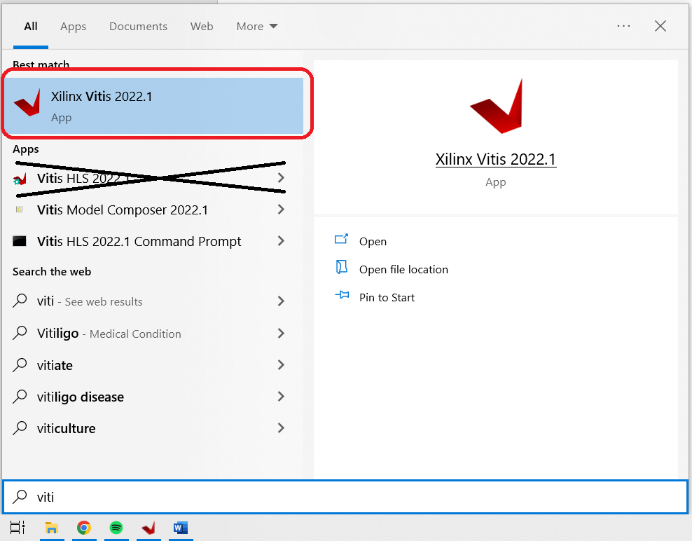
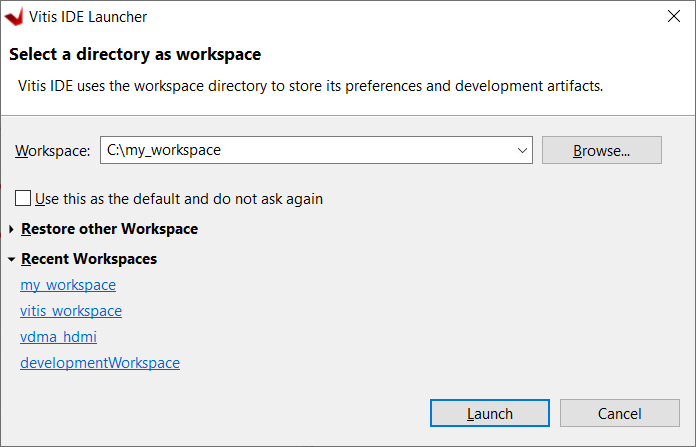
Once you attach an ARM Cortex to your custom IP, it’s time to program the ARM to interface to your IP. The slv\_regX that you connected to your custom IP are memory mapped locations. Thus writing to the correct address, will send bits to the input ports of your custom IP entity and reading from another address will read the output from some other port on your custom IP.

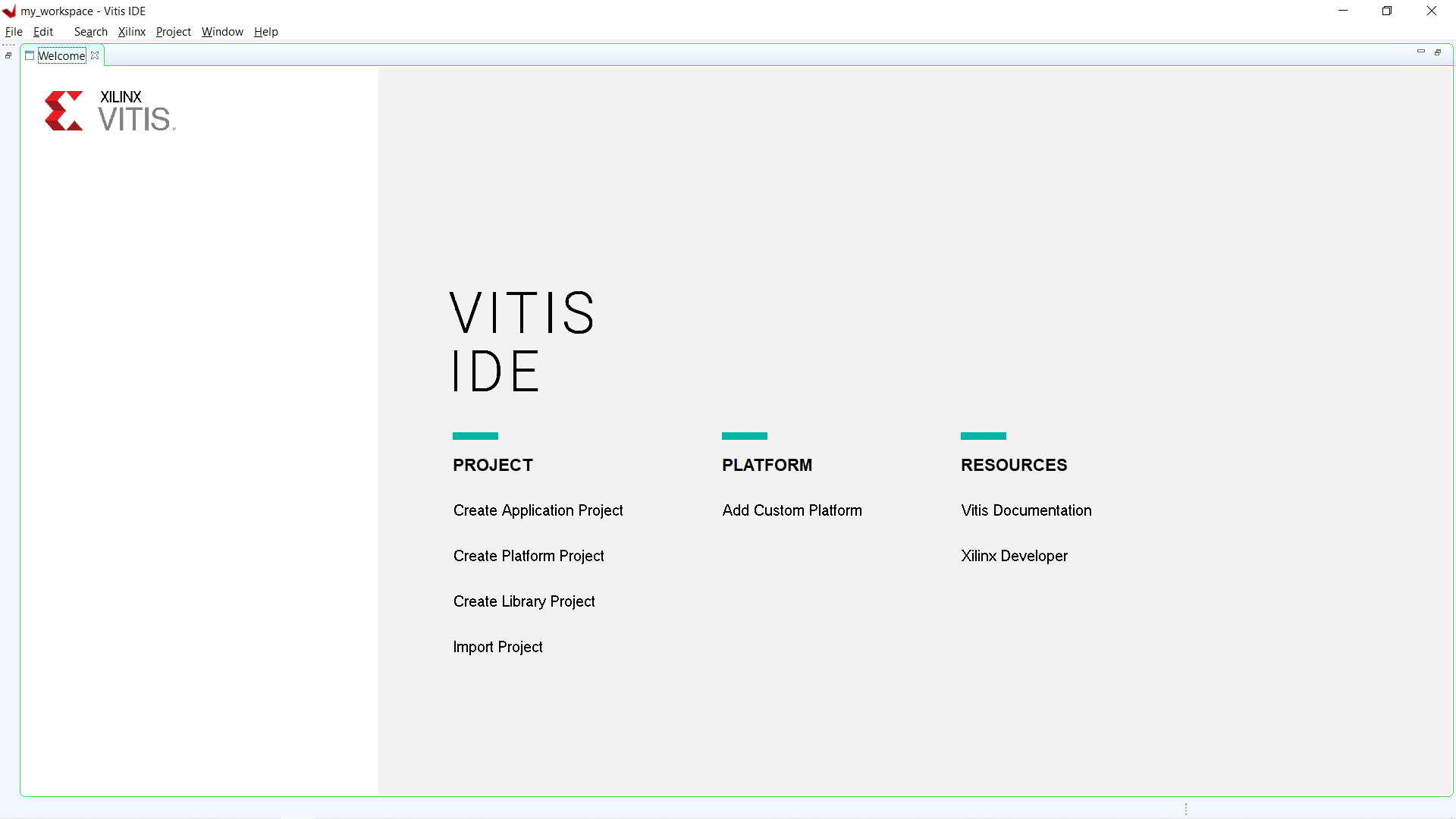
Let’s start at the beginning. Launch Xilinx Vitis. Do not launch Vitis HLS



