You will need to start the process of creating custom IP from some existing Vivado project. This project does not need to have any relationship with the custom IP that you are creating in this howTo.

The goal of this step is to create the entity/architecture for the enhancedPwm\_ip\_v1\_0 component shown in the diagram below. Look at Figure 1 and find the enhancedPWM component. Let’s explore where the signal on its entity are being routed to.

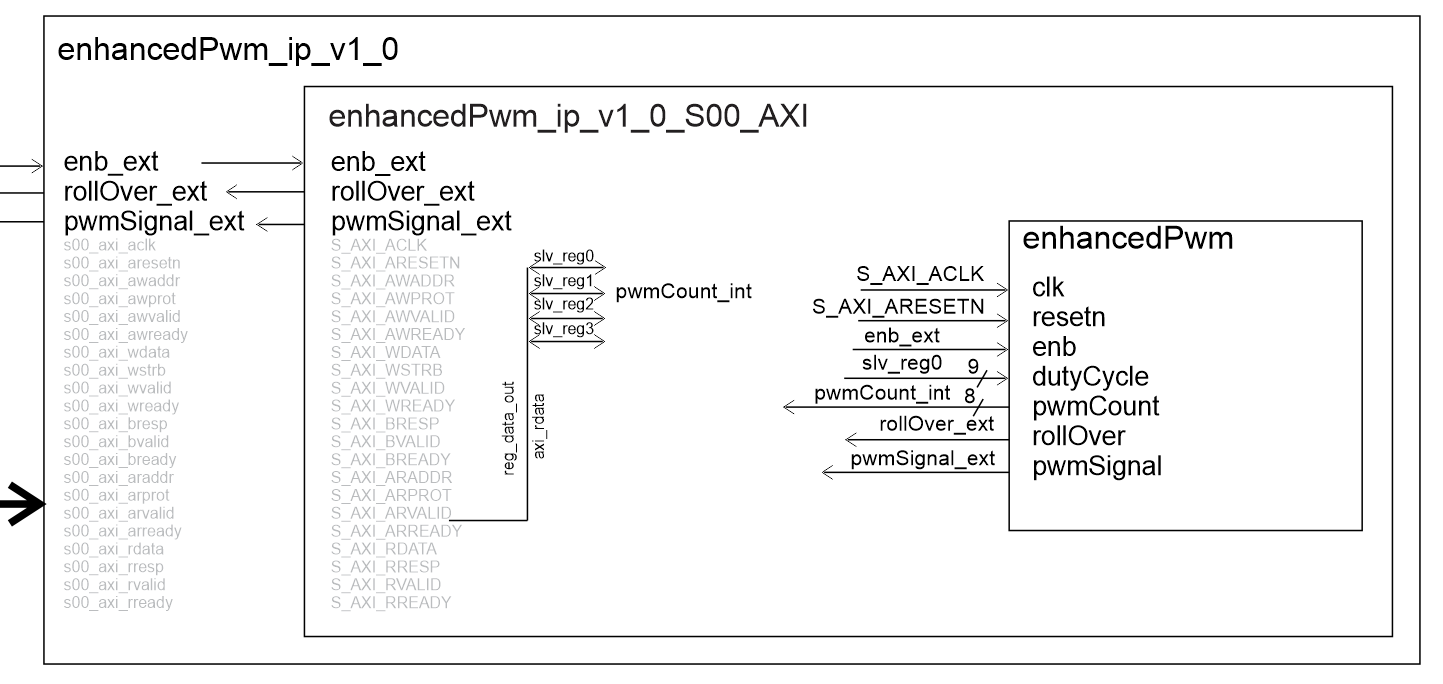


Figure : The hierarchy of components needed to interface a custom module to the ARM cortex and the outside world.

The signals on the enhancedPwn interface will come from several different sources:

* clk and reset will come from the S\_AXI signals coming in through the enhancedPwm\_ip\_v1\_0\_S00\_AXI interface.
* enb, pwmSignal and rollOver will go outside the Zynq chip and connect to PL\_KEY4, pin 3 on header J11 and the interrupt input on the Zynq respectively.
* dutyCycle and pwmCount will be written/read by a program that we will write for the ARM processor in a later stage.

The grey signal names on the entities in Figure 1 are part of the AXI interface and as such, not your concern – leave them alone. When a signal from your custom IP needs to leave the Zynq chip, you must add that signal to the two “enhancedPwm\_ip\_v1\_0” interfaces.

When a signal from your custom IP needs to interface to the ARM processor, you need to deal with the slv\_regX interface.

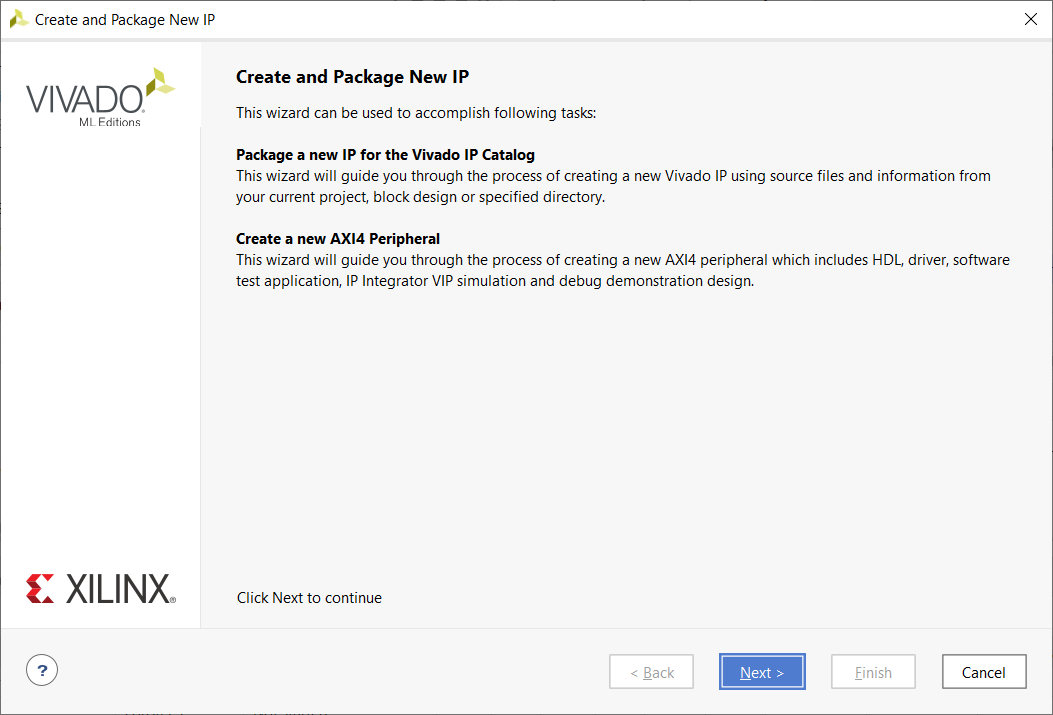
The following process is very detailed and you can easily make errors. So take your time, mind the details and try to stay focused. Good luck.

