MICROCART 2014-2015

Xilinx Tools (XPS, XSDK, and XISE) Setup and Walkthrough

Author: Paul Gerver, Last updated: Feb 2015

Table of Contents

1. Introduction

Welcome to the ultimate Xilinx Tools How-To guide! This guide should provide you with all the necessary knowledge and step-by-step instructions to get you making hardware configurations and programs with the different Xilinx tools.

There are three main tools that we will be learning: the Xilinx Platform Studio (XPS), Integrated Software Environment (ISE), and Software Development Kit (SDK). Students who have taken or are taking CprE 488: Embedded System Design may be familiar with these tools and should be a first stop for questions when working with the tools. Additionally, CprE 488 does has some helpful lab instructions, called MPs, and can also provide some helpful advice for working with these tools (http://class.ece.iastate.edu/cpre488/schedule.asp)

Because of MicroCART's use of the Diligent ZyBo board[1], the following walkthrough will be centered on this development board, and different steps may be taken when working with a different board. The author will try his best to indicate when instances like these may arise.

2. Setup

Setting up access to the Xilinx tools is fairly straight forward given the machine the user is on. This guide will cover three types of machines: Coover 3050-11 and -12 computers (highly recommended), ISU's Remote Linux Servers, and a user's own PC

2.1. Coover 3050-11 and -12

Two machines in the Distributed Sensing and Decision Making Lab (Coover 3050) come with the tools already installed. However, the following steps need to be taken in order to launch the program

1. In a terminal, enter `source /opt/Xilinx/ISE/14.7/settings64.sh`
2. OR `source Xilinx_Tools/setup_scripts/co3050/setup.sh`

2.2. ISU Remote Linux Servers (linux-X, research-x.ece.iastate.edu)

3. `source Xilinx_Tools/setup_scripts/remote_servers/setup.sh`
4. That's it! (Note: these servers are not good for programming the Zybo board when it comes time to launch a program on the board)

2.3. User PC

Some users may opt to download the Xilinx tools on their own PCs for development, but this is not recommended.

5. Download the ISE Design Suite here (~6GB)

[1] Please see the Diligent website for exact features and documents: http://digilentinc.com/Products/Detail.cfm?Prod=ZYBO

[Back to Top]

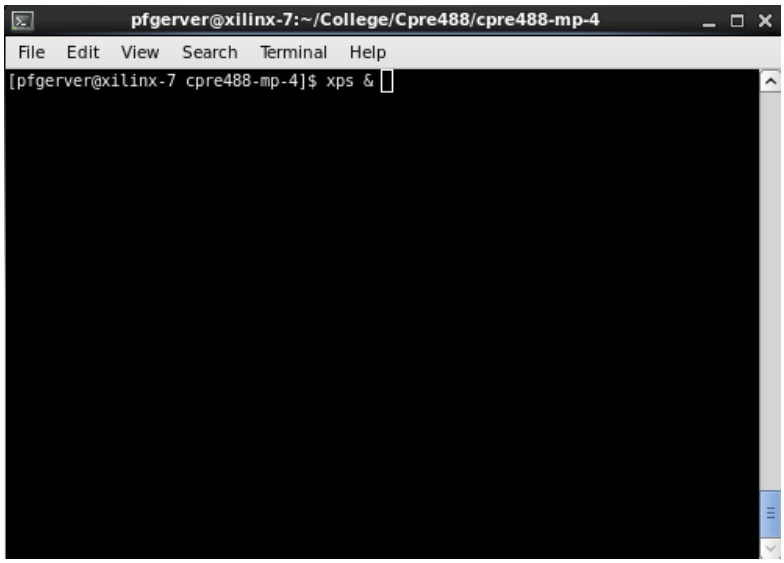
6. Ensure to install the ISE, XPS, and XSDK (The embedded package, I believe)

3. Xilinx Platform Studio (XPS)

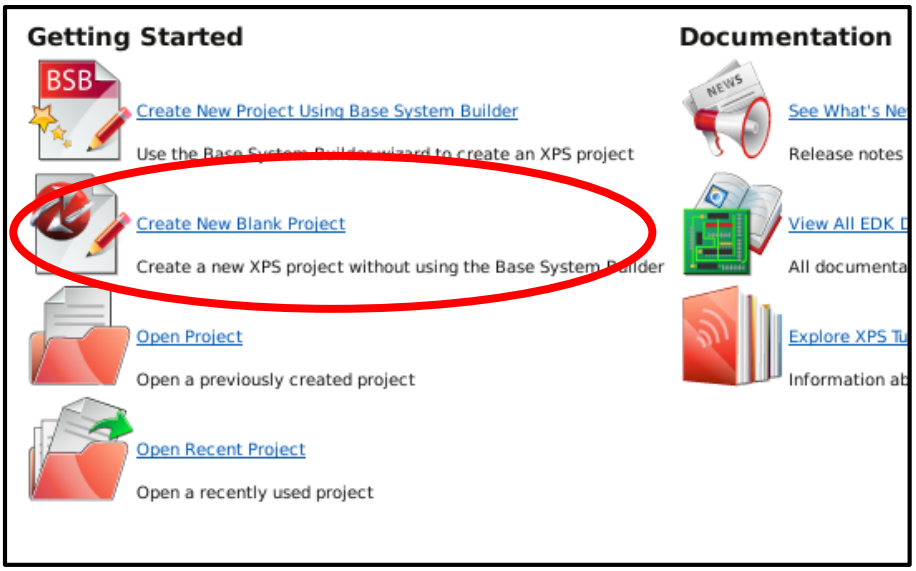
XPS is the base program for creating a hardware description file (a .bit file) that will be loaded onto the Zybo board since it contains an ARM Cortex-A9 processor and a Zynq FPGA. Without a bit file, any program that is launched on the board may not be able to communicate with all the I/O and cool peripherals of the board. We'll first learn to start the XPS program, create a project targeted toward the Zybo (since this program can create bit files for more than just our board), and build our first bit file. Then, we will walkthrough setting up additional configurations that MicroCART uses

3.1. Creating a new XPS project

1. Once our environment and terminal have been setup, enter: `xps &`



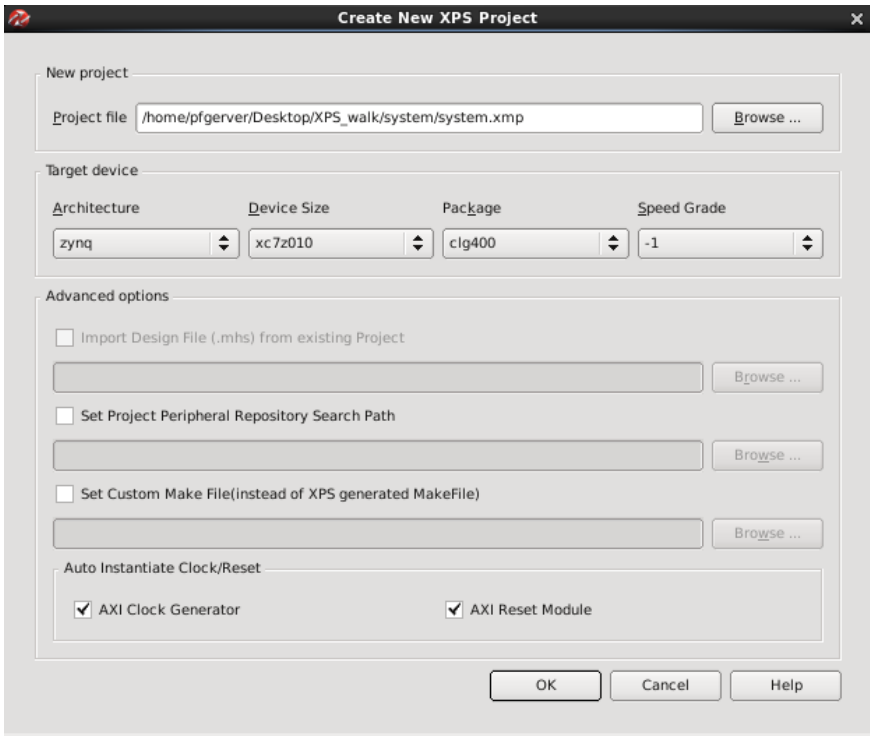
2. You should now be greeted with the opening window. Next click "Create New Blank Project"



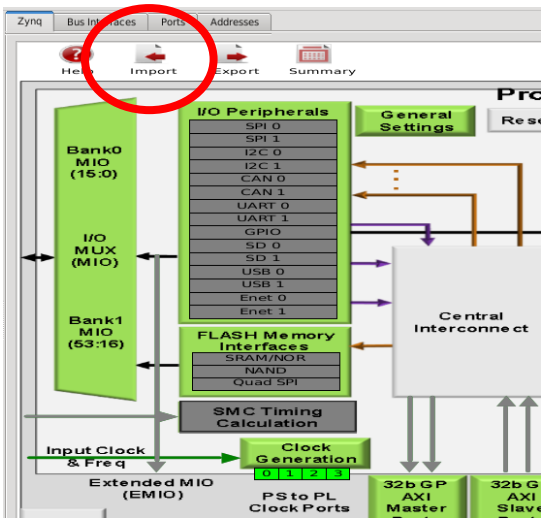
[Back to Top]

3. The “Create New XPS Project” window should appear. Click Browse … to select a location to save the project. (It is highly recommended to save the project in a “system” folder)

4. Select the Target Device fields to look exactly like they do below: Architecture = Zynq, Device Size = xc7z010, Package = clg400, Speed Grade = -1. Click OK when done.

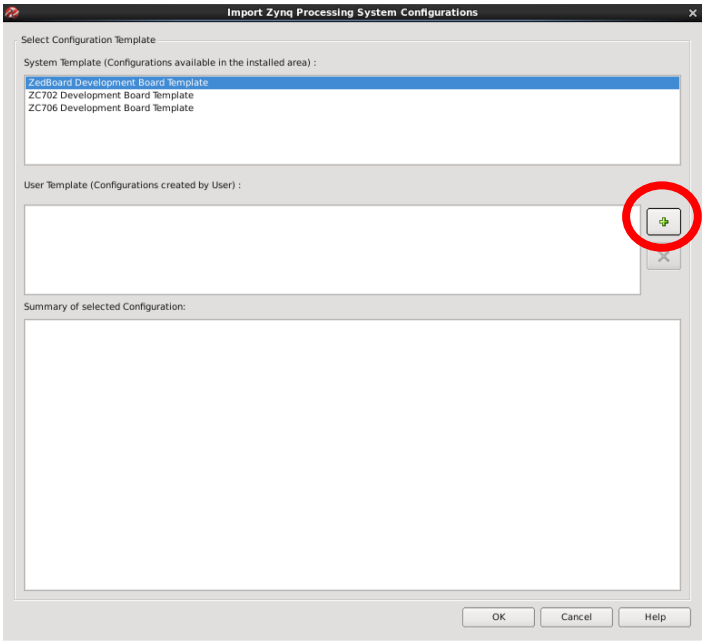


5. The Zynq System view should appear and look like the image below. Click Import to start importing the base definition for the Zybo.