Vitis does not like long paths nor does it like spaces in the path names. I parked my workspace right off root at c:\my\_workspace

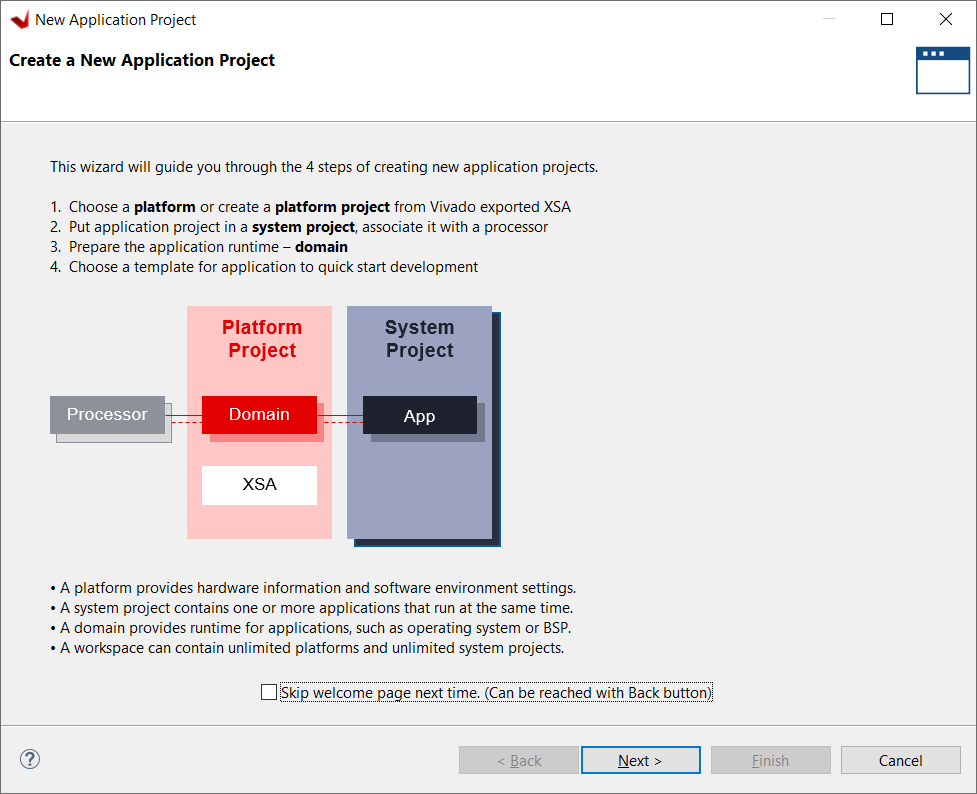


At the Welcome tab, select Create Application Project



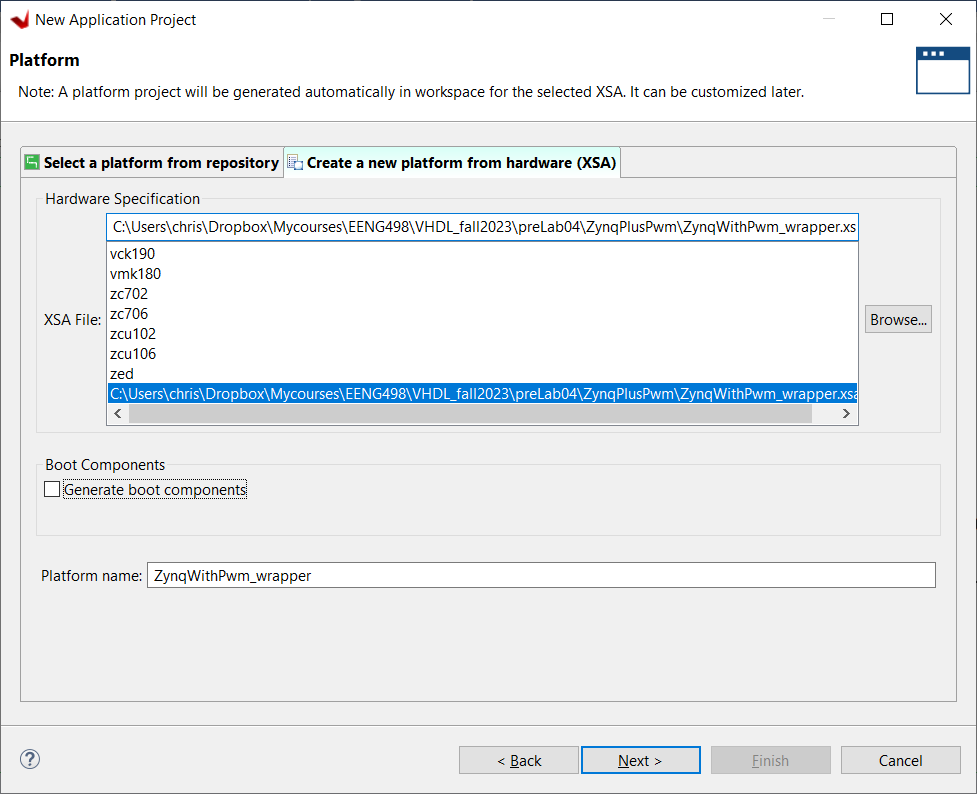
This starts the New Application Project Wizard. The Wizard will walk you through different pages, let’s go through them together.

