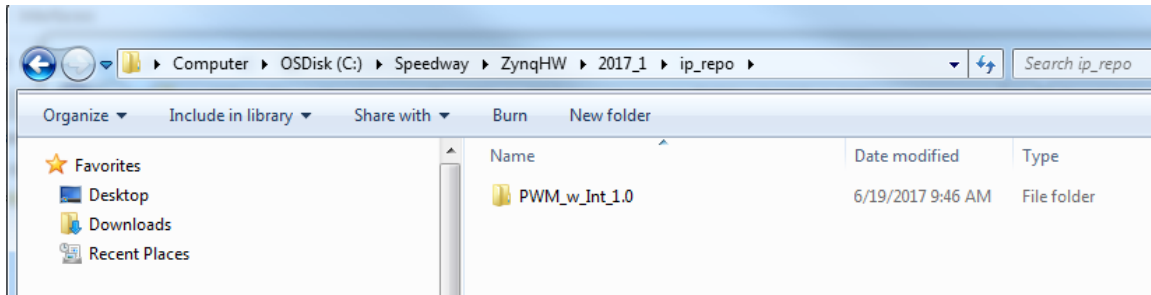


Figure 46 - IP Repository

24. Open the **PWM\_w\_Int\_1.0** folder. Inside are directories for the HDL files as well as for drivers. Lastly, the **component.xml** file is our package IP that is read in by Vivado.

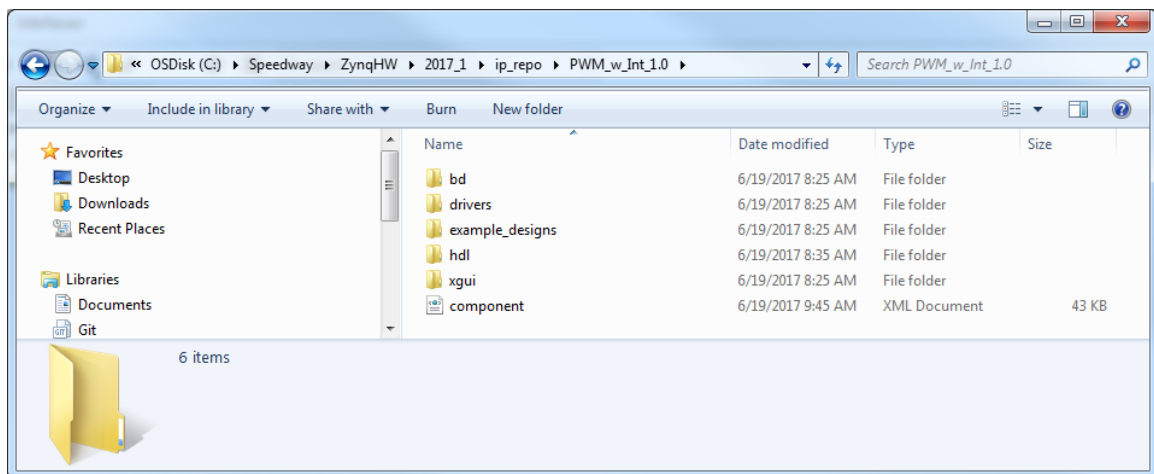


Figure 47 - IP File Structure and Directories

That's it, which completes our creating of our IP. Next, we'll add it to our design!

## Experiment 4: Add IP to Project

This experiment shows how to add the new user IP to our Vivado project. Once added, we'll drop it into our block design and connect it to our AXI Interconnect Block. Additionally, we'll hook up the IP outputs to an Integrated Logic Analyzer core to display the outputs.

### Experiment 4 General Instruction:

Import New IP into the project and connect it to the design.