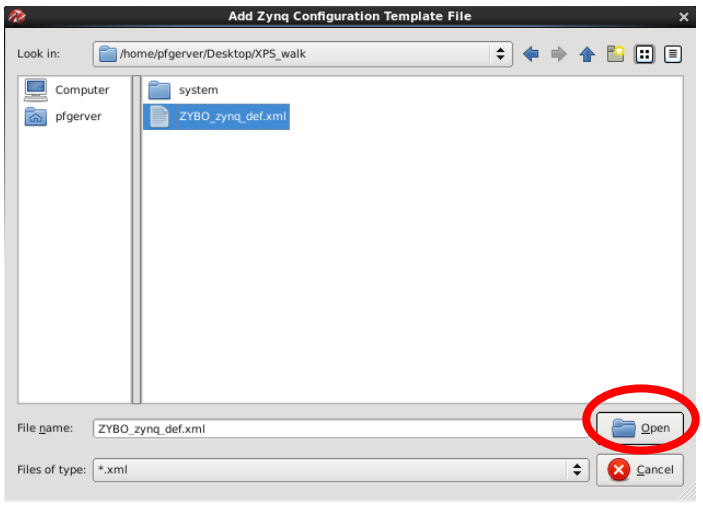


[Back to Top]

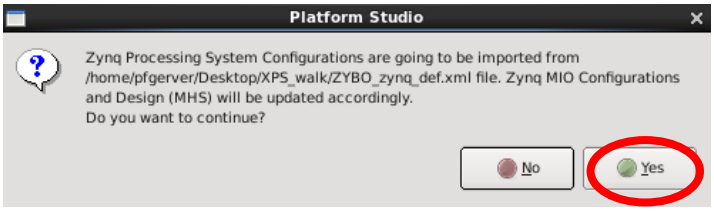
6. Click the + button to add a new definition file and browse for the ZYBO_zynq_def.xml which can be found in "Xilinx_Tools/XPS_files/"



7. Once selected, click Open and click OK to start the import

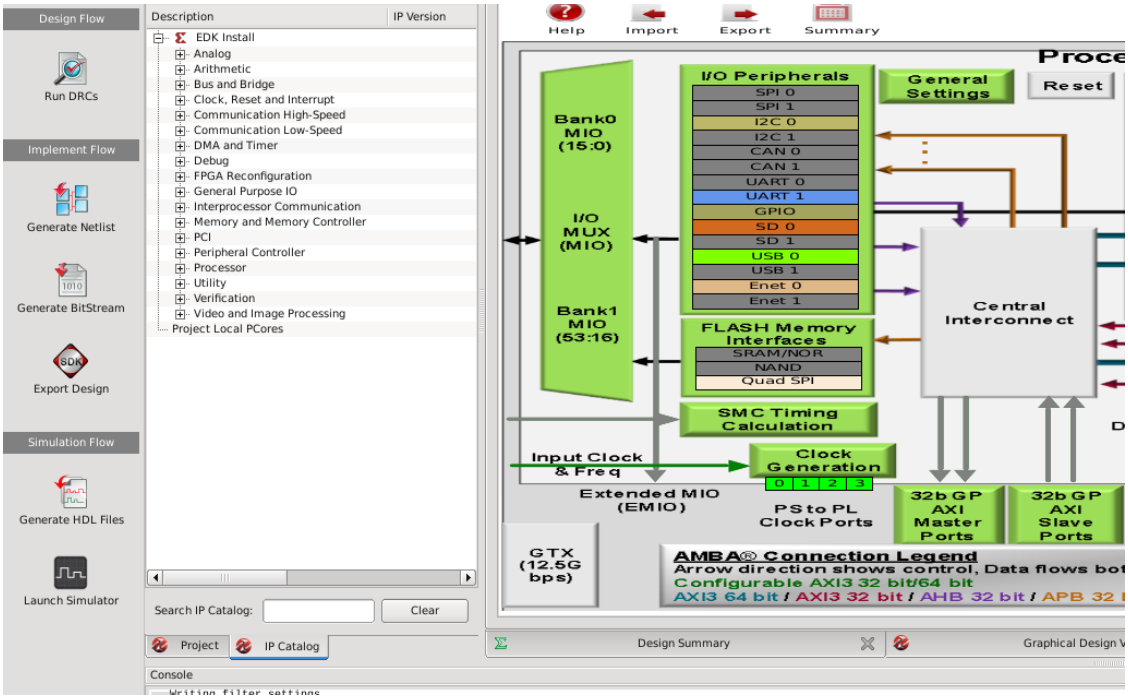


8. XPS will prompt you asking to update the MHS file, click Yes.



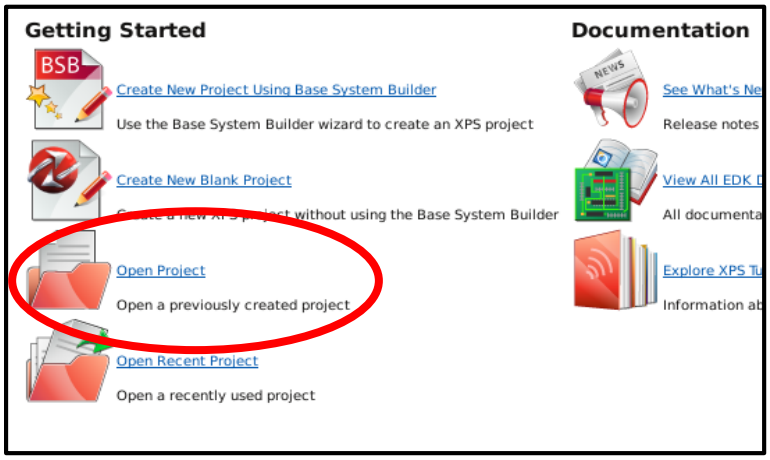
[Back to Top]

9. The default I/O controllers for the SD card, UART1 and other things will be selected and look like the following screen below.



3.2. Opening an XPS Project

1. An XPS project can be opened in two ways:
   a. From the Getting Started window, click Open Project
   b. If a project is already open, click File -> Open Project



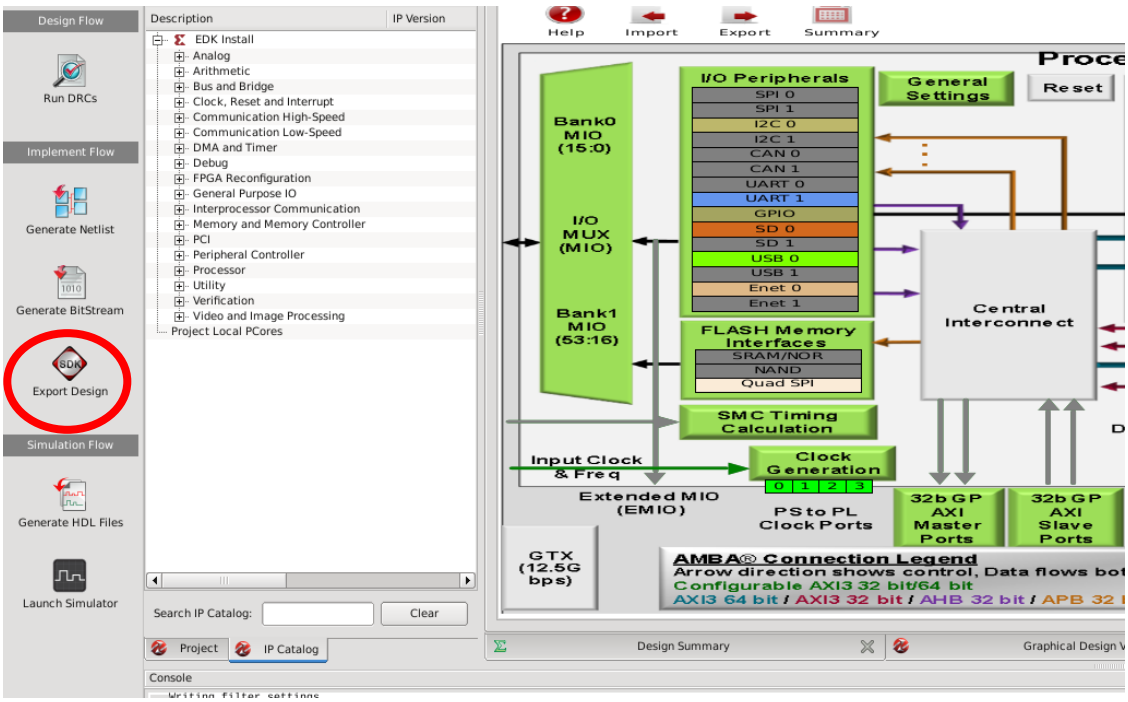
2. Navigate to the project .xmp file (i.e. task/zybo_sensor_board/system/system.xmp) and click Open

[Back to Top]

3.3. Synthesizing an XPS project

DECISION: If users want to add additional features to use Bluetooth, I2C for the sensor board, or other configurations, please skip to the corresponding feature sections below before conducting the export process.

1. To create a bitfile, click Export Design (indicated by the red circle)



2. A Export to SDK / Launch SDK prompt will appear. Ensure XSDK is closed, if it was open, and click Export & Launch SDK (Skip to XSDK)

FORWARNING: The exporting process can take several minutes to complete or over an hour depending on machine load and how much logic needs to programmed/routed.

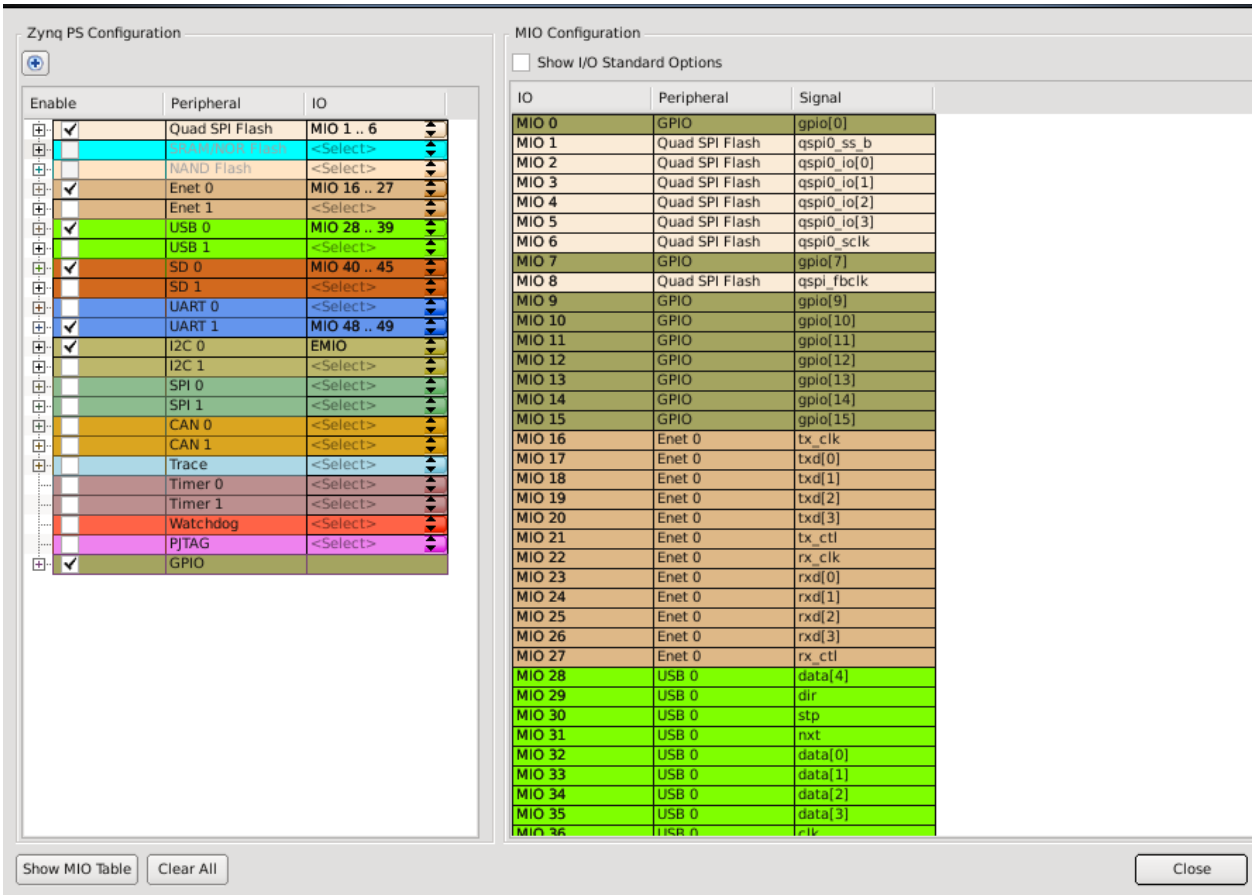
[Back to Top]

3.4. Enabling Bluetooth (UART0) / Editing MIO settings in XPS

By default, UART1 is automatically configured for us when we imported the base Zybo definition file. UART1 is used on the board for receiving JTAG data and uses it as stdout and stdin for the board to communicate with the computer, when running a program. We will be enabling another UART, UART0, which will routed to a PMOD, so a program running on the board can send data via Bluetooth to the paired machine.