**Create a New Application Project:** Click Next.



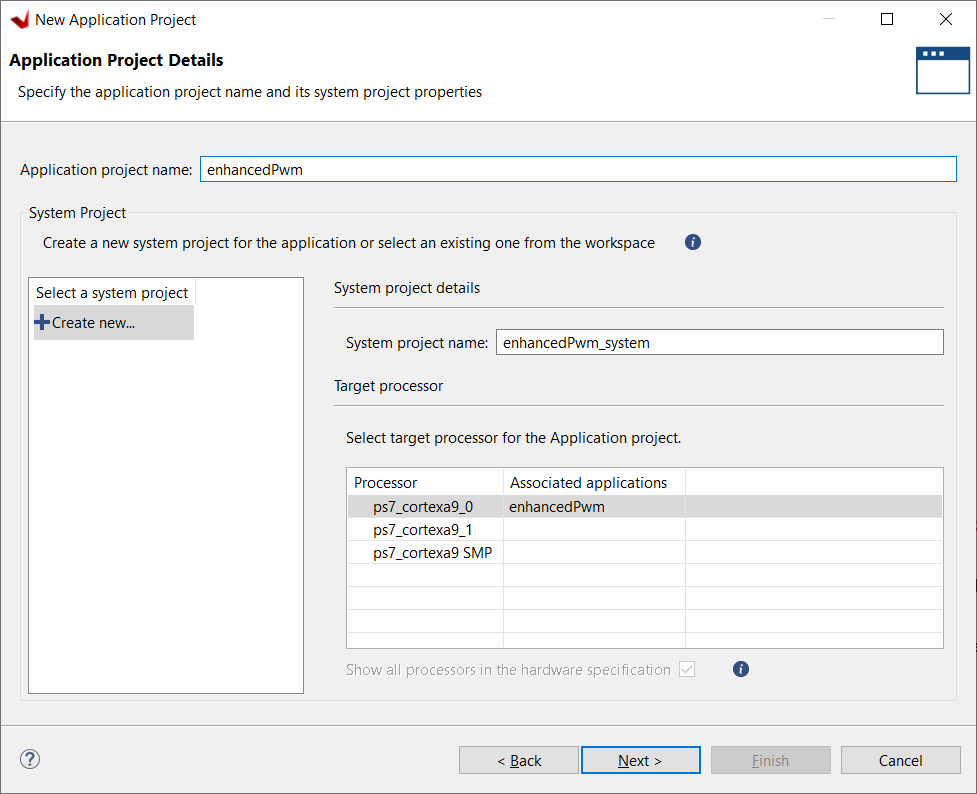
**Platform:** Go to the Create a new platform from hardware (XSA) tab. Click Browse…, in the explorer pop-up navigate to the wrapper file you created in Vivado. In my case, this was at: C:\Users\...\Mycourses\EENG498\VHDL\_fall2023\preLab04\ZynqPlusPwm\ZynqWithPwm\_wrapper.xsa

And the then click Open. You should see your wrapper file highlighted at the bottom of the XSA list.