### Experiment 4 Step-by-Step Instructions:

1. Return to our **ZynqDesign** Vivado Project.
2. Open the block design.

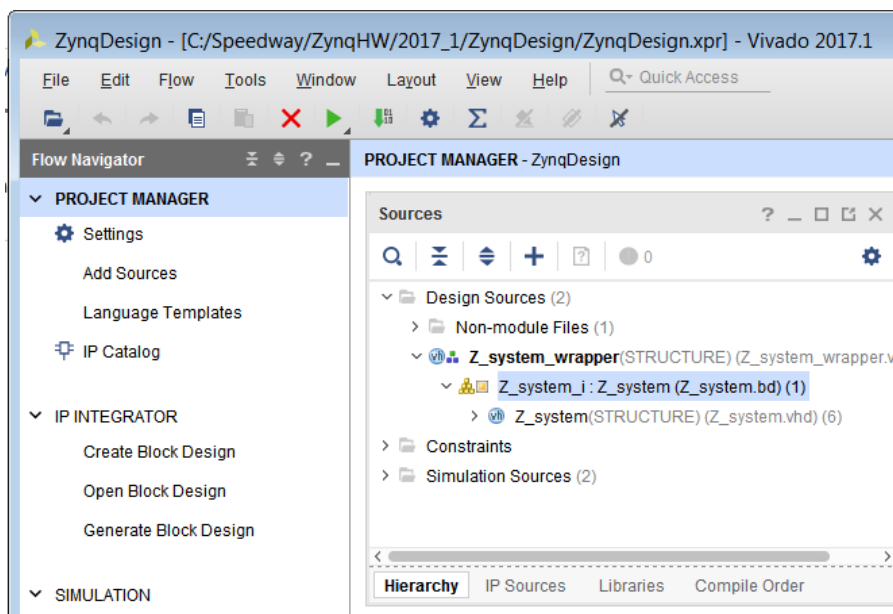


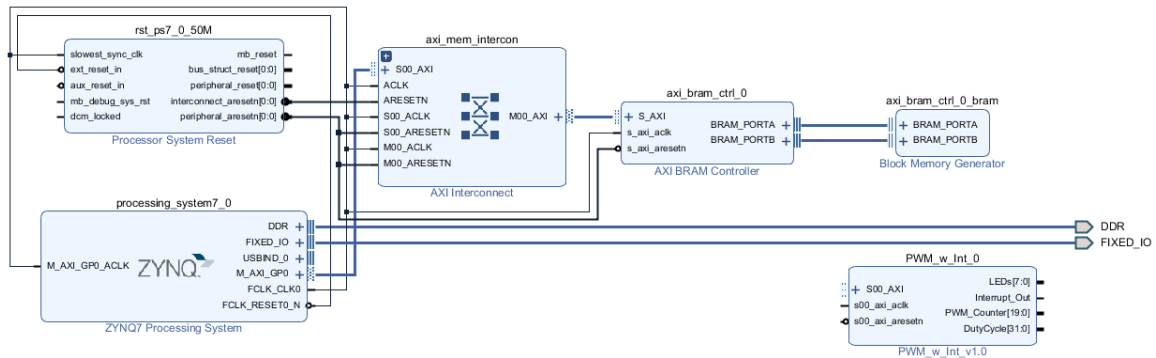


Figure 48 - Open Block Design

3. Select the *Add IP* icon  or right-click inside the *Diagram Window* and select *Add IP*.
4. Enter "PWM" and Double-click **PWM\_w\_Int\_0** to add it to the design.