**Create Custom IP:**

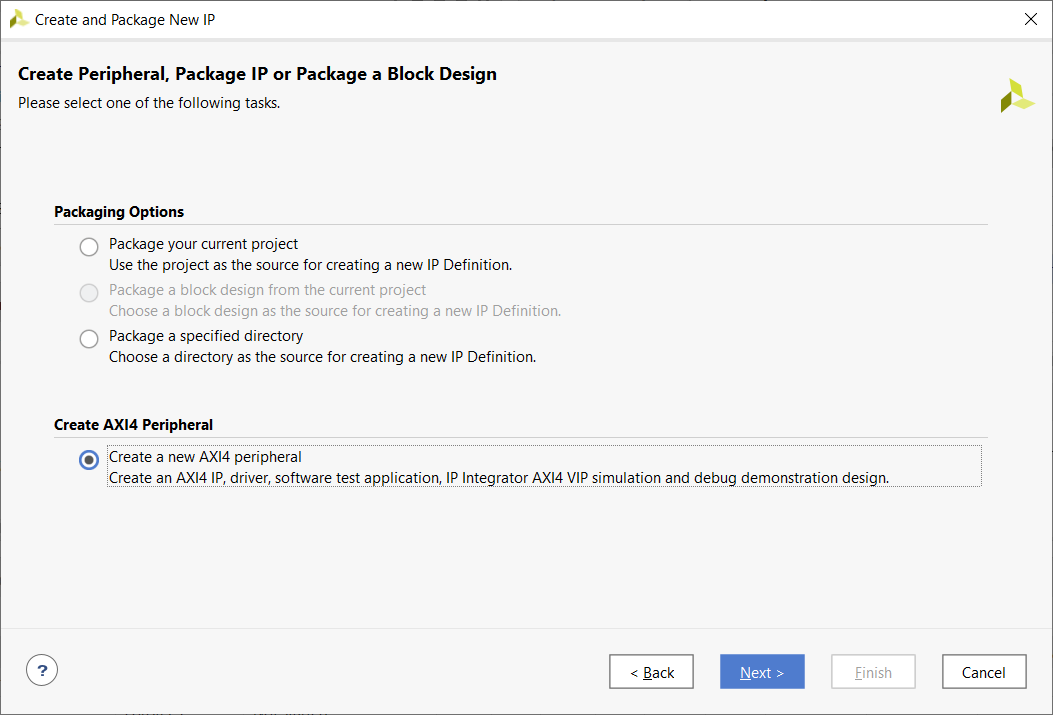
Start by selecting Tools => Create and Package New IP….

You will be presented with the Create and Package New IP wizard:

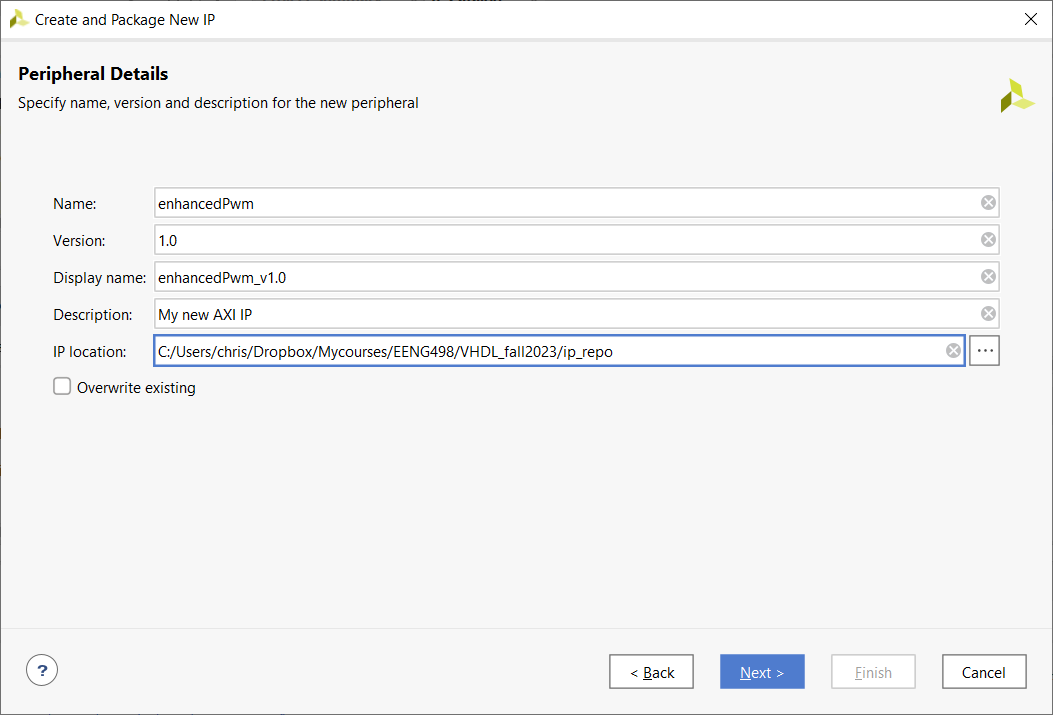
**Create Peripheral, Package IP:** Click Next