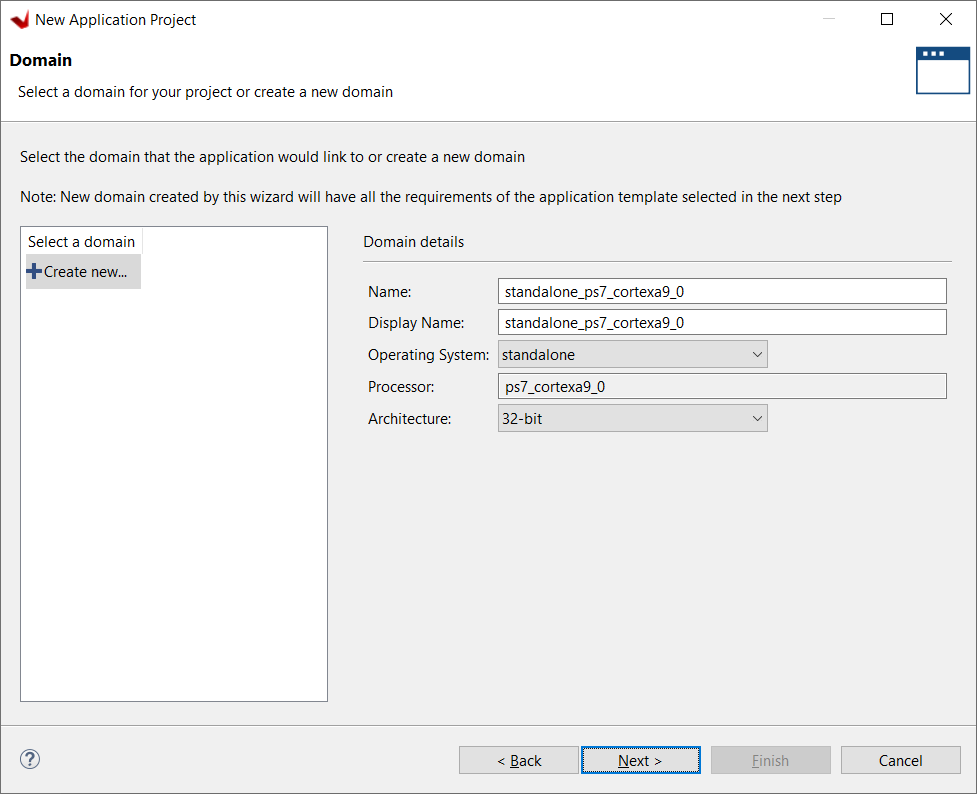


Unselect “Generate boot components” Click Next.

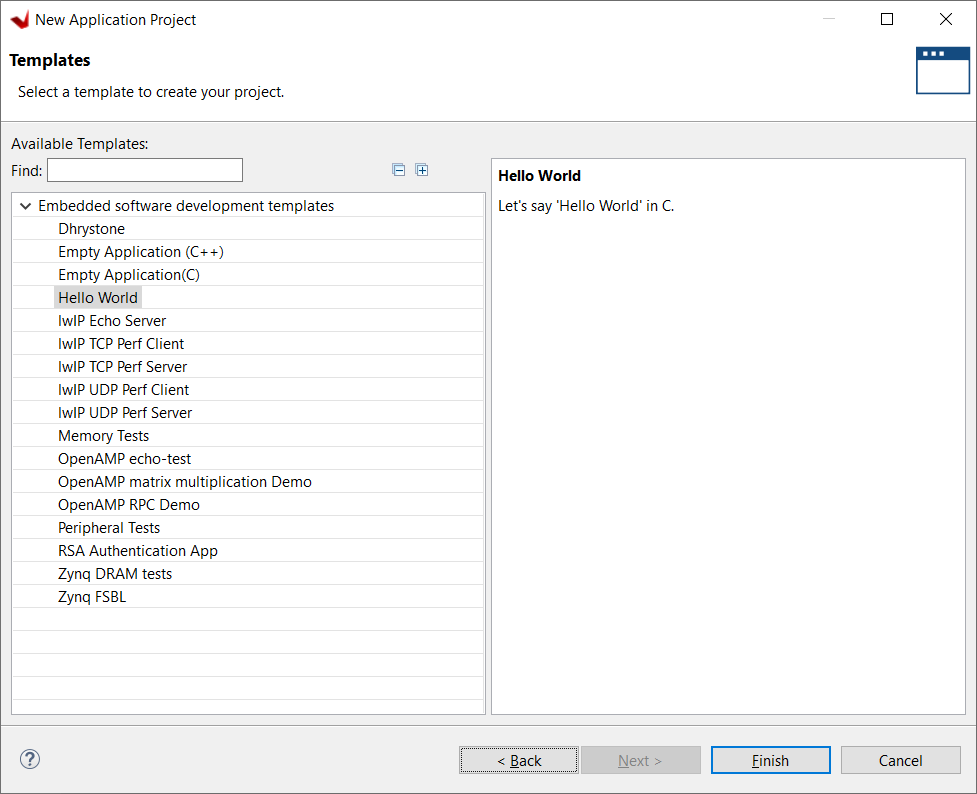
**Application Project Details:** Provide a name in the Application project name field. Click Next.



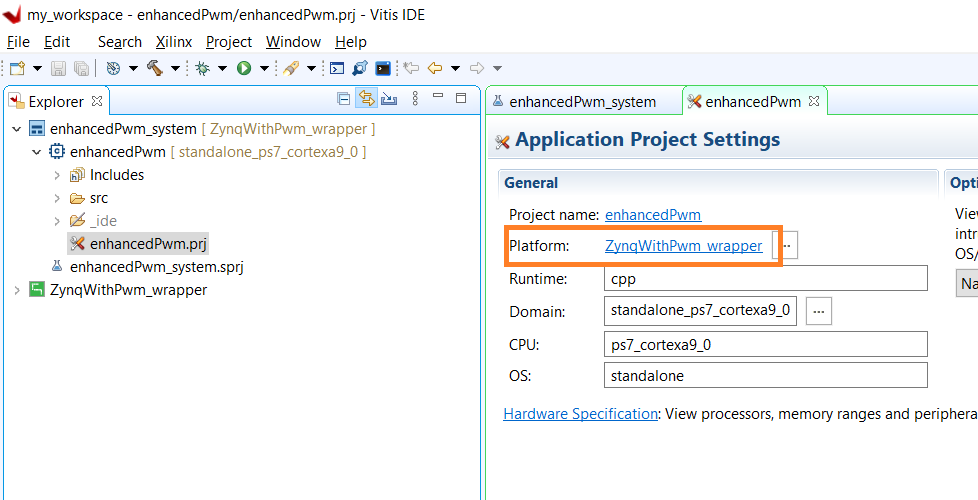
**Domain:** Leave the defaults alone. Click Next.