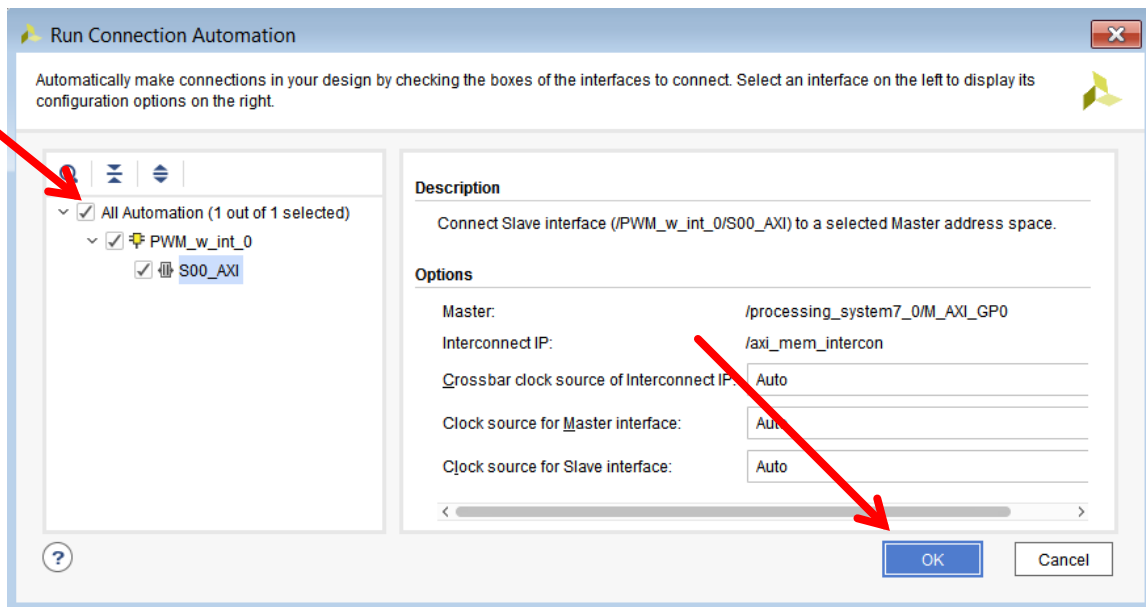
- Click the **Regenerate Layout** button from the vertical shortcut bar . This is what should be displayed:



**Figure 49 - IP Core Inserted**

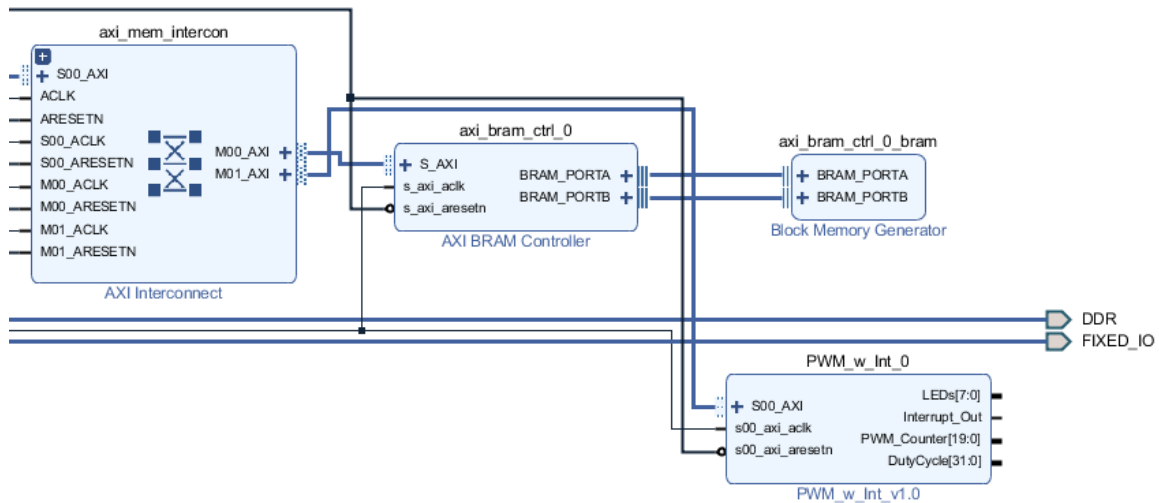
As we asked in an earlier lab, how would we connect additional AXI slave IP? One answer, add more AXI Master Interfaces to the AXI Interconnect core. A second possibility, which we will not do now, is to enable and use the second general purpose AXI Master port from the ZYNQ7 Processing System. However, the tool is going to take care of this for us!

- Click **Run Connection Automation**.
- Make sure **All Automation** is checked, then click **OK**.