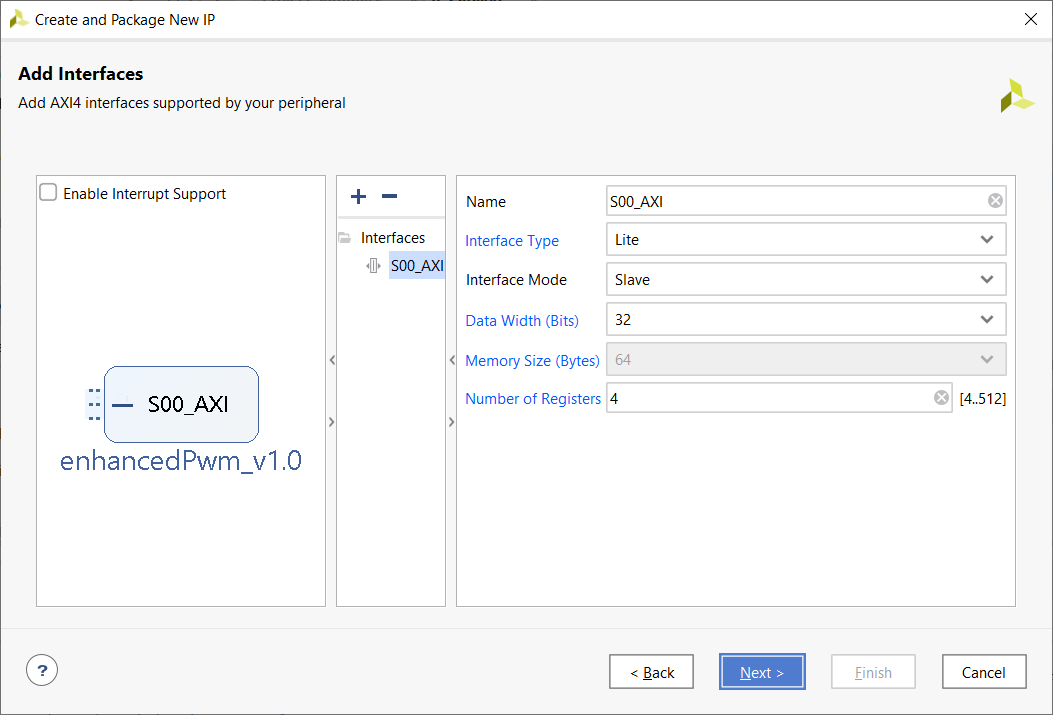
**Create Peripheral, Package IP:** Select Create AXI4 Peripheral. Click Next.



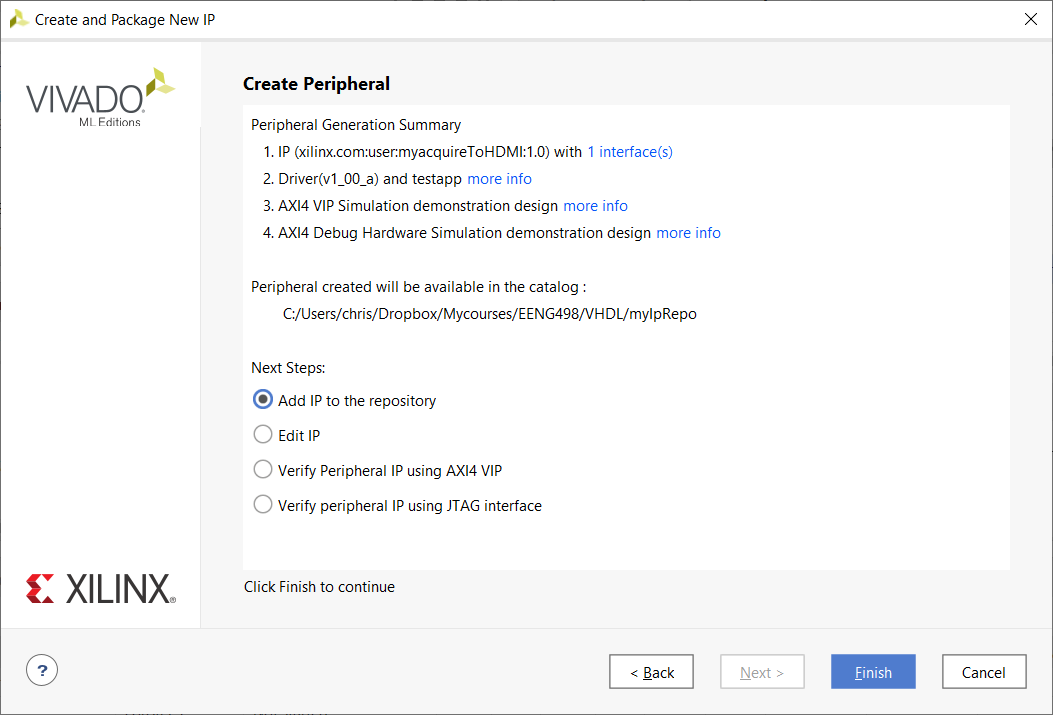
**Peripheral Details:** Complete name and IP location. Click Next.



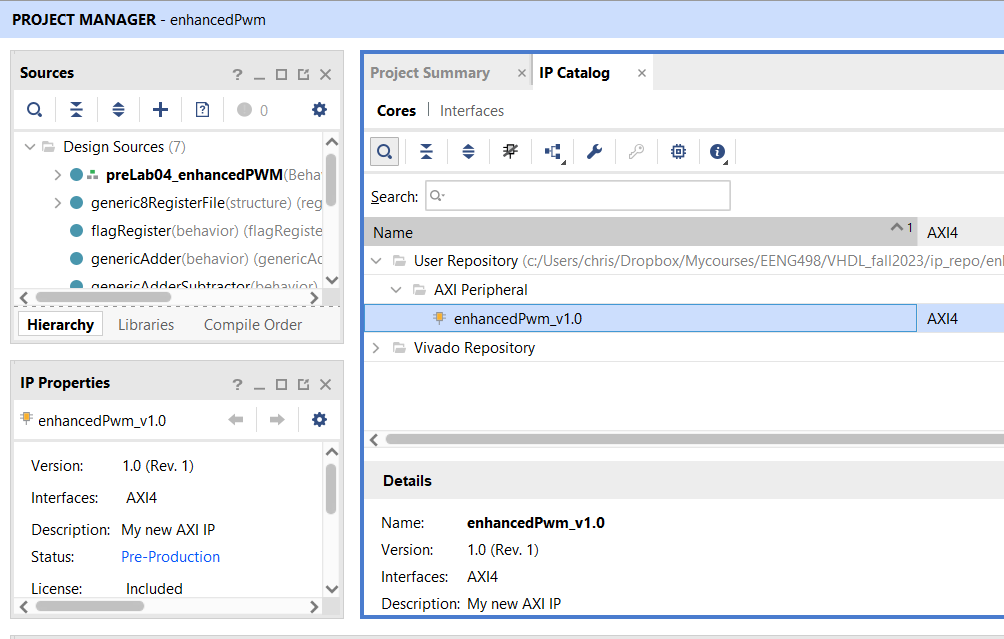
**Add Interfaces:** Select 4 registers, leave all the other defaults alone. Click Next.



**Create Peripheral:** Select the Add IP to repository radio button. Click Finish.



You have just created an empty shell of the new IP.