This walkthrough will setup UART0 communication routing for a BT2PMOD chip through the Zybo's PMOD-B port.

1. In XPS System View, click on the green I/O Peripherals box to bring up MIO Configuration box.

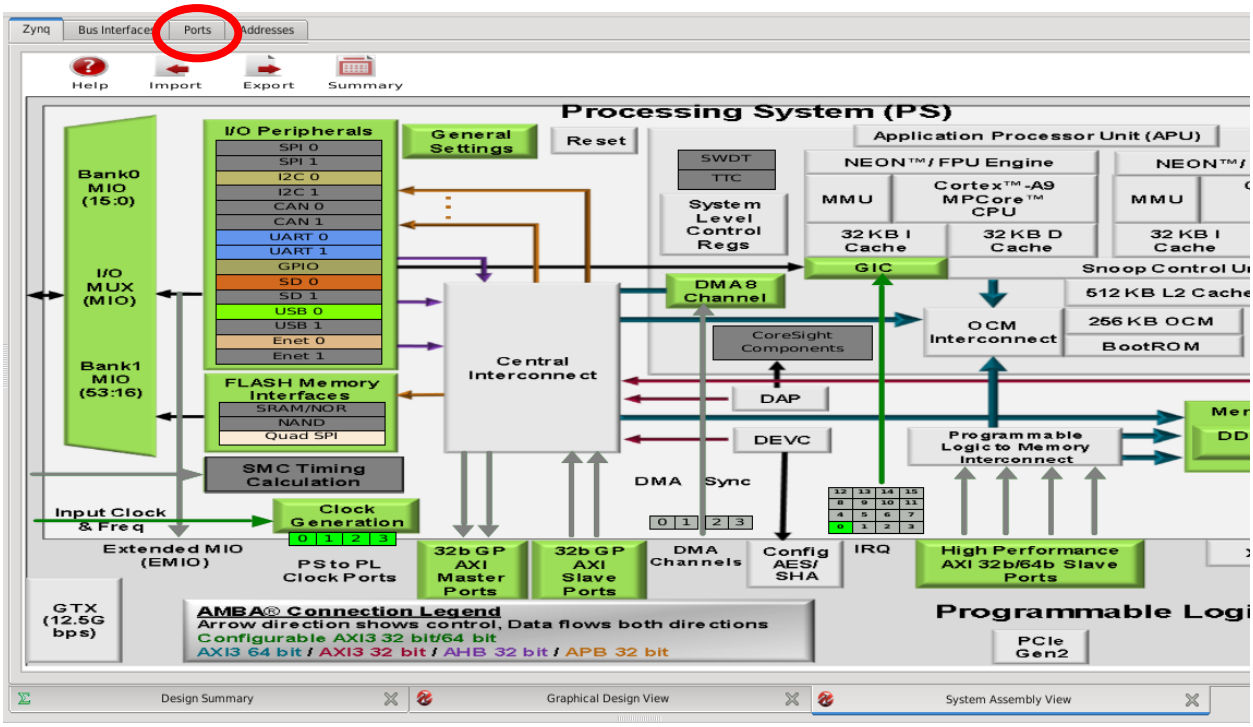


2. Enable the UART0 peripheral by clicking in the white box, and set the IO to EMIO. This makes the UART controller TX and RX pins go through the FPGA

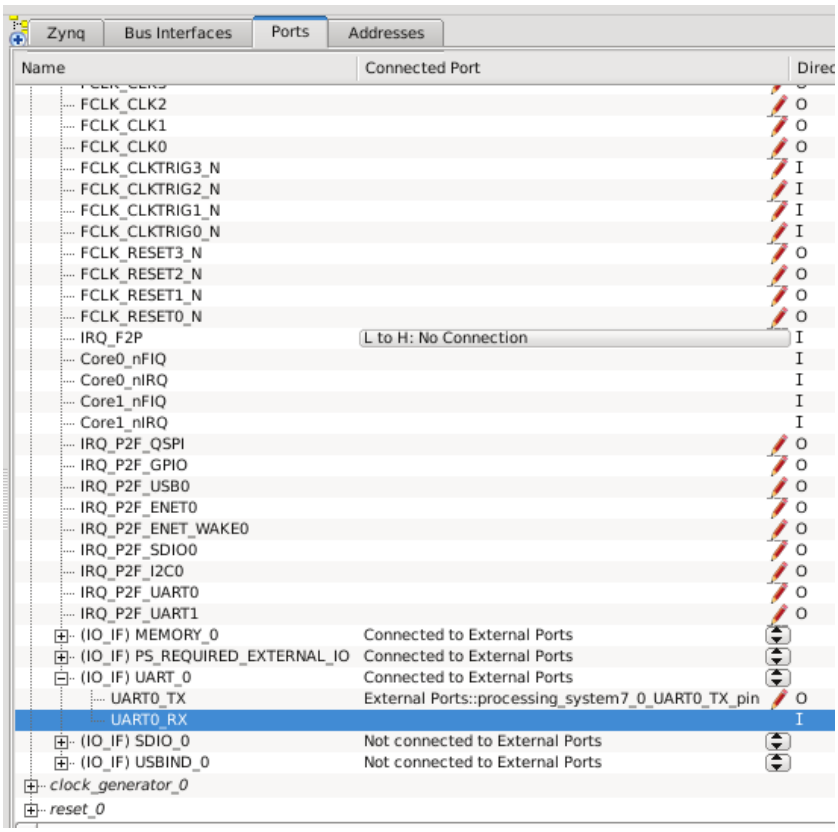
3. When done, click Close

[Back to Top]

4. Next, click on the Ports tab to view system connections

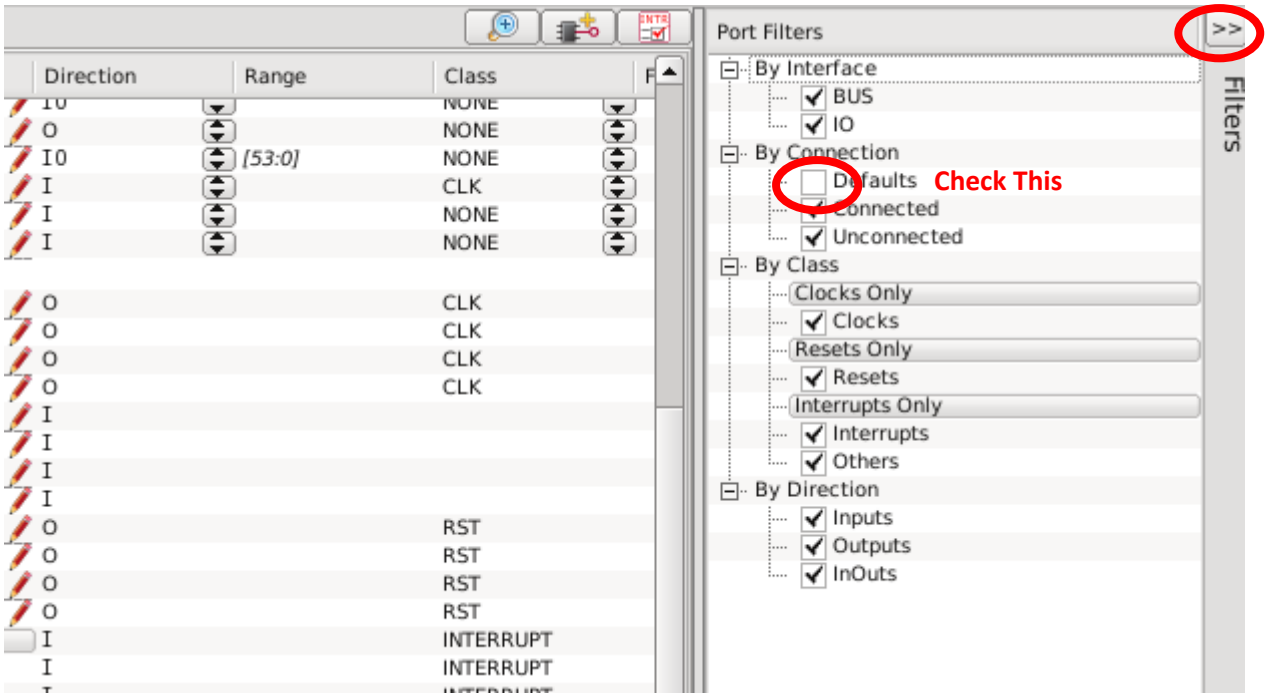


5. Expand processing_system7_0 and the (IO_IF) UART_0 as seen below



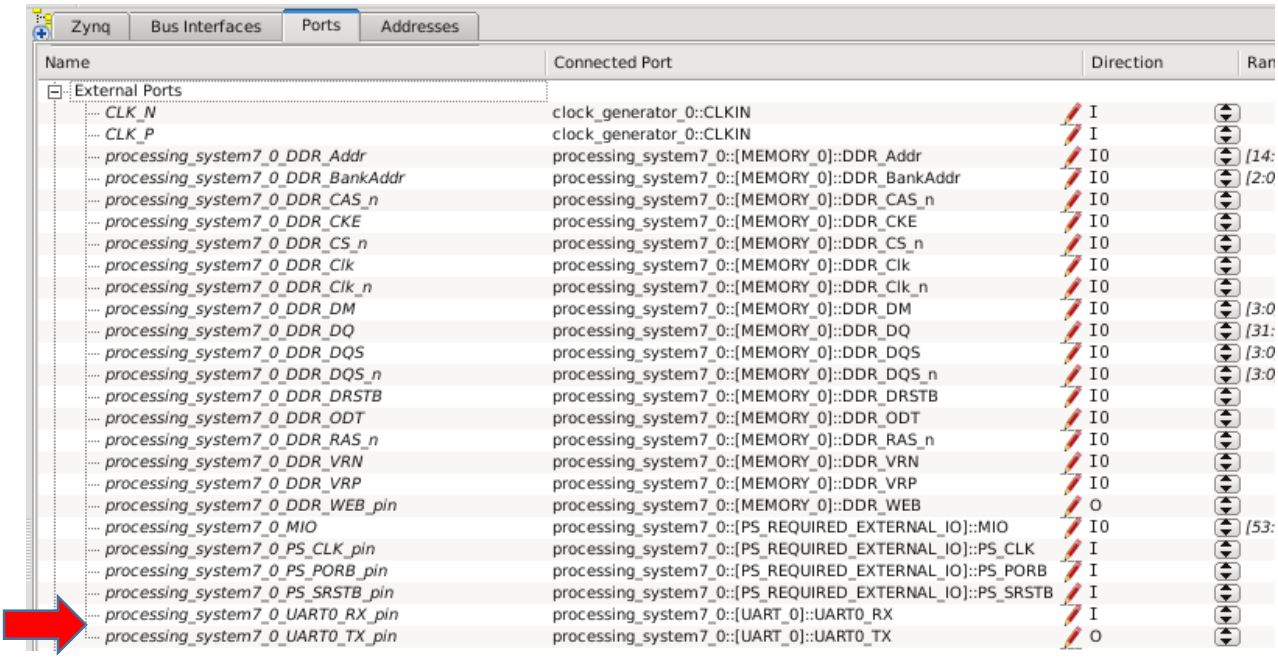
[Back to Top]

i. NOTE: if the sections are not there, you need to enable viewing all connections. This can be done by clicking on the << button and **checking** the Defaults connection box