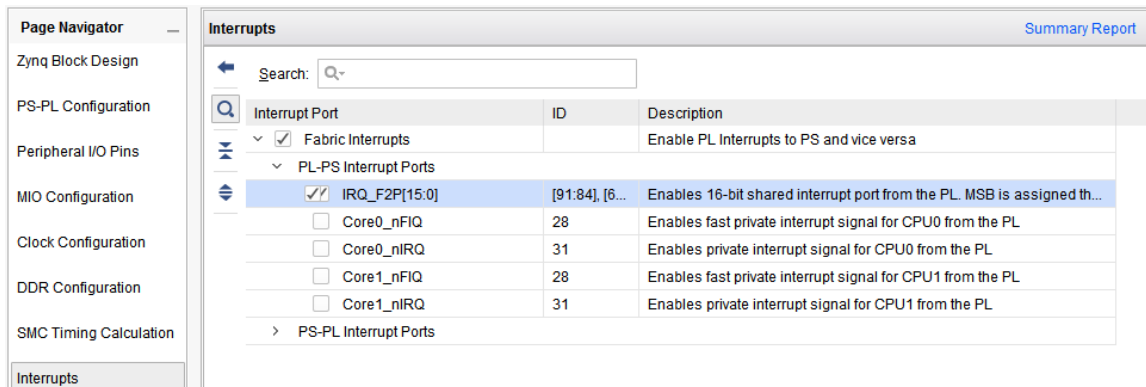
**Figure 50 – Run Connection Automation for PWM**

The automation properly adjusts the AXI Interconnect block to have a 2<sup>nd</sup> Master. Then the AXI, clock, and reset are connected, as shown below.



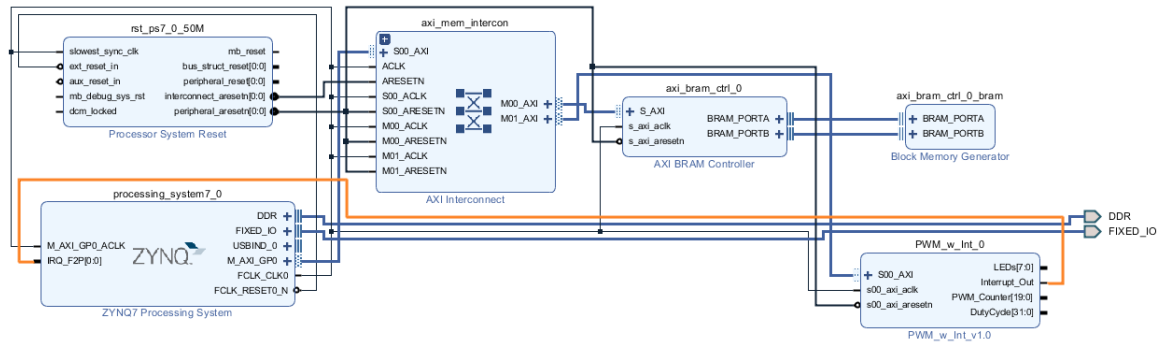
**Figure 51 – PWM Controller AXI, clk, and reset Connected**

8. The PWM\_w\_Int IP has an interrupt output, but the ZYNQ7 Processing System IP block has not been enabled to accept PL interrupts yet. **Double-click** the ZYNQ7 Processing System IP block.
9. Select **Interrupts** from the Page Navigator.
10. Check the box for **Fabric Interrupts** then **expand** it.
11. Expand **PL-PS Interrupt Ports** and check the box next to **IRQ\_F2P[15:0]**. F2P is Fabric to PS. When done, click **OK**.



**Figure 52 - Add PL-PS Interrupts**

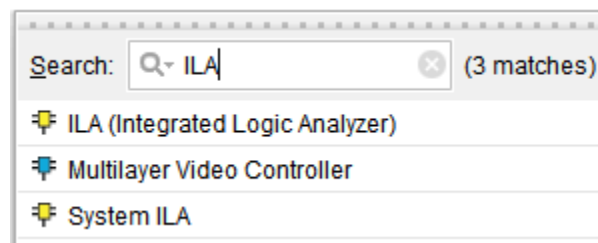
12. Connect **Interrupt\_Out** from the PWM\_w\_Int\_v1.0 IP block to **IRQ\_F2P[0:0]** on the ZYNQ7 Processing System IP block by clicking on either pin creating a line then clicking on the other pin.



**Figure 53 - Connected Interrupt**

Finally, we are left with the PWM outputs.