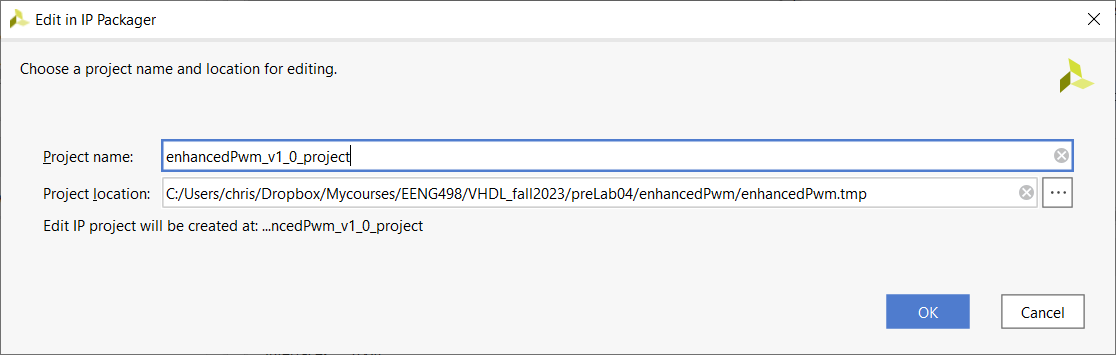
Before moving on, check to make sure that your IP wrapper will be created in VHDL. Do this my right mouse clicking on the enhancedPWM\_v1.0 IP element and selecting IP Settings…

In the Settings pop-up, select Project in the Tool Settings area. Make sure that VHDL is selected as the Target Language.

A screenshot of a computer

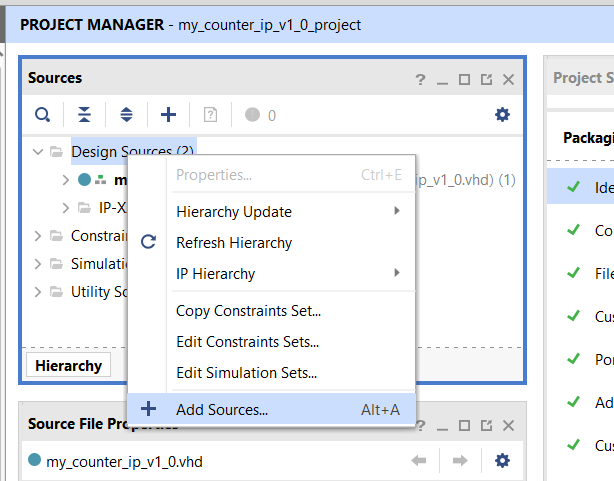
Description automatically generated with medium confidence

It’s time to fill that shell with your custom logic. To do go to the IP Catalog tab, expand the User Repository -> AXI Peripheral and then right click on “enhancedPwm\_ip\_v1.0” and select “Edit in Packager”

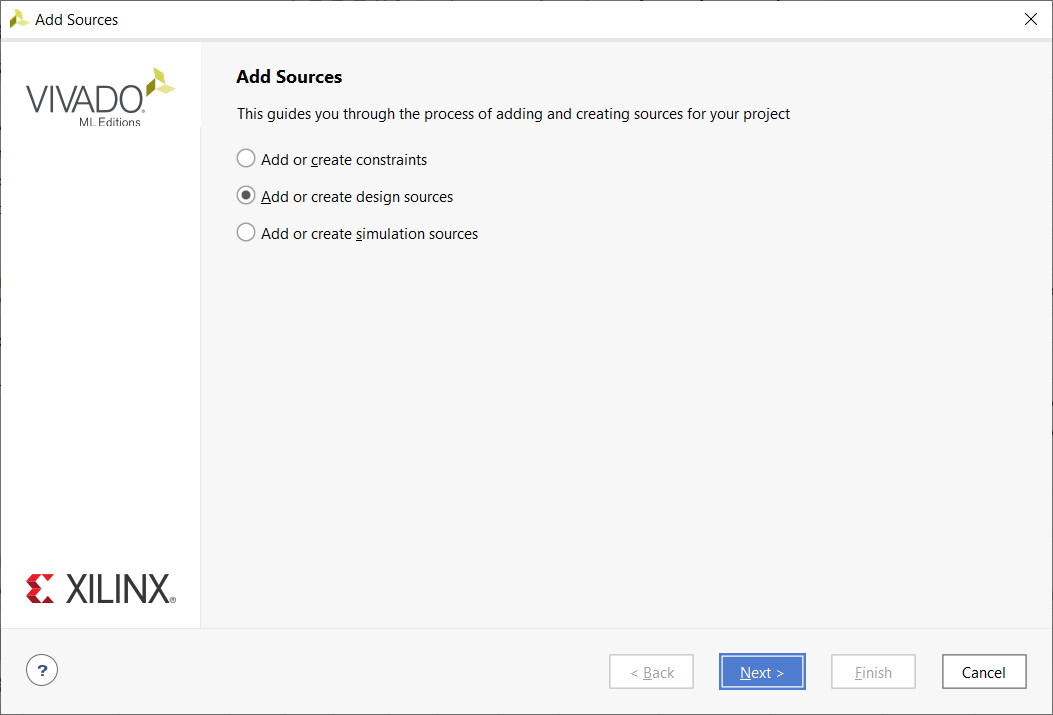


Leave the defaults. Click OK. If you are asked, click OK, to overwrite the previous version. A new invocation of Vivado will launch.

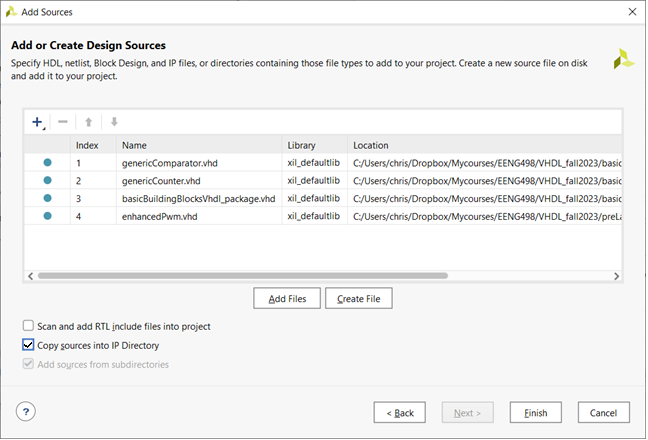
You will now package the enhancedPwm component into a form that will make it compatible to the AXI bus used by the Zynq processor. Start by adding enhancedPwm to the project. Do this by right clicking on Design Sources and selecting Add Sources…



This will launch the Add Sources pop-up.



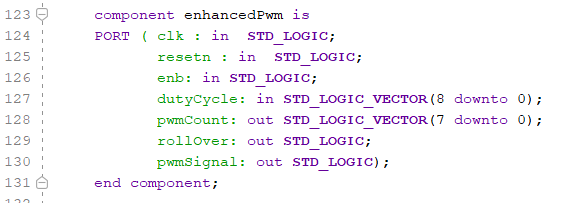
Make sure “Add or create design sources” is selected. Click Next. This will launch the Add or Create Design Sources pop-up.