6. Right-click on the UART0_TX and UART0_RX connections and click Make External. This tells the system that we want to connect those ports to an external source (our PMOD)

7. Once the ports are external, move to the top of the ports list and expand External Ports

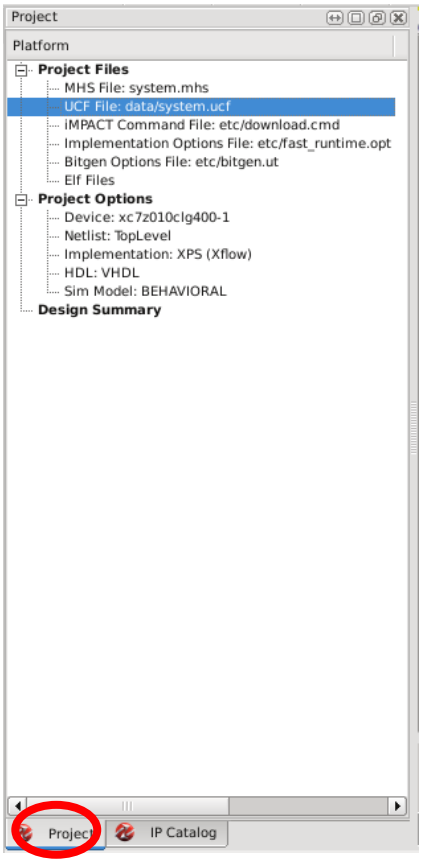


[Back to Top]

8. Take note of the **Name** for these (processing_system7_0_UART0_RX_pin). We will be using this to assign the proper pin location

9. On the left side of XPS, there should be a Project Platform window with two tabs at the bottom: Project and IP Catalog. Click on Project

10. Double click on UCF File: **data/system.ucf** under the Project Files list (highlighted in blue)



11. The system.ucf file will open (probably blank) where you can put in pin assignments for external ports. For this specific case, we will assign the TX and RX pins to PMOD-B pins 2 and 3. It should look something like this below.

```
1  NET "processing_system7_0_UART0_TX_pin"      LOC=U20 | IOSTANDARD=LVCMOS33; #IO_L22P_T3_AD7P_35  #JB2
2  NET "processing_system7_0_UART0_RX_pin"      LOC=V20 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_AD15P_35 #JB3
```

NOTE: the pins are assigned by Name and Location (please see the Zybo board spec sheet PMOD descriptions and the Master UCF file for all pin locations)

[Back to Top]

12. Finally, click Export Design

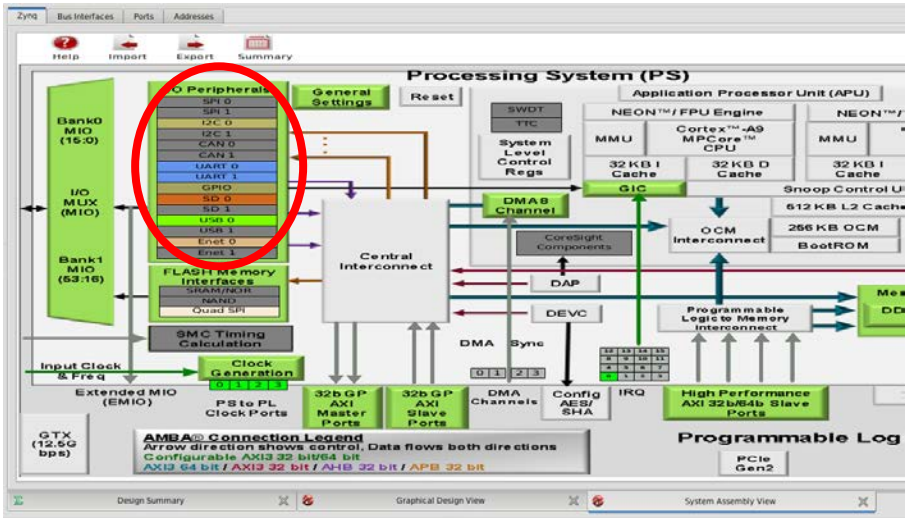
13. A Export to SDK / Launch SDK prompt will appear. Ensure XSDK is closed, if it was open, and click Export & Launch SDK (Move to Section 4: XSDK)

FORWARNING: The exporting process can take several minutes to complete or over an hour depending on machine load and how much logic needs to programmed.

3.5. Enabling I2C for Sensor Board Use

The Zybo board has two I2C controllers that can be used to communicate with peripherals. For example, MicroCART uses one I2C controller to talk with the SparkFun 9-degrees of freedom sensor board (MPU9150). A LOT of time has been spent getting the proper configurations for this to work, so PLEASE follow the instructions carefully.

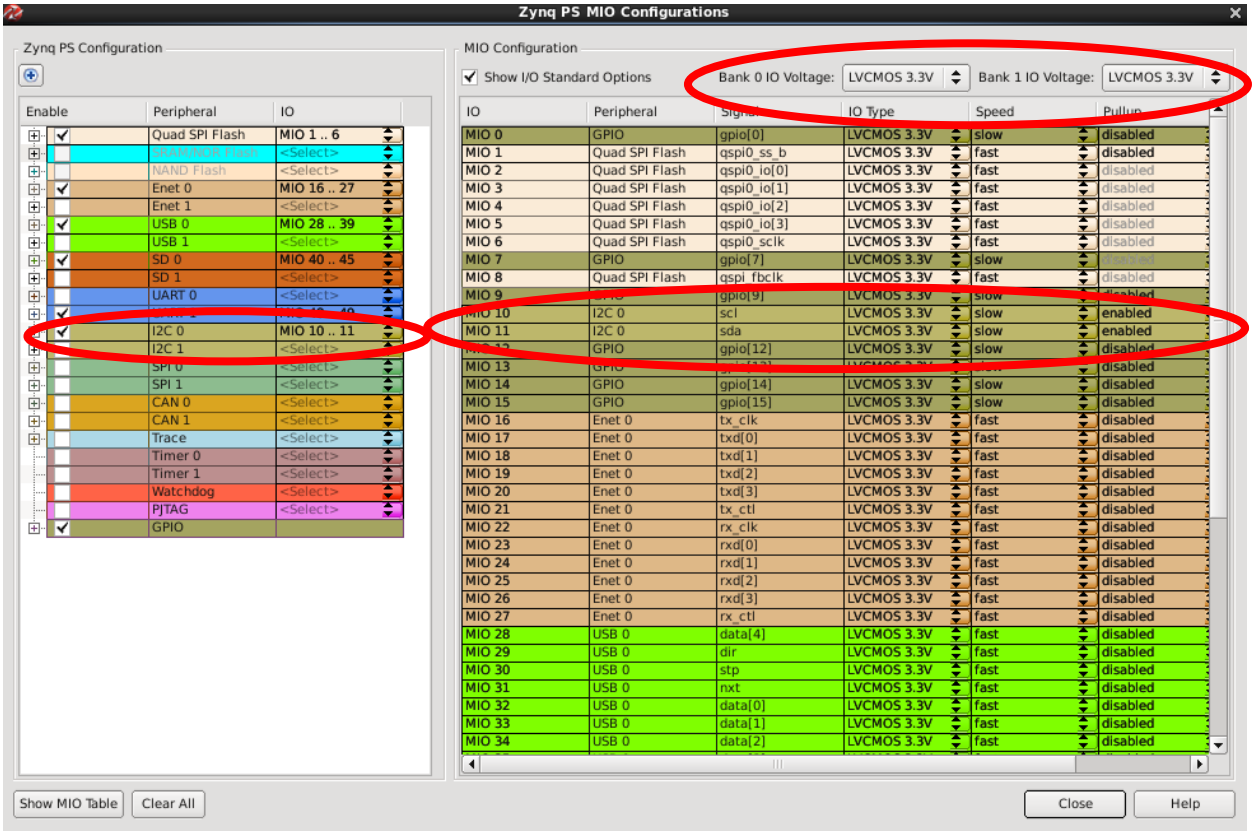
1. In XPS System View, click on the green I/O Peripherals box to bring up MIO Configuration box.



[Back to Top]

2. **VERY IMPORTANT** Check the Show I/O Standard Options

3. Chance I2C0's IO to MIO 10..11, set LVCMOS 3.3V for both Bank 0 and 1, **ENABLE** Pullup on MIO 10 and 11



NOTE: MIO10 and 11 automatically route to PMOD-F pins 2 (SCL) and 3 (SDA). If you'd like to assign them to alternative PMODS, Change the connection to EMIO and preform pin assignment process starting at Step 4 for UART0 configuration.

4. Start the Export Design process (Synthesizing an XPS Project)

[Back to Top]

4. Xilinx Software Development Kit (XSDK)

The XSDK is the main program used to create software applications for the Zybo board as well as flash the FPGA bitstream file onto the board. There are three main things required for a project:

1. A system hardware platform (automatically exporting design from XPS)
2. A Board Support Package (Contains software functions for interacting with the Processing System controllers i.e. UART and I2C or logic cores on the FPGA)
3. An Application Project (A simple hello world, an NES emulator, or something else)

As mentioned above the system_hw_platform is imported for us by XPS when we export our bitstream file to the XSDK. We'll walkthrough the other two items as well as programming the FPGA from within the XSDK.

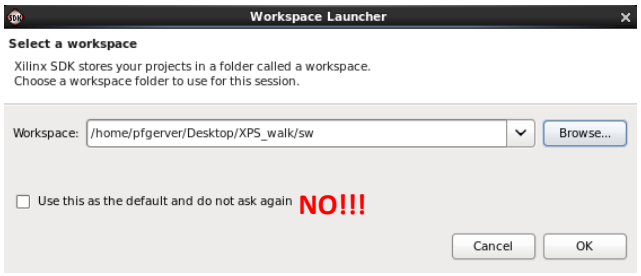
4.1. Exported System Hardware Platform

1. Whenever Export & Open XSDK or Export Only are clicked, the bitstream file will be generated and automatically open up XSDK and/or update XSDK if it is open.

2. When opening XSDK, you will be prompted for a workspace. A good naming scheme is to put the workspace into a folder called "sw". Either give a location of a new workspace or select a task's sw directory to open up it up.