13. Access the **Add IP** again, search for **ILA**:

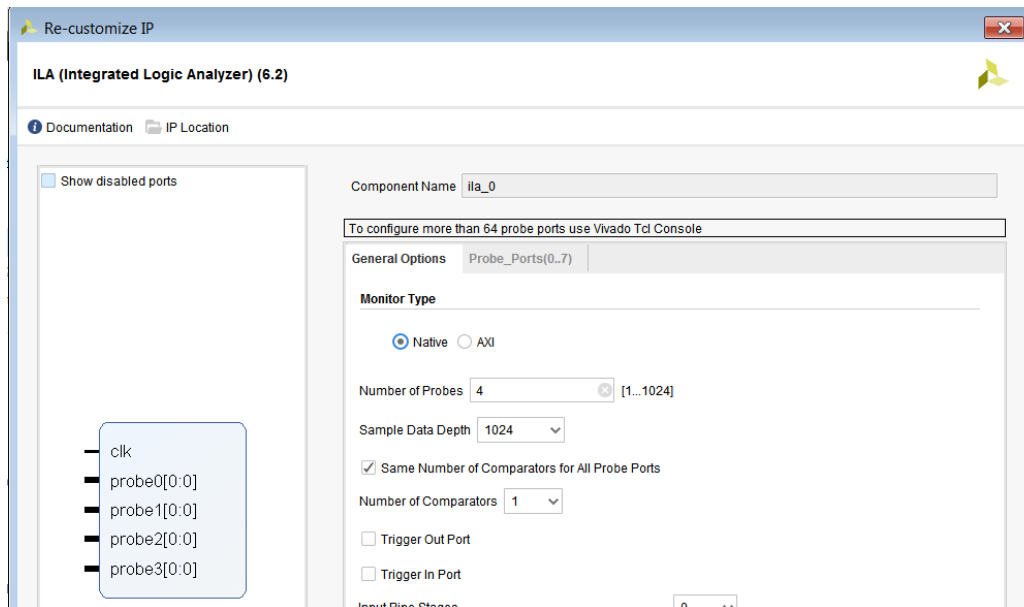


**Figure 54 - ILA IP Core**

14. **Double-click** this IP to add it to the design.
15. **Double-click** the ILA IP to customize it.



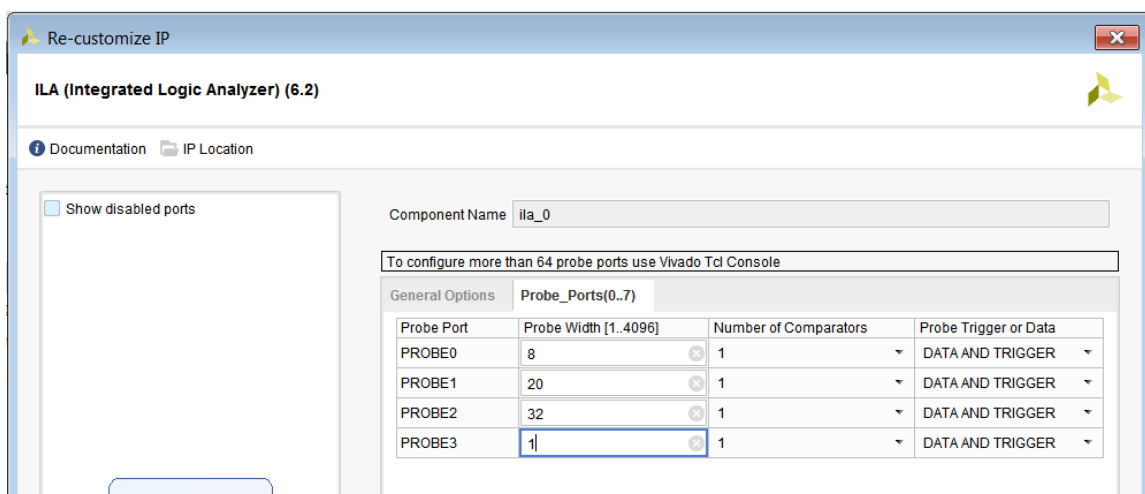
16. Set the *Monitor Type* to “Native” and set the number of *Probes* to 4. Then click the **Probe\_Ports(0...7)** Tab.