Click Add Files. In the Add Source Files pop-up, navigate to the enhancedPwm, select the file, then click Open. Do the same for the genericCounter, genericCompare and basicBuildingBlocks\_package. Make sure to check the “Copy sources into IP Directory. If you do not check this box and then delete this IP unit for any reasons, the linked sources files will also be deleted. Ask me how I know :/ Click Finish.

Edits to: enhancedPwm\_ip\_v1\_0\_S00\_AXI

Start by declaring the enhancedPwm as a component inside enhancedPwm\_v1\_0\_S00\_AXI.vhd Open the enhancedPwm.vhd file, copy the entity description for the enhancedPwm and then paste it into the enhancedPwm\_v1\_0\_S00\_AXI.vhd file below the signal declarations and just before the begin statement in the architecture. Then change the entity description into a component description as shown below.

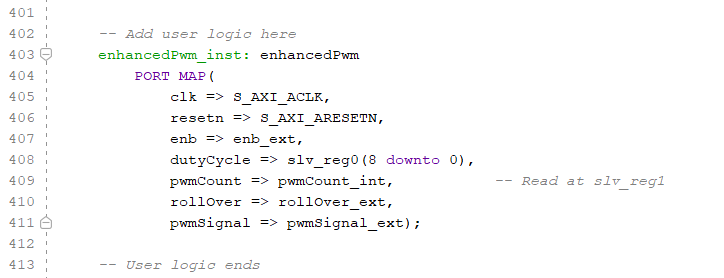


\*Please note, your line numbers will be different as this screen shot represent the file when it was completed. You have only just started editing the file.

You now need to instantiate the enhancedPwm component inside the AXI file. Do this as follows.

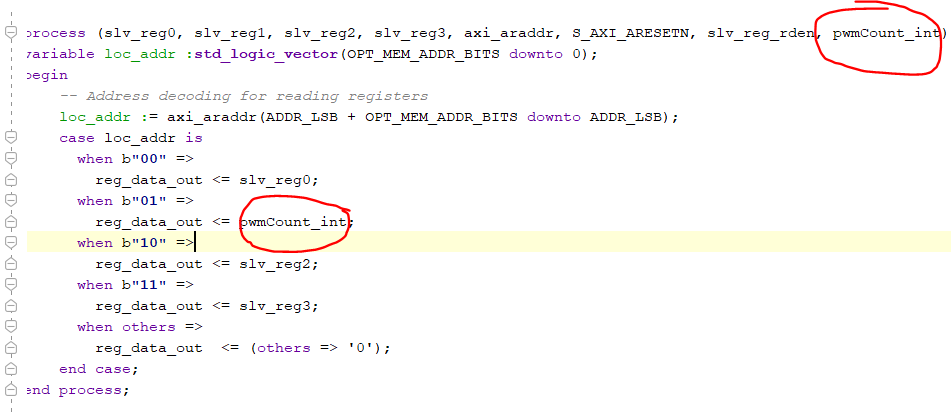
In the “add user logic” section of the AXI file, instantiate the enhancedPwm component. The instantiation will:

* Have the ARM set the value of the dutyCycle input through the slv\_reg0.
* Send the pwmCount output to a local signal called pwmCount\_int and on the ARM through slv\_reg1.
* The clk is provided by some external signal, S\_AXI\_CLK. You will see later that this is the common bus shared by all peripherals. This is important because it keeps your custom hardware synchronized with the ZYNQ processor.

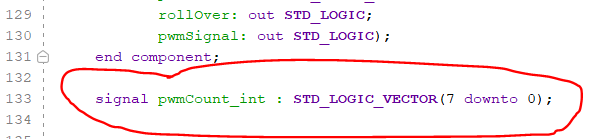


\*Please note, your line numbers will be different as this screen shot represent the file when it was completed. You have only just started editing the file.

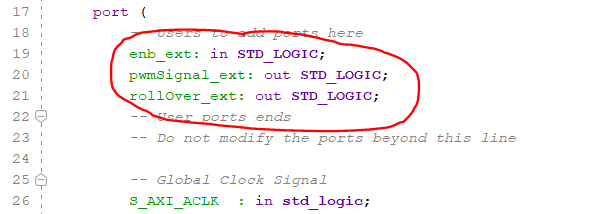
Line 370: Add the pwmCount\_int signal as the value for reg\_data\_out in place of slv\_reg1. This will allow the pwm counter to be read by the Zynq processor when the Zynq processor tries to read slv\_reg1. Also add pwmCount\_int to the sensitivity list.



Just below your enhancedPwm component declaration you added earlier, add the declaration of pwmCount\_int signal.



Line 19: Add signals to the port description of enhancedPwm\_ip\_v1\_0\_S00\_AXI. You will need to route these signals outside the Zynq chip. The 1-bits enb\_ext will get its value from PL KEY4 and pwmSignal\_ext will go to LED4 and rollover\_ext will go to LED3.

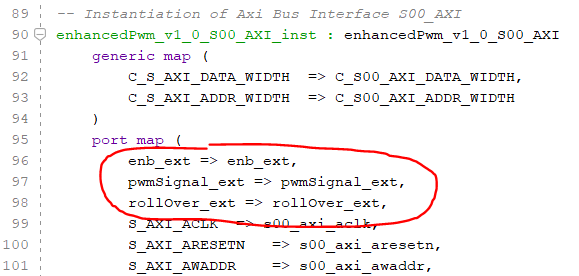


You are done editing enhancedPwm\_ip\_v1\_0\_S00\_AXI. It’s now time to edit this file’s wrapper enhancedPwm\_ip\_v1\_0

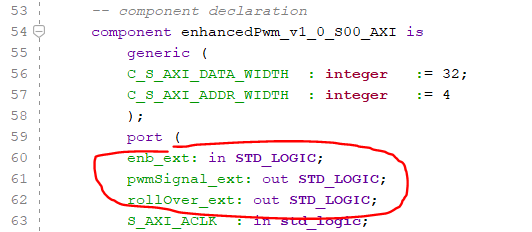
Edits to: enhancedPwm\_ip\_v1\_0

You will need to send the enb\_ext, pwmSignal\_ext and rollover\_ext signals outside this module. It’s fairly straightforward.

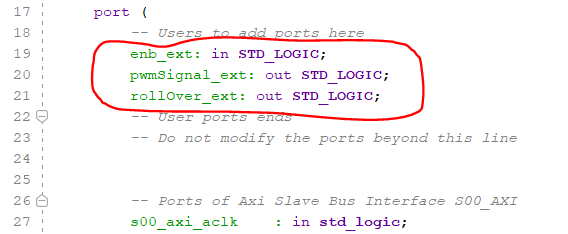
Line 94: Add signals to instantiation of enhancedPwm\_ip\_v1\_0\_S00\_AXI.