**Templates:** Leave the defaults alone. Click Finish.



You are at the main development window. Before you proceed, take note of the Platform name, this is circled in orange.



Before you proceed, you will need to do some low-level tinkering with three makefiles used by Vitis to compile your build. The location of these makefiles relies on three pieces of information that will be unique to your development environment. Please replace my locations and names with those used in your environment. I’ve color coded the relevant information to make this easier.

* My workspace was placed in: c:\my\_workspace
* My platform name is: ZynqWithPwm\_wrapper
* My custom IP is called: enhancedPwm\_ip\_v1\_0

All three of the following Makefiles are identical and all three need updated in exactly the same way.

C:\my\_workspace\ZynqWithPwm\_wrapper\hw\drivers\enhancedPwm\_ip\_v1\_0\src\Makefile

C:\my\_workspace\ZynqWithPwm\_wrapper\ps7\_cortexa9\_0\standalone\_ps7\_cortexa9\_0\bsp\ps7\_cortexa9\_0\libsrc\enhancedPwm\_ip\_v1\_0\src\Makefile

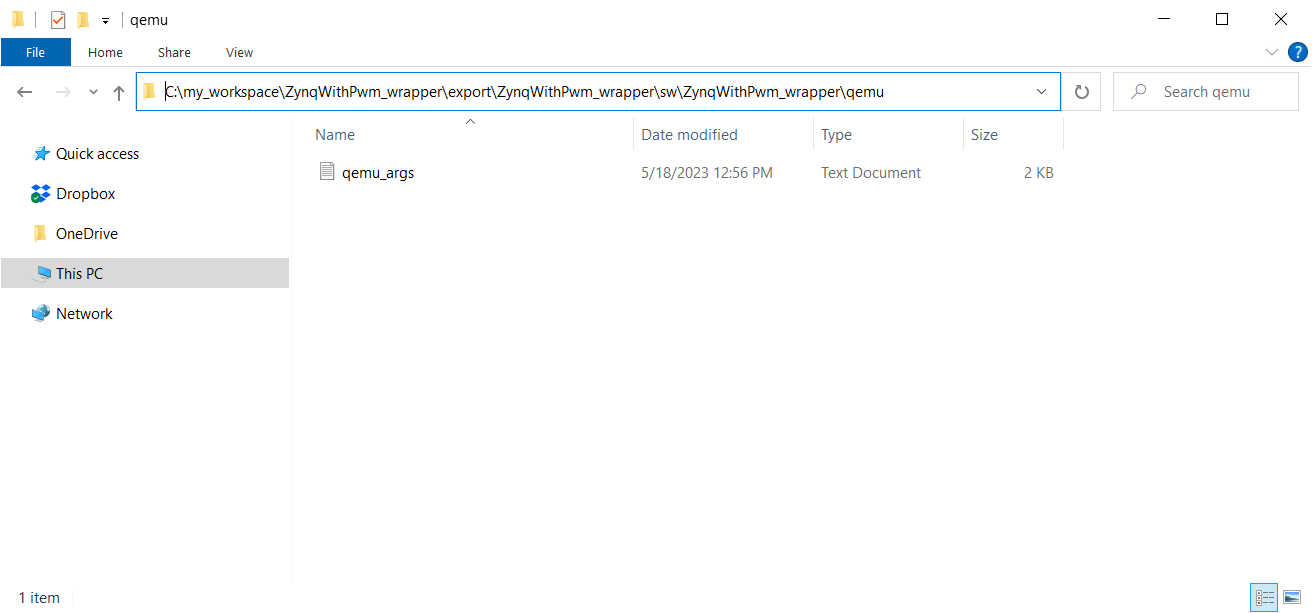
C:\my\_workspace\ZynqWithPwm\_wrapper\zynq\_fsbl\zynq\_fsbl\_bsp\ps7\_cortexa9\_0\libsrc\enhancedPwm\_ip\_v1\_0\src\makefile

|  |  |
| --- | --- |
| Original | Modified |
|  |  |
| INCLUDEFILES=\*.h  LIBSOURCES=\*.c  OUTS =\*.o | INCLUDEFILES=$(wildcard \*.h)  LIBSOURCES=$(wildcard \*.c)  OUTS = $(wildcard \*.o) |
|  | INCLUDEFILES=\*.h  LIBSOURCES=\*.c  OUTS = \*.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} |

Sometimes, the Build process complains that it is missing a file/folder called qemu at:

C:\my\_workspace\ZynqWithPwm\_wrapper\export\ZynqWithPwm\_wrapper\sw\ZynqWithPwm\_wrapper