**Figure 55 - Customize ILA IP**

17. Change the Probes Widths to match the following:

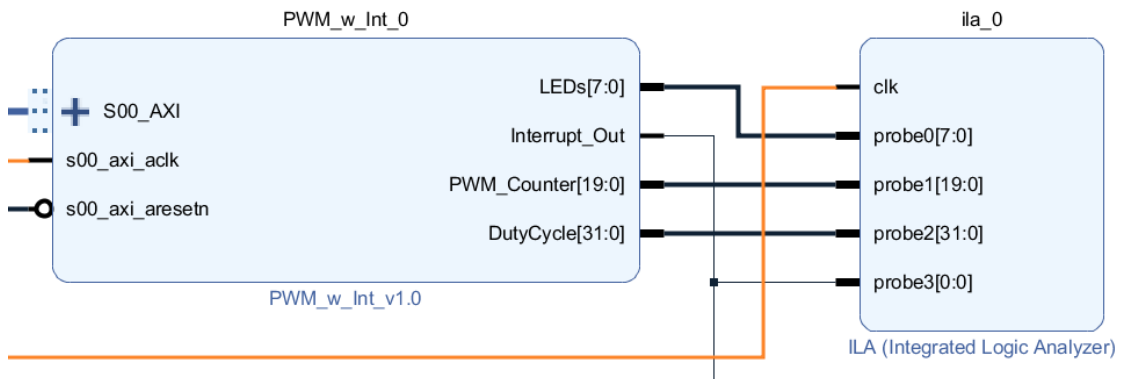
- PROBE0 = 8
- PROBE1 = 20
- PROBE2 = 32
- PROBE3 = 1



**Figure 56 - ILA Probes Configuration**

18. Click **OK**.

19. Connect **LEDs** to **probe0**, **PWM\_Counter** to **probe1**, **DutyCycle** to **probe2** and **Interrupt\_out** to **probe3**. Connect the clk to **FCLK\_CLK0**.



**Figure 57 - Connect ILA and PWM**

20. Now that we've added new IP, we must update our addressing. Click the **Address Editor** Tab in the block design. Click **Auto Assign Address** if the address is not already assigned.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF
PWM_w_int_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF

**Figure 58 - Assigned IP Addressing**

That completes connecting our custom IP to our design. We've created completely new IP, packaged it in the IP Packager, imported it into our design and connected it to the Zynq Processing System. In the next lab we exercise this IP through a debug interface and then add a software application to interact with it.