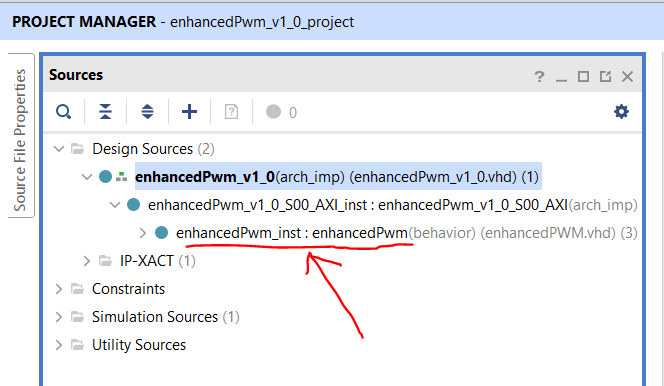
Line 59: Add signals to component declaration of my\_counter\_ip\_v1\_0\_S00\_AXI



Line 19: Add signals to port description of my\_counter\_ip\_v1\_0



Save all the files and check the sources area of the project manager tab once it is done updating. The enhancedPwm component should be part of the hierarchy and there should be no syntax errors.

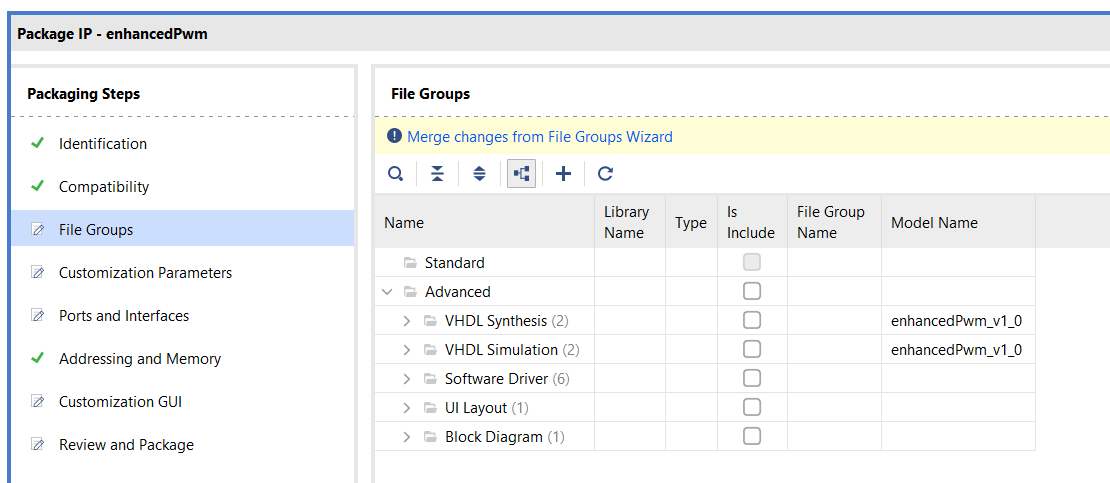


Now it’s time to finish packaging your IP. You will go through the packing steps in the Package IP tab. Each of these is taken in turn below. If you do not see the Package IP tab, click on

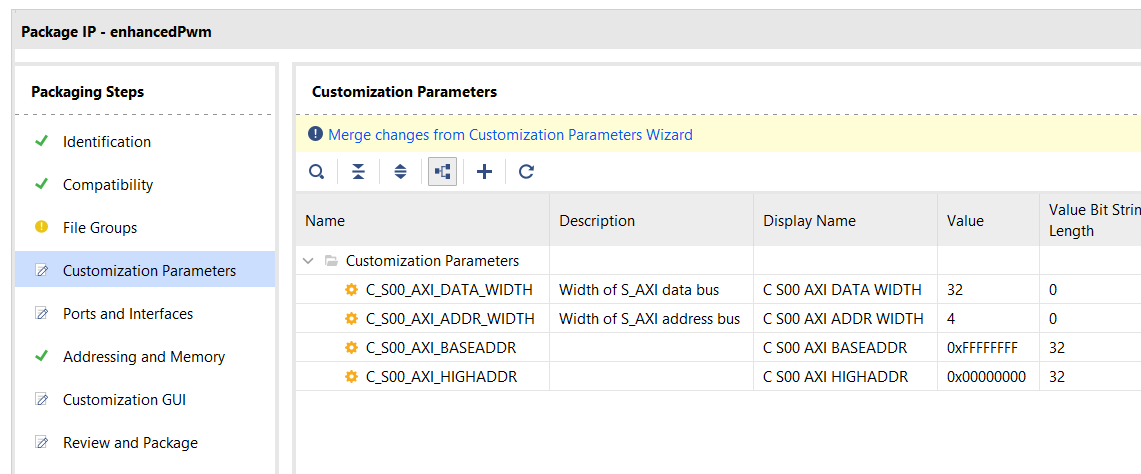
**Identification** – Leave alone

**Compatibility** – Leave along

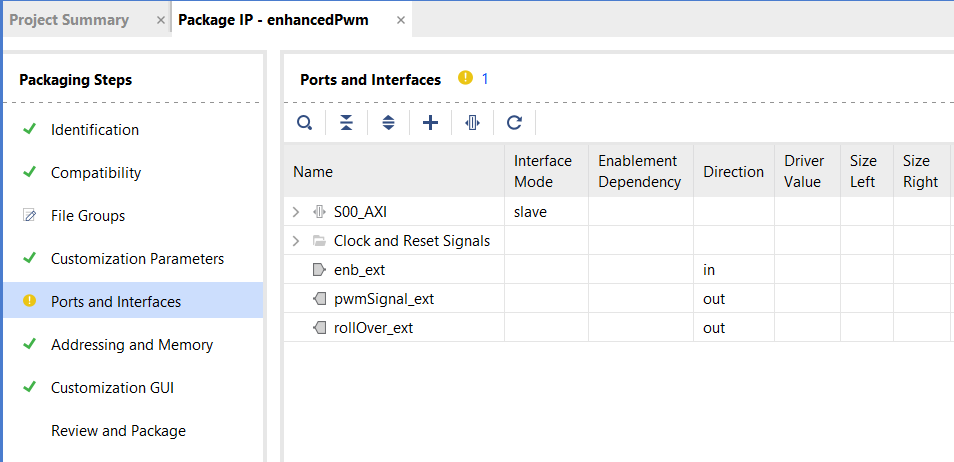
**File Groups** – Click on “Merge changes from File Groups Wizard” This step saves the state of the project. You MUST do this before quitting otherwise you will loose all your work. Ask me how I know :/