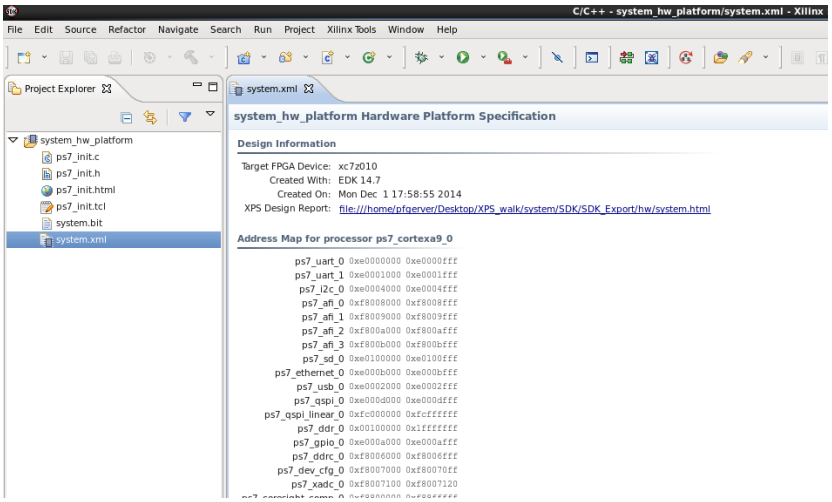
   DO NOT: check "Use this as the default and do not ask again"

3. Click OK when a proper directory has been given.



4. View the Eclipse-based XSDK. As you can see, the system_hw_platform is already in there for us.

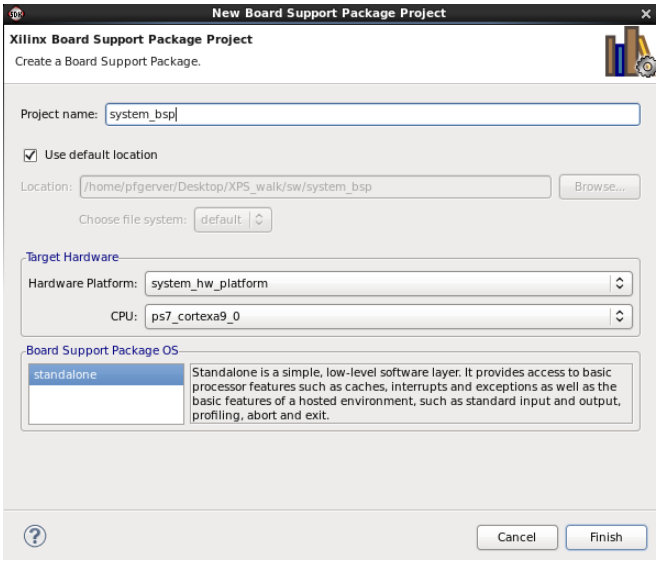
[Back to Top]



4.2. Creating a new Board Support Package (BSP)

The BSP package is responsible for grabbing all necessary Xilinx library files so they can be called by your program. In other words, it holds the drivers necessary to interface with the hardware on the board.

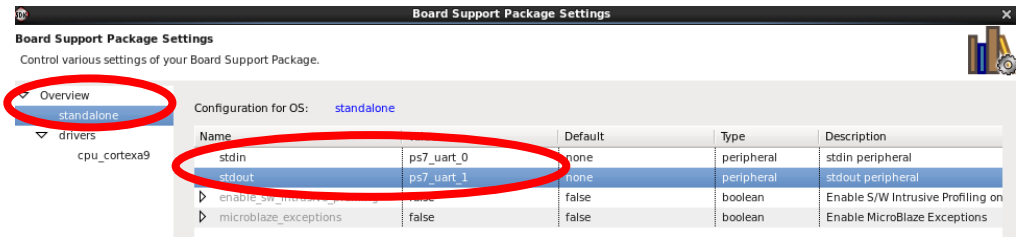
1. Click File and select New->Board Support Package
2. In the New Board Support Package Project, enter a project name (system_bsp in the example).
3. Ensure system_hw_platform and ps7_cortexa9_0 are selected in Target Hardware
4. Ensure “standalone” is selected.
5. Click Finish to create the BSP



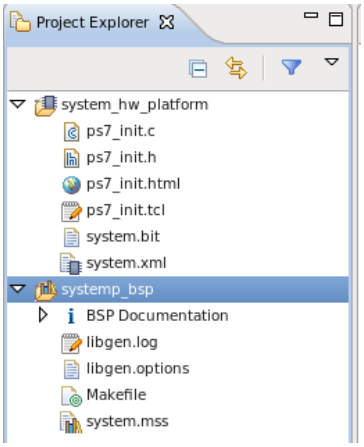
[Back to Top]

IMPORTANT: If UART0 is enabled, follow the steps below, otherwise ensure these are set properly anyway.

6. Click on "standalone" and change the value of stdin AND stdout to **ps7_uart_1**



7. Click OK when done, you should now see the BSP in the project explorer



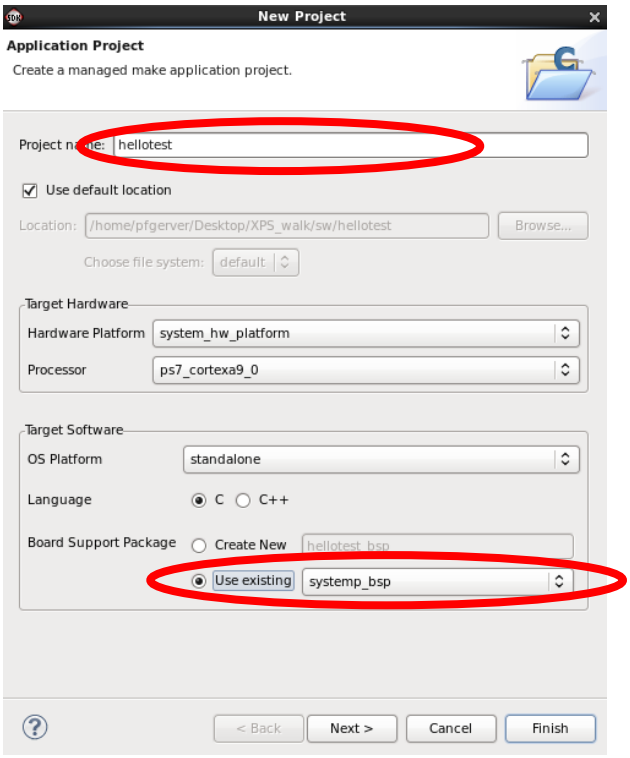
4.3. Creating a new Application Project

Lastly, we have the software program that will run on the board. This is the last piece to the Zybo puzzle, and we're almost there. Projects can be written in C or C++, and can use some standard libraries like stdio, stdlib, string, and otherse. HOWEVER, some libraries (often from Linux) are not implemented like time.h and other things that an OS would handle are not available and other methods must be used. Hopefully you do not run into these instances. (Off topic hint: If you need timing things, check out xtime_l.h)

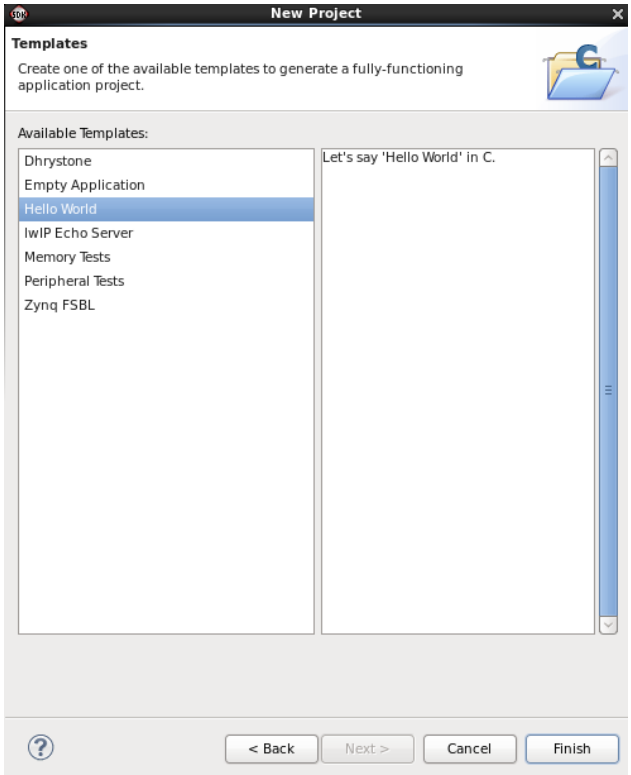
1. Click File -> New -> Application Project to open the New Project box

[Back to Top]

2. Enter a project name, and select the “Use existing” radio button for the BSP. We want to use our newly created one rather than create one.

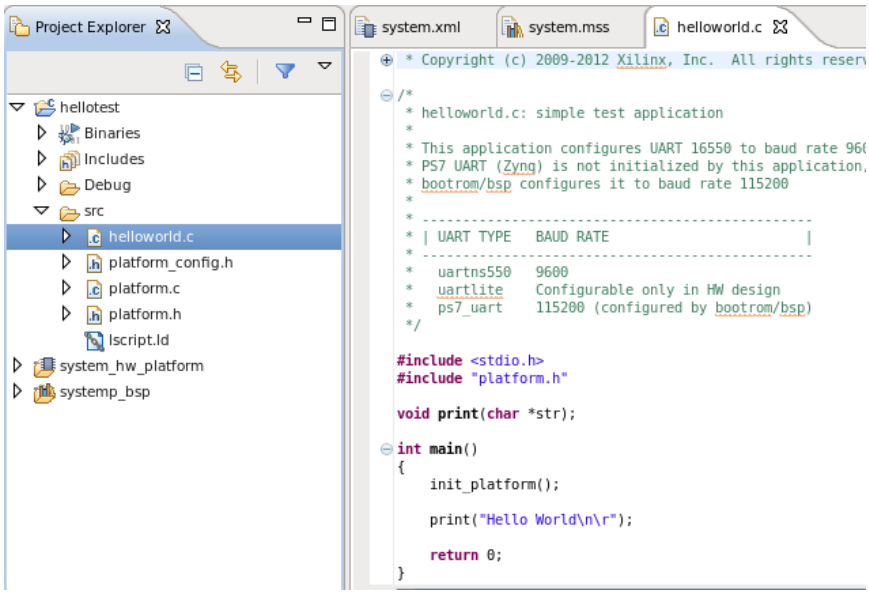


3. Click next and select a template (Hello World is probably best) and hit Finish



[Back to Top]

4. The new project should appear on the Project Explorer. Expand the project, src, and open the helloworld.c



4.4. Configuring JTAG

We now have all the components, we're now ready to program the board, but wait, we haven't setup how we're going to communicate with the board.

You may now plug in the board to the development computer (yours, Coover machine, etc.)

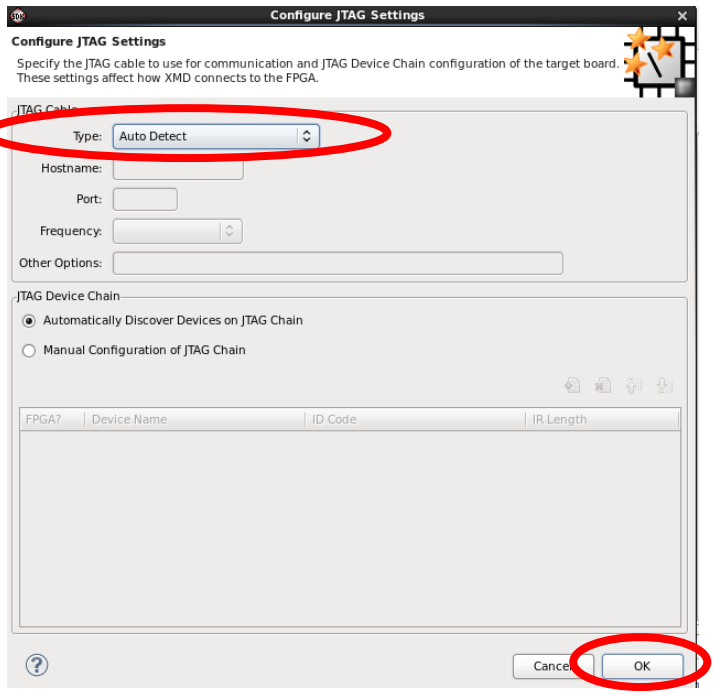
Flip the switch to power on the board.