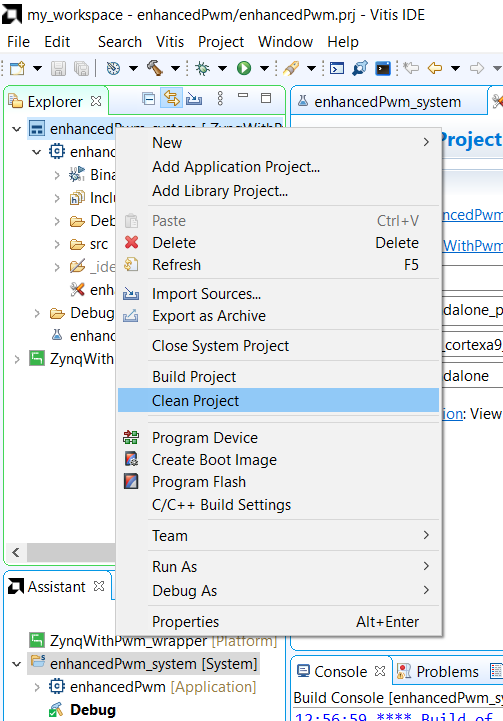
If this is the case, go to this folder create a folder called qemu and an empty text file called qemu\_args.txt



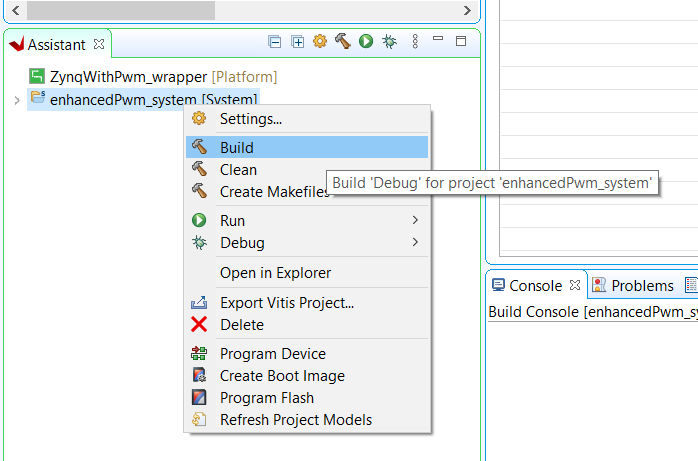
You can also look at the following and try this modification of your makefiles:

* <https://support.xilinx.com/s/article/75527?language=en_US&_ga=2.70550867.234489591.1684351575-1696831773.1681097104>
* https://blog.csdn.net/yihuajack/article/details/120881411

If your build fails for any reason, I would strongly advise cleaning the project so that the build starts fresh with the changes that you have made to the makefile and directory structure.

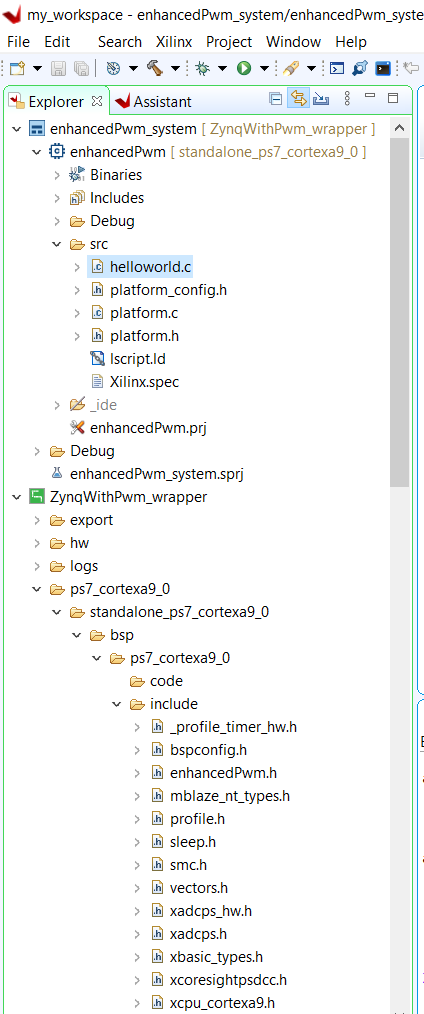


In the Assistant tab, right mouse click on your System name and select Build.

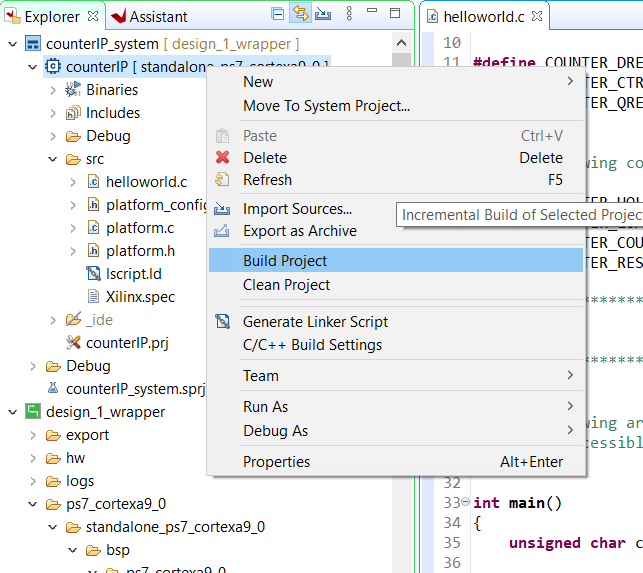


This will create all the source code that you need to interface to your custom IP through the AXI bus. You only need to build the System when you make changes to the wrapper and import it. This is generally because you found a flaw in your custom IP.