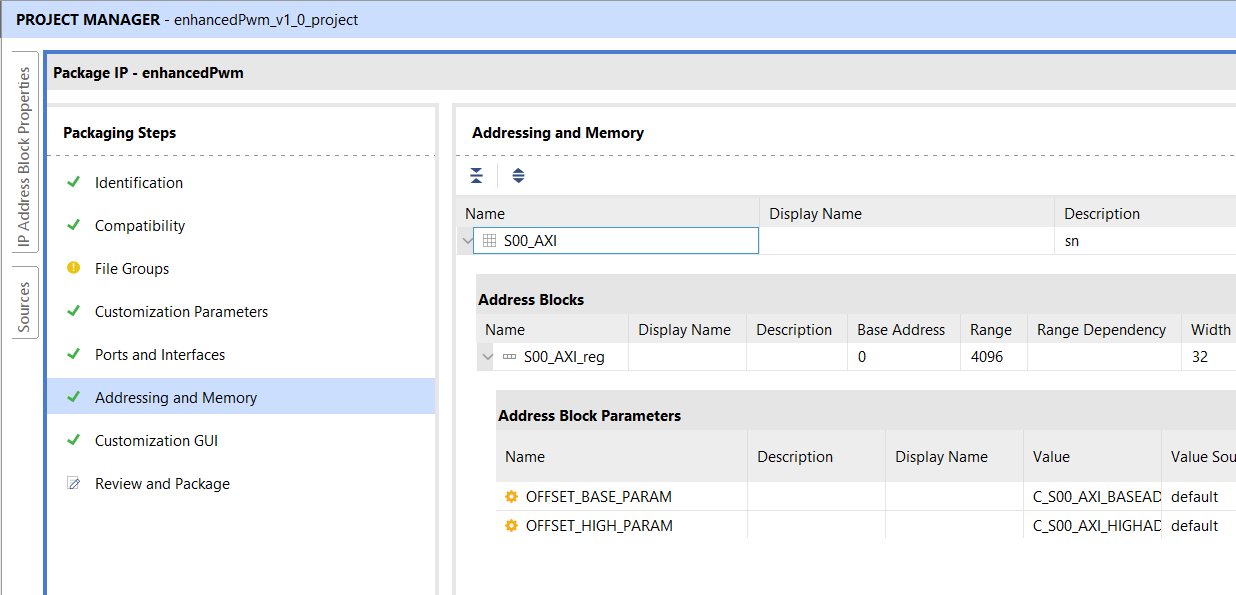
**Customization Parameters** – Click on “Merge changes from Customization Parameters Wizard”



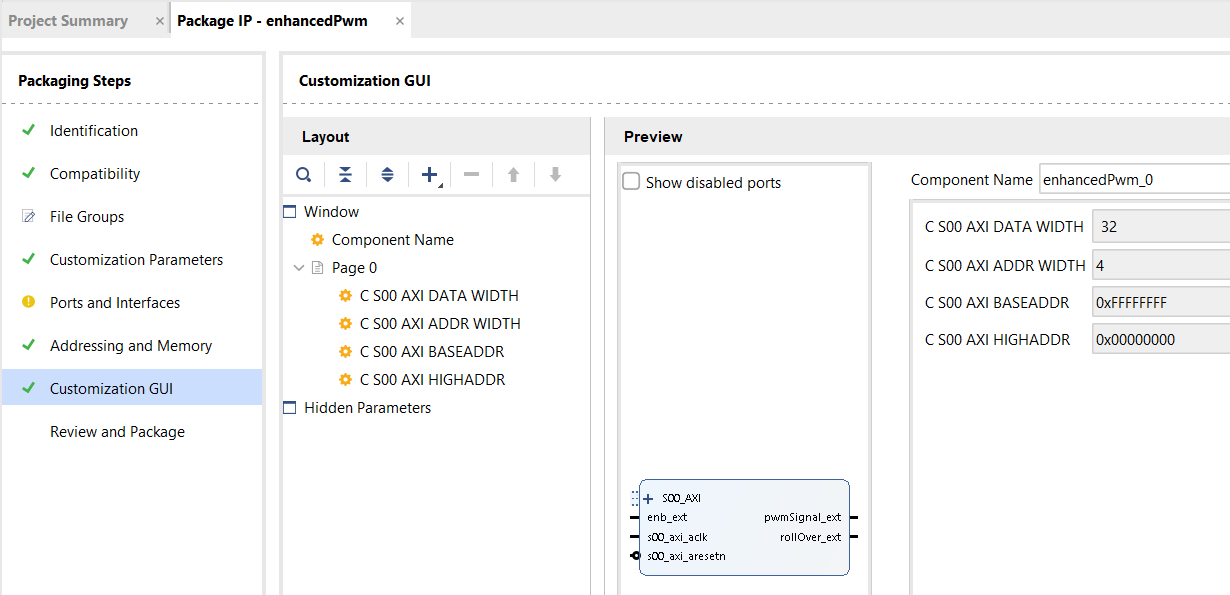
**Ports and Interfaces** – Notice that your enb\_ext, pwmSignal\_ext and rollover\_ext signals have the correct direction and are part of the IP interface.



**Addressing and Memory** – Leave defaults.



**Customizing GUI** – Leave defaults. Notice that your enb\_ext, pwmSignal\_ext and rollover\_ext are part of the block diagram. You will need to connect them in a coming step.