1. In the menu bar, click Xilinx Tools -> Configure JTAG

2. If the board is plugged into Coover 3050-11,-12 or a personal PC, then use Auto Detect (It is ill-advised to be programming the Zybo board from a remote linux machine at this point)

[Back to Top]

3. Click OK

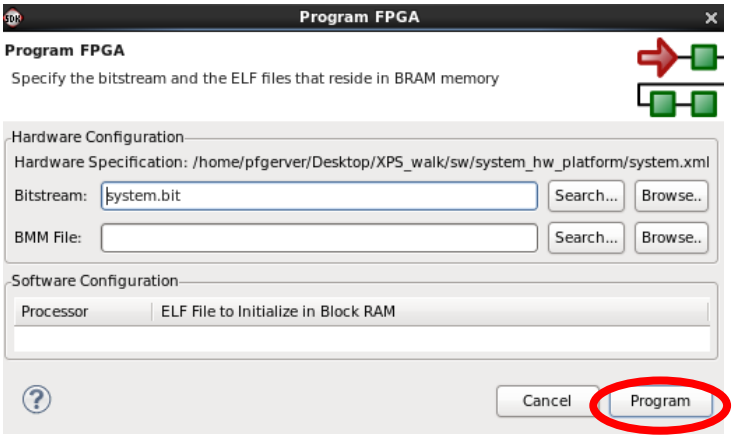


4.5. Launching an Application Project

With all our pieces ready to go, we can finally program the board and launch our software program.

IMPORTANT: A COM terminal should be open after the board has been turned on AND before launching the program. Either go through Putty on Windows (COMn Baud 115200) or if through a linux machine (sudo screen /dev/ttyUSB1 115200).

1. On the menu bar, click Xilinx Tools -> Program FPGA, and click Program to start the process



[Back to Top]

2. If the JTAG configurations and setup were correct, the bitstream should load onto the board and a blue LED should turn on indicating the board is DONE programming.

3. With the board program, we can now launch our software, click on the Application Project we want to launch and click the Green Play Arrow button to start.

4. View the data received from the board through the COM terminal.

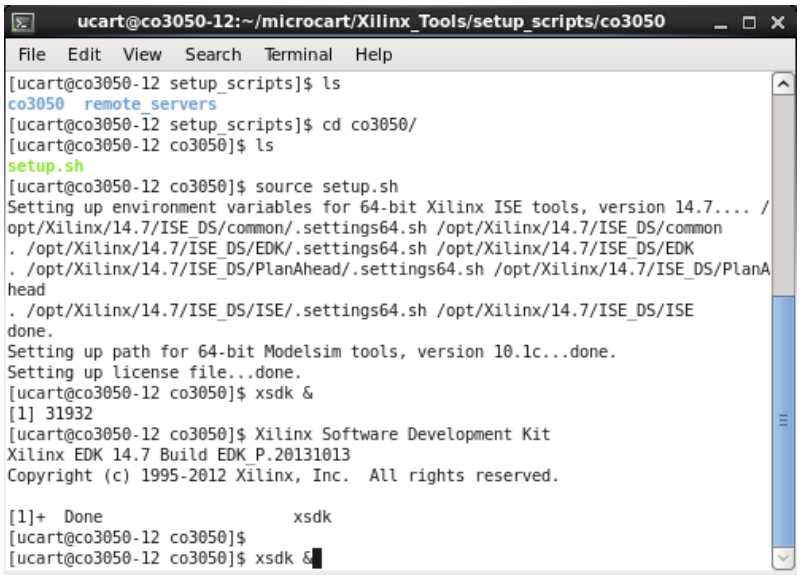
[Back to Top]

5. Running a MicroCART Task Example

We'll run through an example of how to run a MicroCART example. Please, please be read documentation for the task before running it since there might be some peripheral you need to attach first or for other configurations.

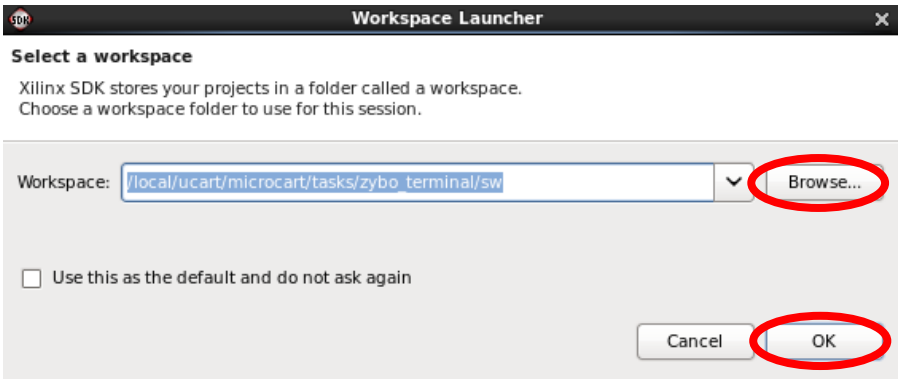
5.1. Zybo Board Terminal

1. First, open up XSDK by running `xsdk &`



2. You should now see the workspace prompt come up. Click Browse… and navigate to "tasks/zybo_terminal/sw".

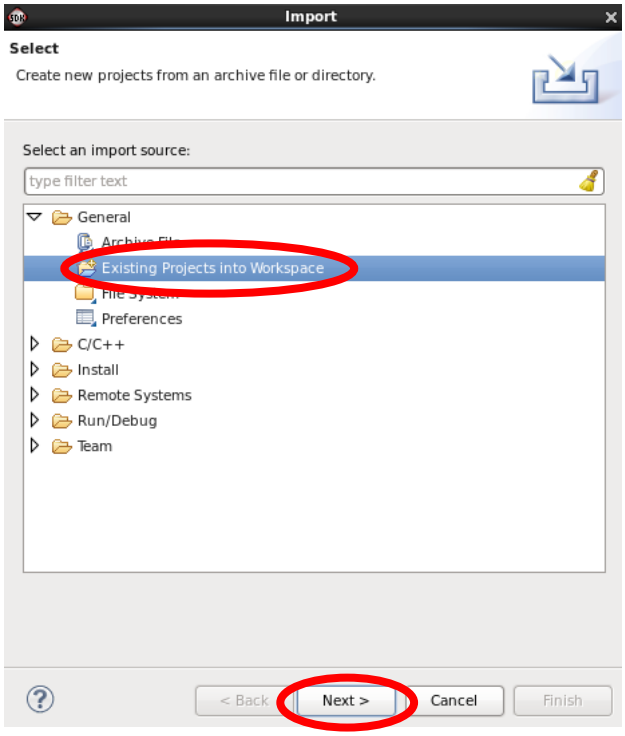
3. Click OK. Do it now, and **DO NOT** check the Use this as the default and do not ask again box.



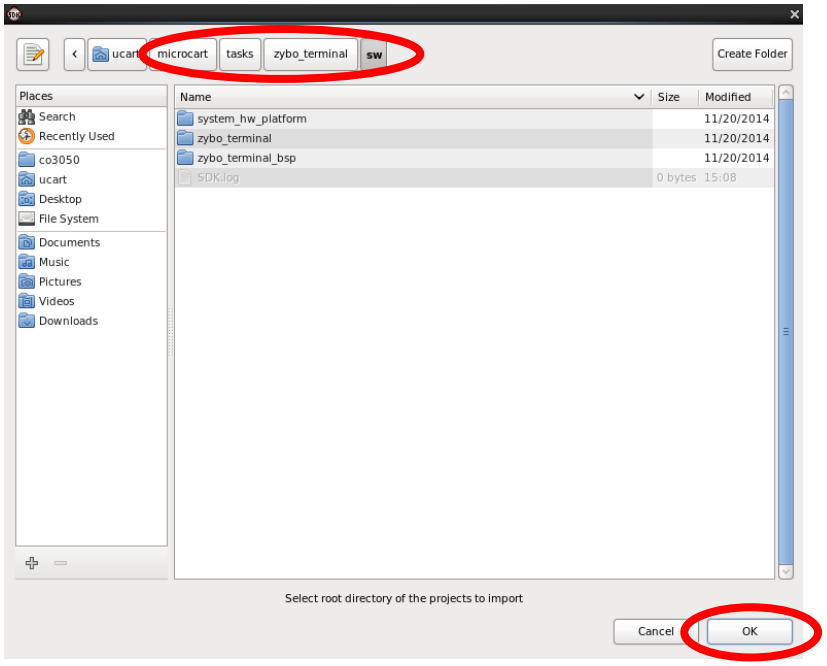
4. Don't be alarmed if the workspace is empty. We just need to import the project that is already there. Click File -> Import ...

[Back to Top]

5. In the Import window, go to Existing Projects into Workspace under the General folder. Click Next.

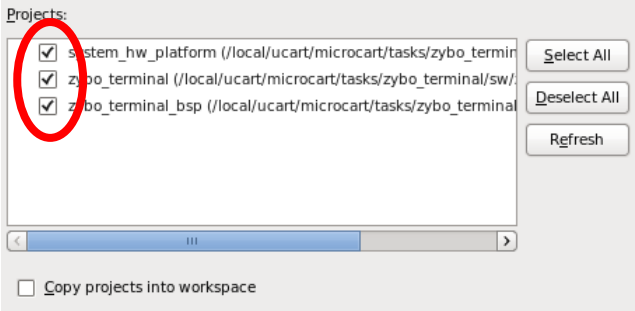


6. Choose the Select root directory and click Browse… Once again, navigate to "tasks/zybo_terminal/sw"



[Back to Top]

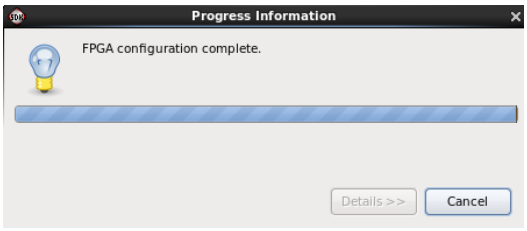
7. Click OK and view the selected projects in the Import window (ensure all three components are checked)



8. Click Finish to import the project into the workspace.

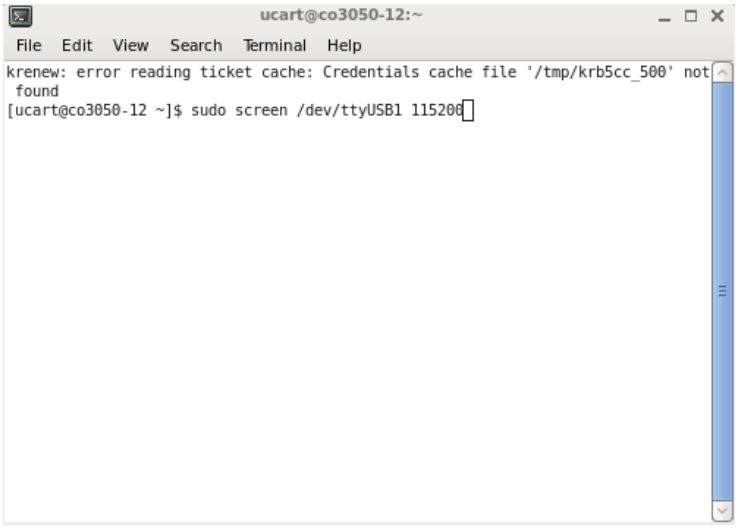
9. Next, configure the JTAG appropriately (Auto Detect should be adequate). Also, plug in the Zybo board if you haven't yet.