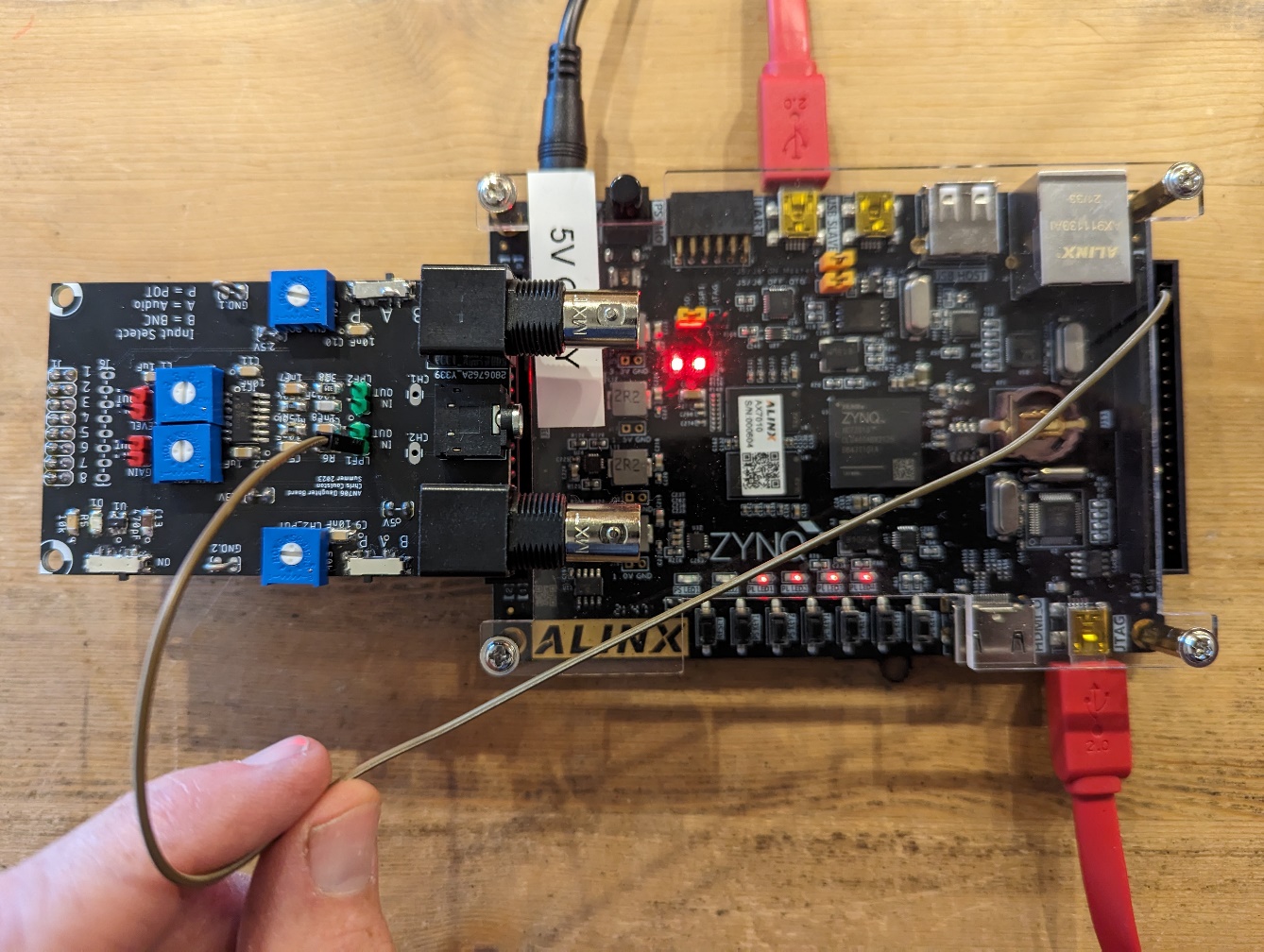
I like to stack the Assistant and Explorer tabs next to one another. The following two locations will prove to be very useful in your development.



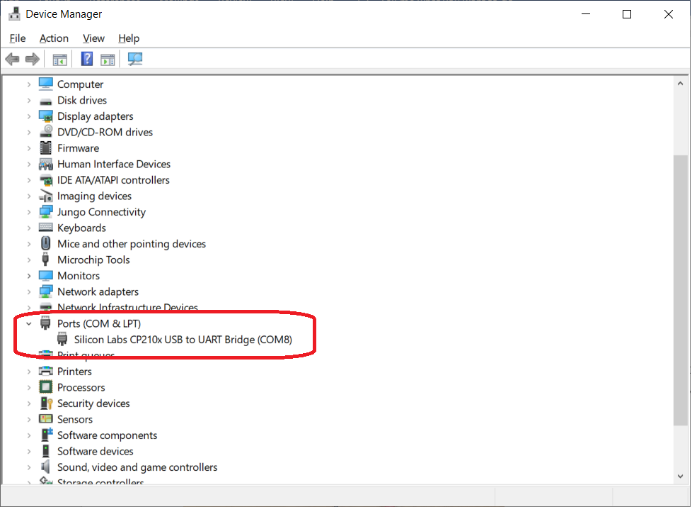
Double click the helloworld.c file and replace the contents with that provided on the class web page. Now we will compile this code by right mouse clicking on the enhancedPwm and selecting Build Project