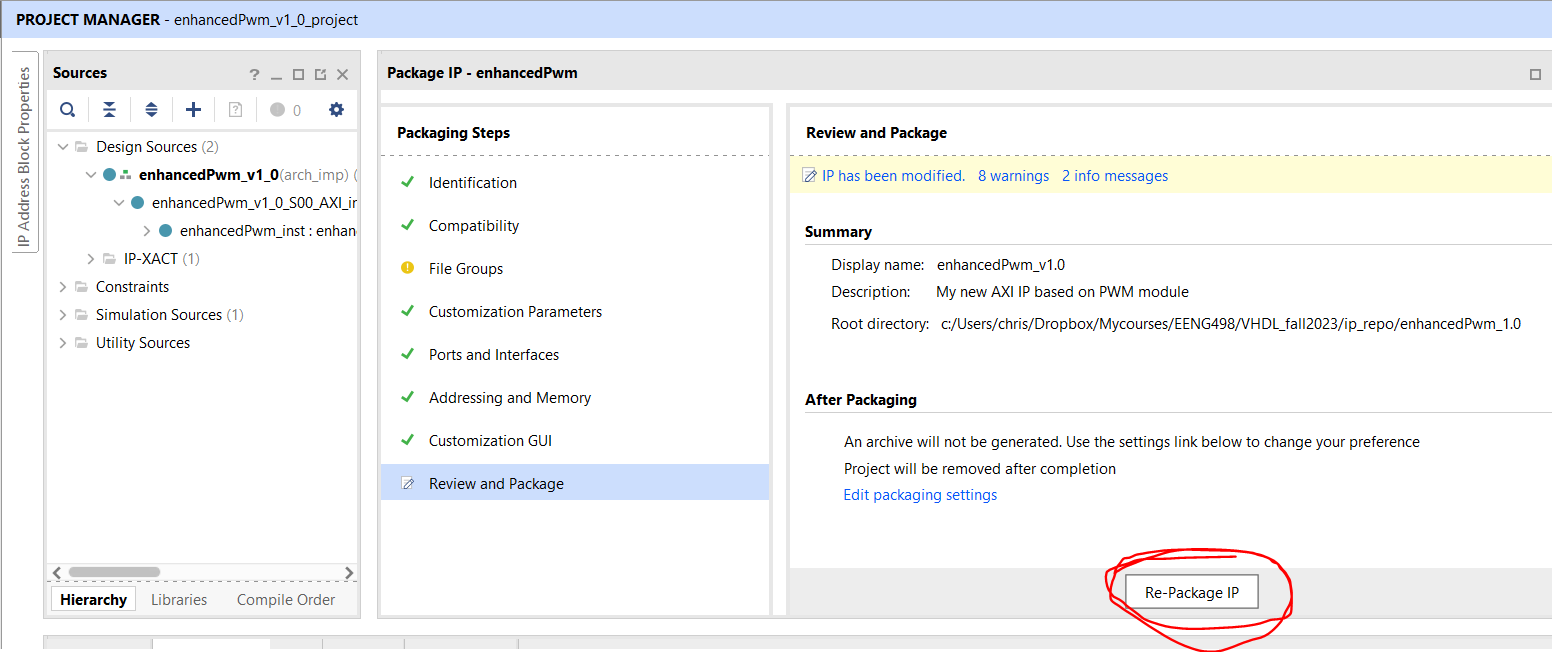
Before completing the final step of the IP process, it is much easier if you clear up any syntax errors that you may have created. To do this, I like to run a simulation with the intention of weeding out errors, not to actually perform a simulation. The error will most likely be contained in a file in the source project that you created to generate this IP. In my case, this was:

lab04\_oscopeHardware\lab04\_oscopeHardware.tmp\acquirewithhdmi\_v1\_0\_project\acquireWithHDMI\_v1\_0\_project.sim\sim\_1\behav\xsim

After weeding out the easy errors from the simulation. You might as well try to Generate Bitstream to weed out all elaboration errors from vector size mis-matches.

When you get all the errors corrected, you should be safe to complete the process to generate your IP.

**Review and Package** – Re-Package IP

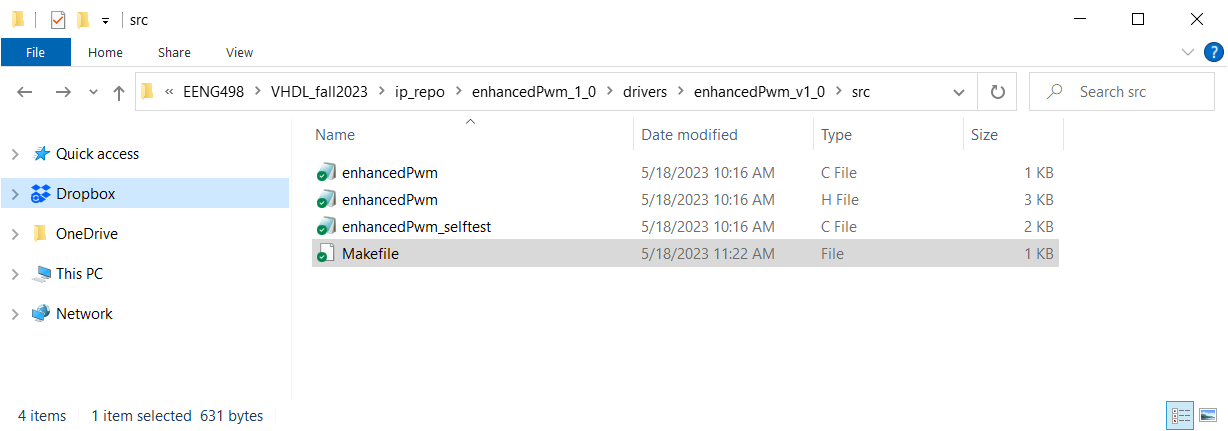


Click the Re-Package IP button and agree to close the project. You can close this project when it’s done.

One final step that will solve a lot of problems later needs to be done. Use the windows explorer to navigate to your ip\_repo folder. From this folder navigate to:

...\enhancedPwm\_1\_0\drivers\enhancedPwm\_v1\_0\src

In this directory you should find the following files:



Double click on Makefile and open it in Notepad. Replace the contents at the end of Makefile with the text shown in the Modify column below. Save and exit.

|  |  |
| --- | --- |
| Original | Modify |
| INCLUDEFILES=\*.h LIBSOURCES=\*.c OUTS = \*.o  libs: echo "Compiling myip" $(COMPILER) $(COMPILER\_FLAGS) $(EXTRA\_COMPILER\_FLAGS) $(INCLUDES) $(LIBSOURCES) $(ARCHIVER) -r ${RELEASEDIR}/${LIB} $(OUTS) make clean  include: ${CP} $(INCLUDEFILES) $(INCLUDEDIR)  clean: rm -rf ${OUTS} | INCLUDEFILES=$(wildcard \*.h)  LIBSOURCES=$(wildcard \*.c)  OUTS = $(wildcard \*.o)  **OBJECTS = $(addsuffix .o, $(basename $(wildcard \*.c)))** ASSEMBLY\_OBJECTS = $(addsuffix .o, $(basename $(wildcard \*.S)))  libs:  echo "Compiling myip"  $(COMPILER) $(COMPILER\_FLAGS) $(EXTRA\_COMPILER\_FLAGS) $(INCLUDES) $(LIBSOURCES)  $(ARCHIVER) -r ${RELEASEDIR}/${LIB} **${OBJECTS} ${ASSEMBLY\_OBJECTS}**  make clean  include:  ${CP} $(INCLUDEFILES) $(INCLUDEDIR)  clean:  rm -rf **${OBJECTS} ${ASSEMBLY\_OBJECTS}** |

It is very important that you have tabs on the statements following “libs:”, “include:”, and “clean:”. Ask me why :/ You are now done creating your custom IP. It’s time to integrate this IP with a Zynq processor.