10. Program the FPGA by going to Xilinx Tools -> Program FPGA.



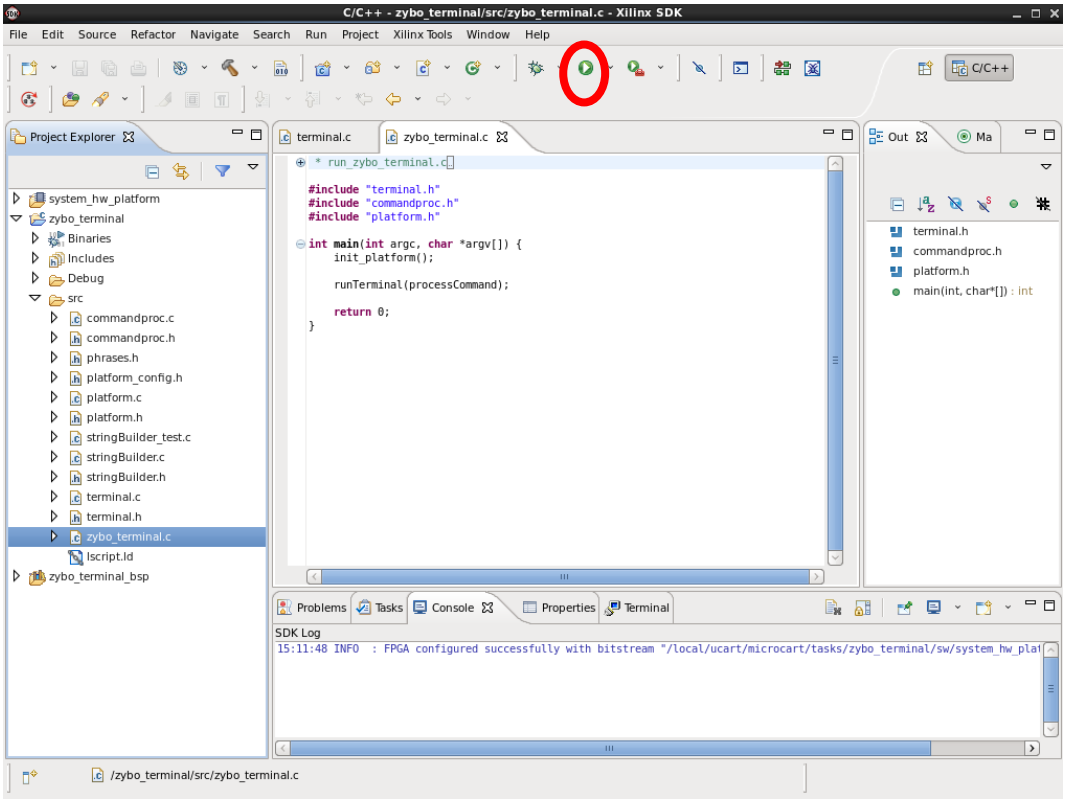
11. Once done, check that the blue LED on the Zybo board is on.

12. Next, open up a COM terminal for the board so we can view and interact with the board when the program is running. In a linux terminal (or Putty), open the connection:



[Back to Top]

13. Go back to XSDK and click the green play arrow to launch the program



14. In the terminal, you should now see a prompt. Enter “hello”, “talk”, and then “led”. Make sure to watch the board. You can also walkthrough the code to see what other commands there are

[Back to Top]

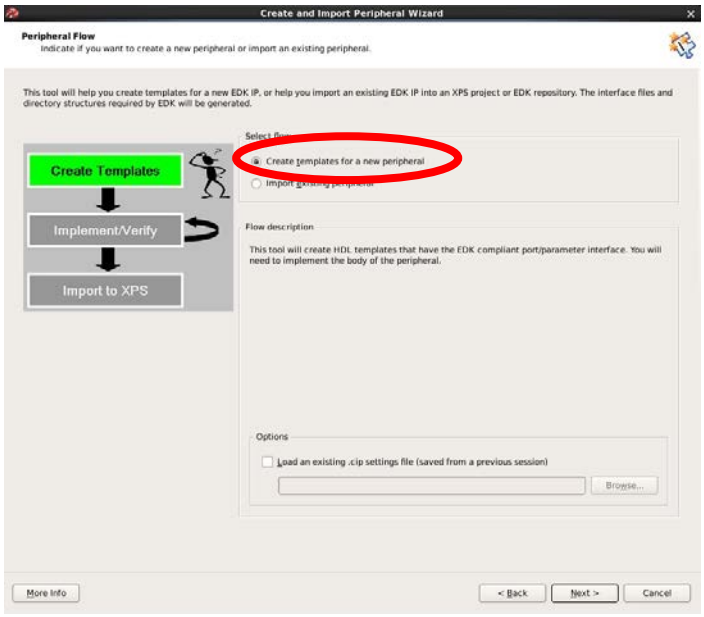
6. Creating a Custom AXI IP Core (Custom Logic Block)

While Xilinx provides many IP cores for users to configure the FPGA to interact with the Zybo's peripherals, there comes a time when we need to use the FPGA to do something unique like generate signals or record signals coming in from peripherals that can't be done in software. This is where custom logic blocks are needed and we'll need a way to convert our ideas into usable blocks in XPS.

6.1. Getting Started

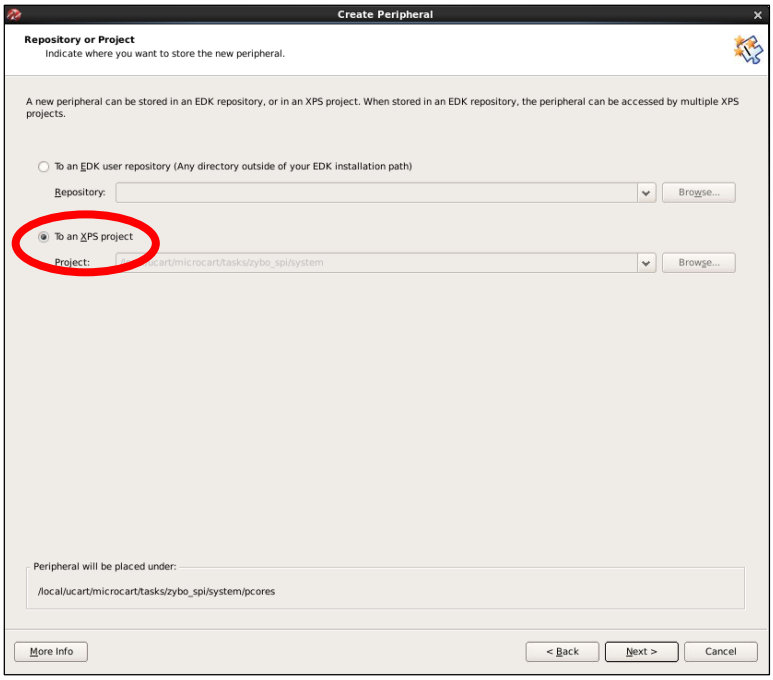
1. With the Xilinx environment setup, start XPS
2. Create or open a new project
3. On the menu bar, click Project -> Create and Import Peripheral

4. When the wizard dialog box opens, ensure "Create templates for a new peripheral" is selected and click Next



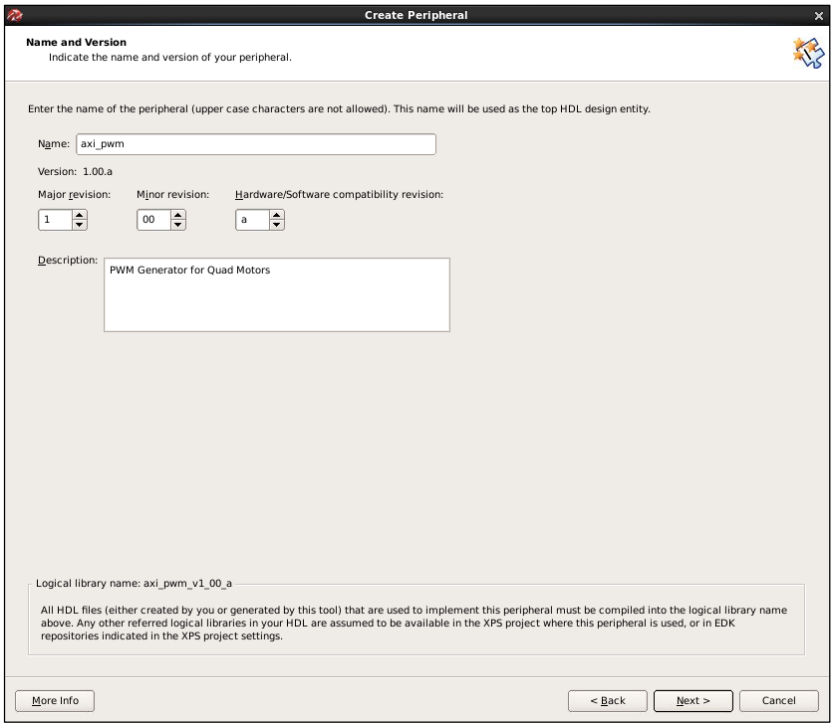
[Back to Top]

5. Ensure “To an XPS Project” button is selected. Click Next



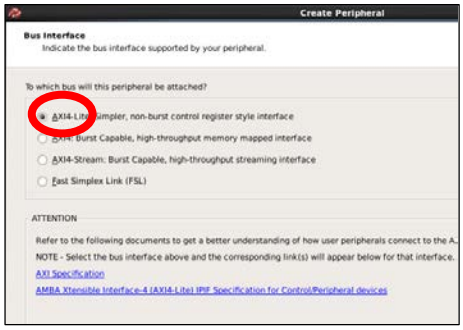
6. Name the custom core something unique. It is recommended to name it “axi_<name>” because the Zybo board uses AXI buses to communicate data. For our case, we’ll use “axi_pwm” because we’re creating a core that can generate PWMs.