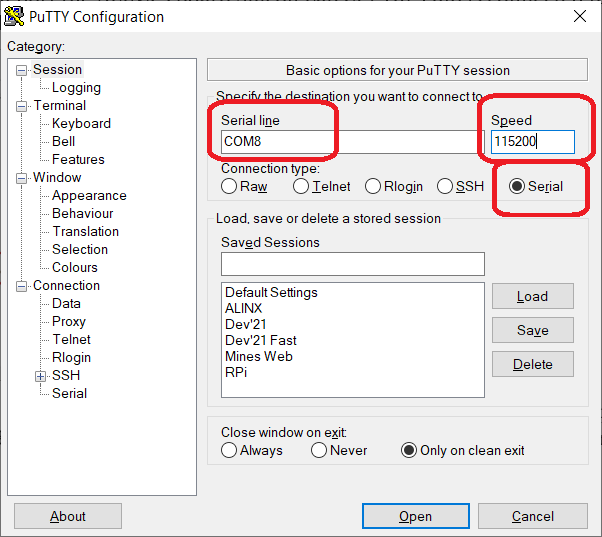
Connect power to the ALINX board as well as the JTAG and UART cables to your PC. Run a jumper from pin 3 of the J11 header to the LPF input.



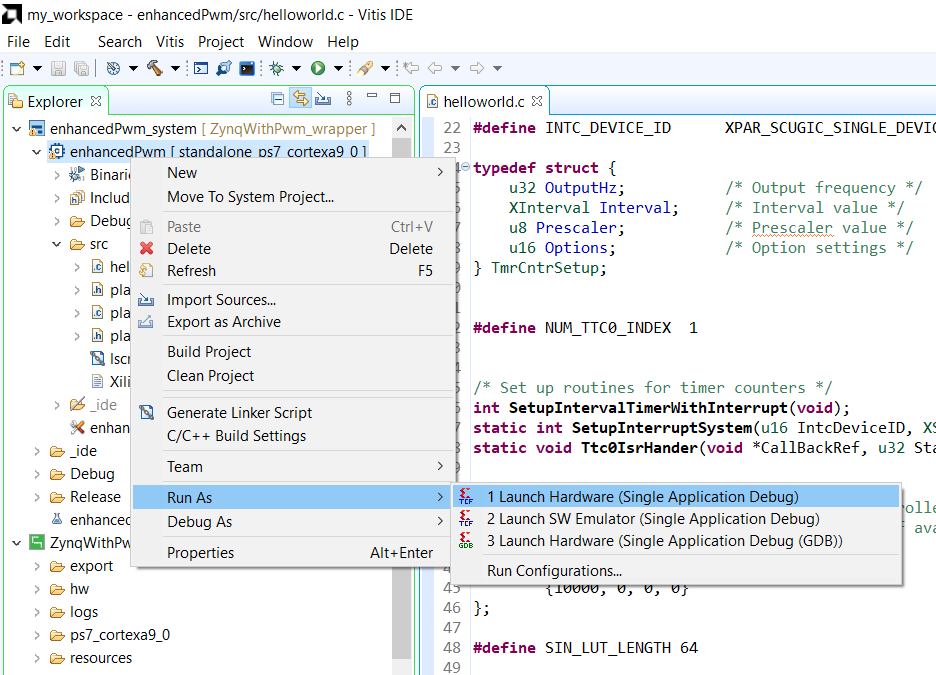
Determine which virtual COM port the ALINX’s board Silicon Lab CP2102 chip is connected to. On a PC, do this by launching the Device Manager.



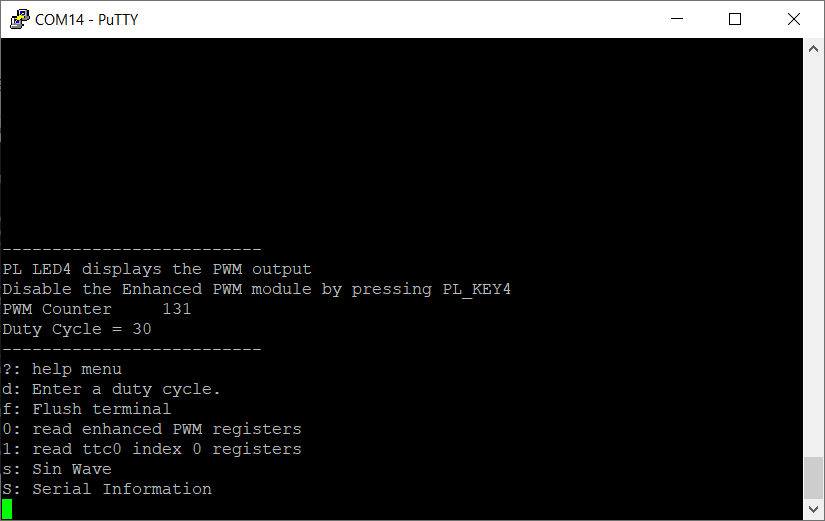
Scroll down the device manager to Ports, expand and look for the Silicon Labs chip. In my case, it is attached to COM8. Now launch Putty, connect to COM8 at 115200 Baud.



Then download the hardware and software combination to the ALINX board by right mouse clicking on the enhancedPwm [standalone], selecting Run As, then selecting 1 Launch Hardware (Single Application Debug).



You can now interact with your custom IP through the Putty interface.



If/when you make changes to your hardware in Vivado, you will need to communicate these changed to Vitis. To do this, just right mouse click on the wrapper and select Update Hardware Specification. In the pop-up, navigate to the updated wrapper file and open it.



Next you should use Clean Project option to remove any legacy files that are no longer needed. Then you will have to re-editing of the 3 makefiles and then build the system.

This process can be a real pain, so be patience.