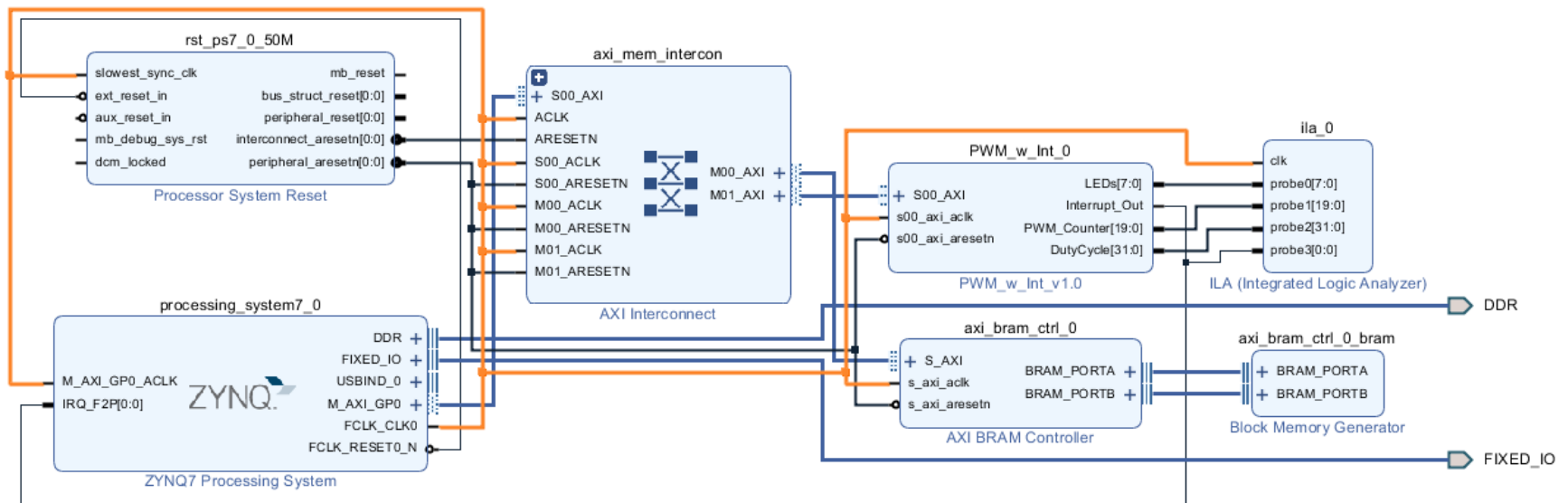


Figure 59 - Fully Connected Design

## Experiment 5: Add LogiCORE™ IP JTAG-AXI core

Before we continue, we'll add one more component that we'll use in the next lab. This experiment shows how to add a JTAG-AXI core. We'll explain this IP in the next section, but it's easier to add it now as we will generate a bitstream at the end of this lab.

### Experiment 5 General Instruction:

Add JTAG to AXI Master IP into the project and connect it to the design.

### Experiment 5 Step-by-Step Instructions:

1. In the block design, select **Add IP** then add the **JTAG to AXI Master** Core (search for JTAG).

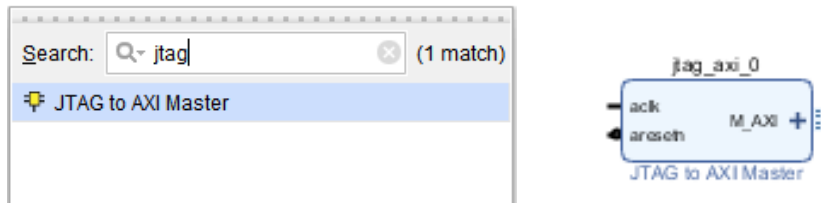


Figure 60 - Add JTAG to AXI Master

The JTAG to AXI Master needs a slave interface to connect into. Note this IP is an AXI Master, thus it will initiate AXI transactions with an AXI slave. By connecting this directly into the AXI Interconnect, the JTAG to AXI Master will be able to interact with all peripherals connected to that interconnect block.

2. Click **Run Connection Automation**. Make sure **All Automation** is selected, then click **OK**.

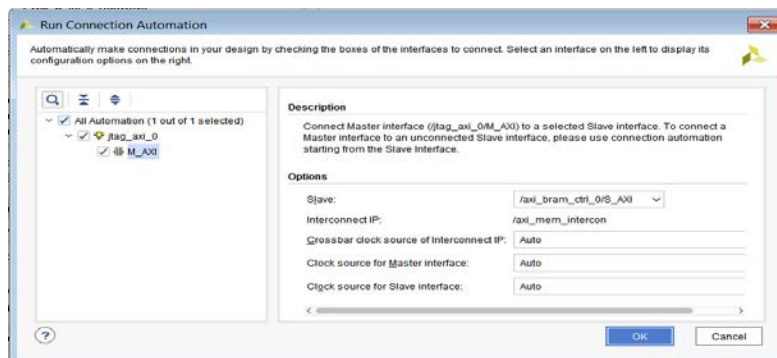


Figure 61 – Run Connection Automation on JTAG AXI

The final block design should appear as follows:

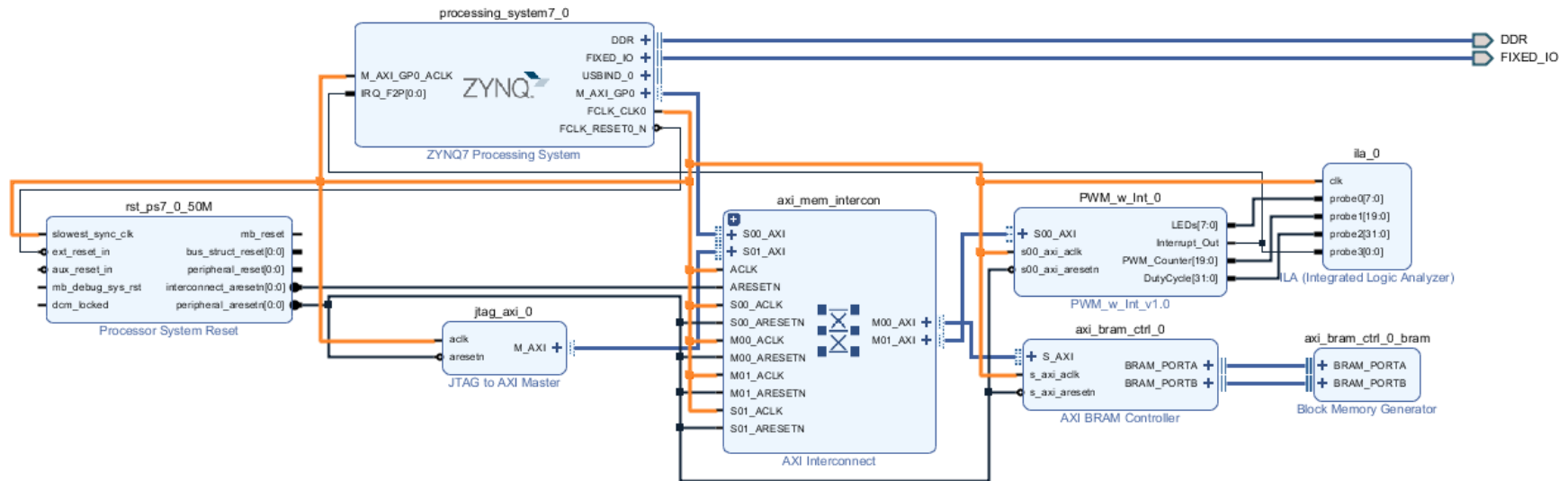


Figure 62 – Final Block Design

3. **Validate** the block design. If OK, **Save** the block design.
4. Right-click on Z\_system\_i in the Sources tab and **Reset Output Products** then **Generate Output Products**.
5. In the Flow Navigator pane, click **Generate Bitstream**. This will take a few minutes depending on your PC. Also allow synthesis and implementation to run when prompted to.
6. Open implemented design when bitstream is successfully written.

### Questions:

**Answer the following questions:**

- *How many AXI Masters and Slaves can be added to the AXI Interconnect?*

---

- *How many interrupts can be generated from fabric resources?*

---

This concludes Lab 7.

## Revision History

Date	Version	Revision
6 Nov 13	02	Initial Draft
19 Nov 13	03	Pilot updates
6 Nov 14	04	Updated for Vivado 2014.3
5 Jan 15	05	Updated for Vivado 2014.4
06 Mar 15	06	Finalize for Vivado 2014.4. Update dma_test.c to use valid xil_types and also for BRAM xparameters.h bug in 2014.4
16 Mar 15	07	Minor edits for release
Oct 15	08	Updated to Vivado 2015.2
July 2016	09	Updated to Vivado 2016.2
May 2017	10	Updated to Vivado 2017.1
June 2017	11	Updated to Vivado 2017.1 for MiniZed + Rebranding
Nov 2017	12	Minor Update

## Resources

[www.minized.org](http://www.minized.org)

[www.microzed.org](http://www.microzed.org)

[www.picozed.org](http://www.picozed.org)

[www.zedboard.org](http://www.zedboard.org)

[www.xilinx.com/zyng](http://www.xilinx.com/zyng)

[www.xilinx.com/sdk](http://www.xilinx.com/sdk)

[www.xilinx.com/vivado](http://www.xilinx.com/vivado)

## Answers

### Experiment 1

- *Where is the IP project located?*

**In the same directory as our current project:**

**C:\Speedway\ZynqHW2017\_1\ip\_repo**

### Experiment 5

- *How many AXI Masters and Slaves can be added to the AXI Interconnect?*

**16 and 16.**

- *How many interrupts can be generated from fabric resources?*

**8 per M\_AXI\_GP = 16**

**Plus nFIQ & nIRQ per core = 4**

**20 Total**