7. Latsly, add a description if desired and click Next

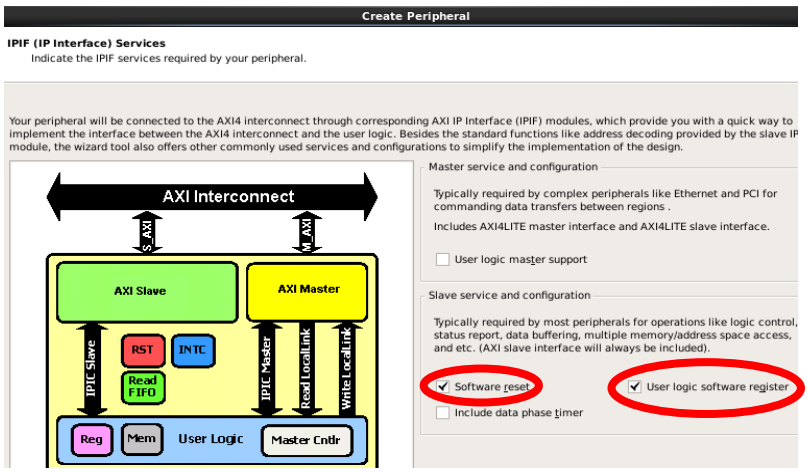


[Back to Top]

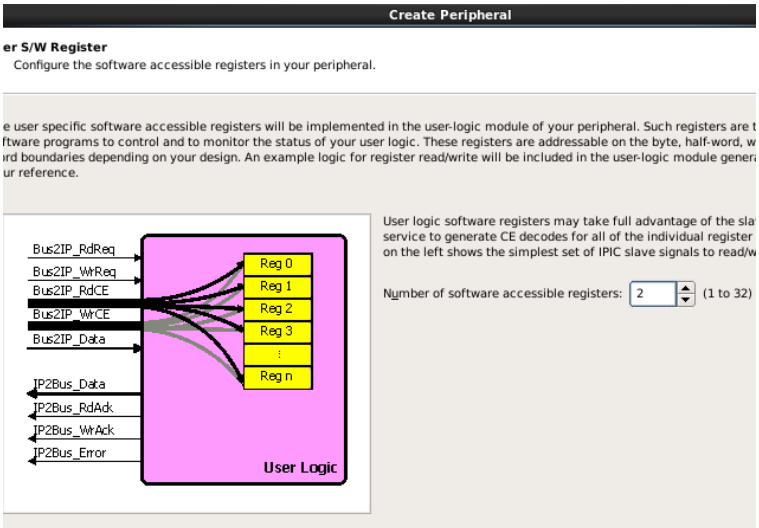
8. For most purposes, the **AXI4-Lite** interface is preferred so it can have registers that are easy to access in software. Click Next



9. Make sure “Software reset” and “User logic software register” are checked. Click Next

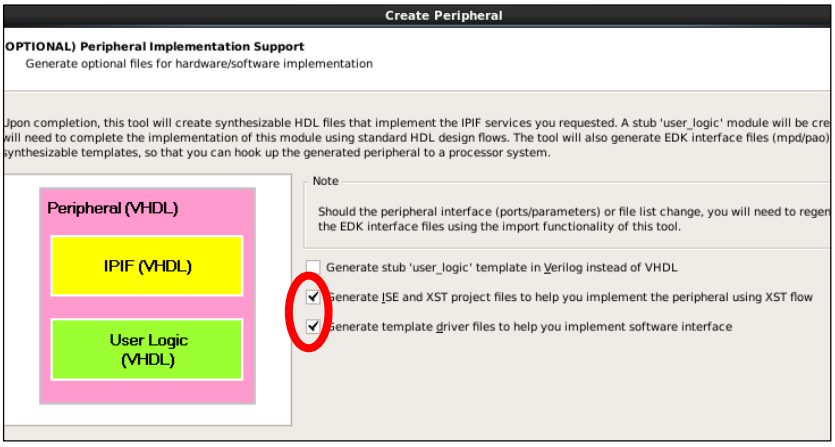


10. Set the number of registers the core should have (we’ll be using 2, though this number will highly vary between purposes). When you’re ready click Next twice because the default IPICs are fine

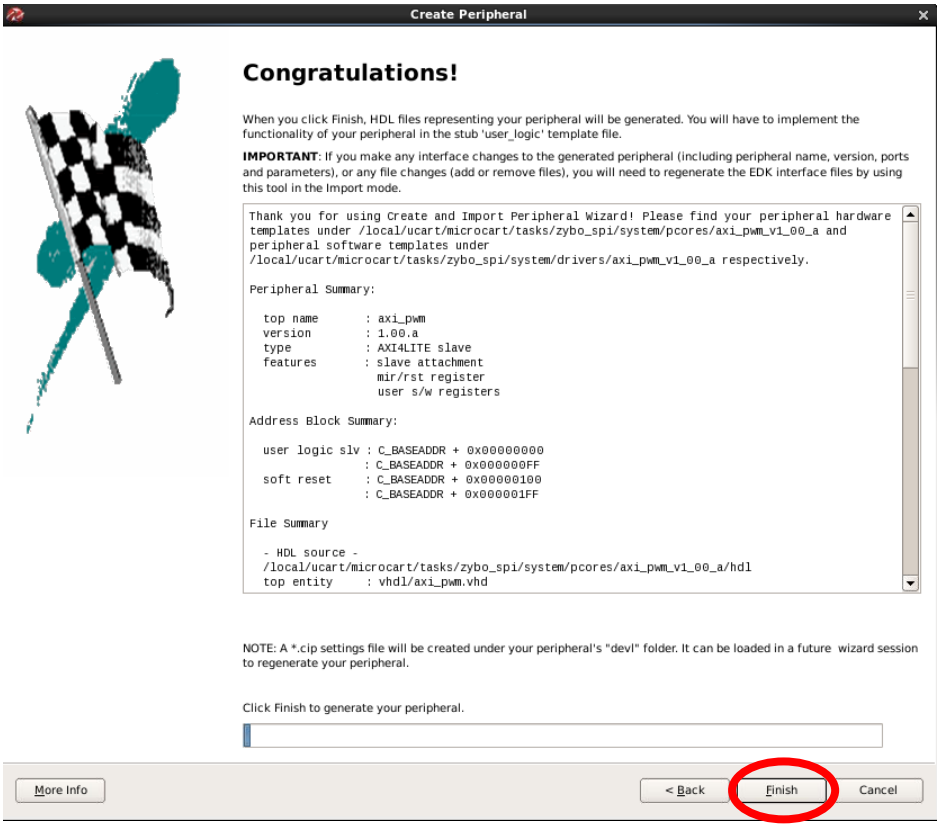


[Back to Top]

11. For the Peripheral Implementation Support, make sure “Generate ISE and XST projects files” and “Generate template driver file” are checked. Click Next



12. Review the summary and click Finish when satisfied. We’ve created a new peripheral, but it’s an empty logic block, so we’ll work on implementing that logic. Go ahead an minimize XPS at this point since we won’t need it for awhile.



[Back to Top]

6.2. Xilinx Integrated Software Environment (XISE)

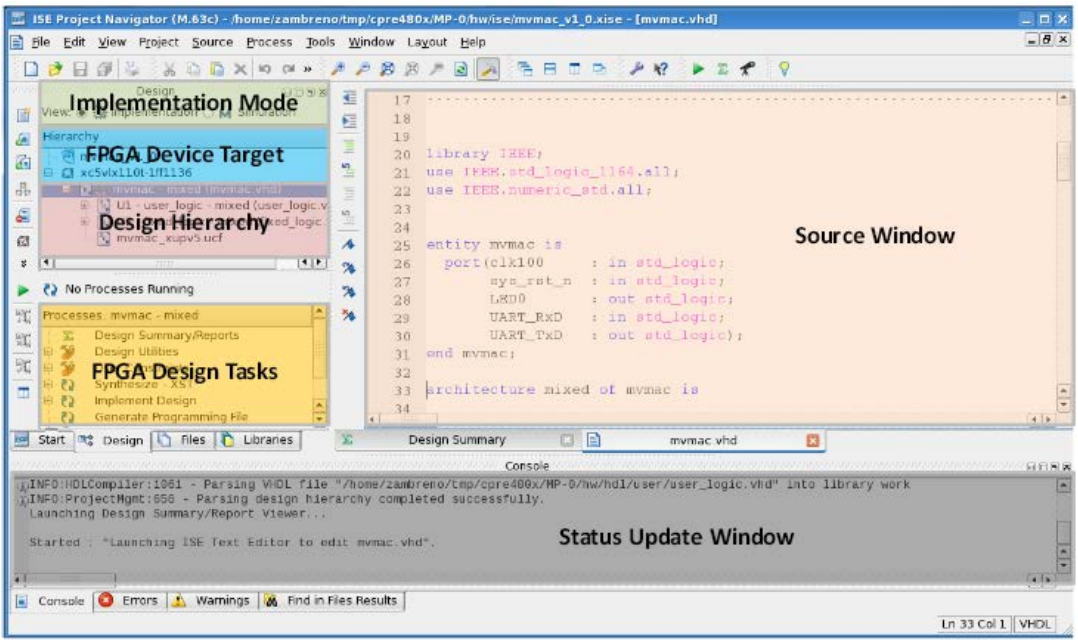
XISE is a program tool used to code and synthesize custom logic cores (in VHDL or Verilog) that can be used in a platform project. The HDL can be modified in XISE and the logic can be simulated through a simulation program like ModelSim for users to test their logic before synthesizing a logic block for use on an FPGA.

1. Navigate to the directory that contains your XPS project (i.e. system) and follow the path: system/pcores/axi_pwm_v1_00_a/devl/projnav
2. If you check the contents of the folder, you will find an .xise file (This is the XISE project file)
    a. A general layout for a custom core directory can be seen below:

```
axi_ppm_v1_00_a
|-- data
|   |-- axi_ppm_v2_1_0.mpd  // Core port description for XPS
|   `-- axi_ppm_v2_1_0.pao  // HDL file analysis order for XPS
|-- devl
|   |-- projnav       // ISE project file for pcore development
|   `-- synthesis     // Scripts to synthesize this module in XPS
`-- hdl               // Generated VHDL and Verilog files
    `-- vhdl
        |-- axi_ppm.vhd     // Top-level module
        `-- user_logic.vhd  // Main IP functionality placed here
```

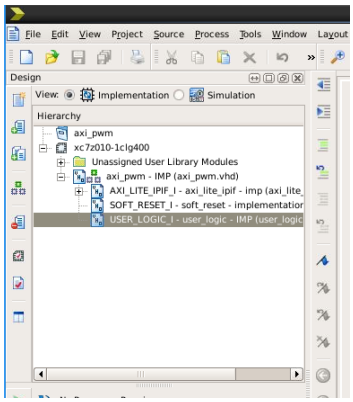
3. Open the XISE project by entering: `ise axi_pwm.xise &`

4. You should now be greeted by the XISE window (Here's a general layout)



[Back to Top]

5. First, we'll edit our "USER_LOGIC_I" which is where we will put our VHDL code for the logic block. Double click on USER_LOGIC_I in the Design Hierarchy section



6. You should see the automated code template pop up with lots of helpful warnings and indications where we should put our code. Since each different core is going to be different, it's up to you to properly code your VHDL.

   AXI_PWM.vhd is the entire core itself which holds a few smaller logic blocks inside including:
   1) The user logic that we will define

   2) A reset core

   3) The AXI bus wrapper that allows software to read/write to the registers within our core

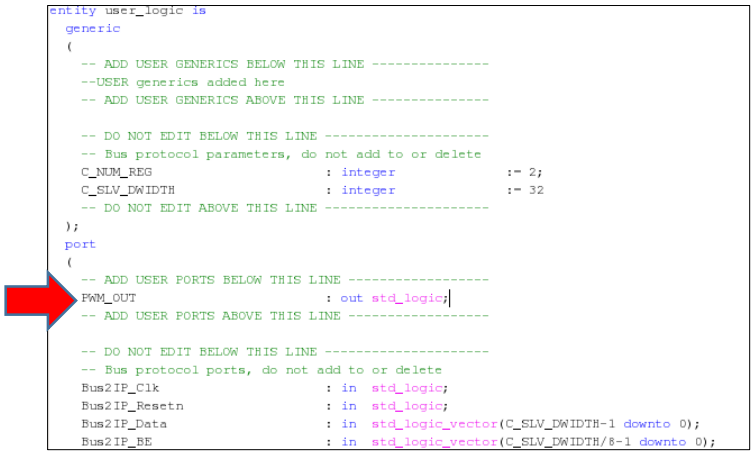
   In general, there will be three main things you will need to edit.
   1. User logic – For our combinational and synchronous logic
   2. Axi_pwm.vhd – For passing through any external ports you may need
   3. axi_pwm_v2_1_0.mpd – For declaring ports and notifies XPS that we'll have input or output ports for the core (useful for getting inputs or outputs to PMODs)

   For this section, it is highly recommended to read over MP-1 for CprE 488 as it may provide more insight (http://class.ece.iastate.edu/cpre488/labs/MP-1.pdf)

   For the remainder of this section, we will walk through declaring these external ports

[Back to Top]

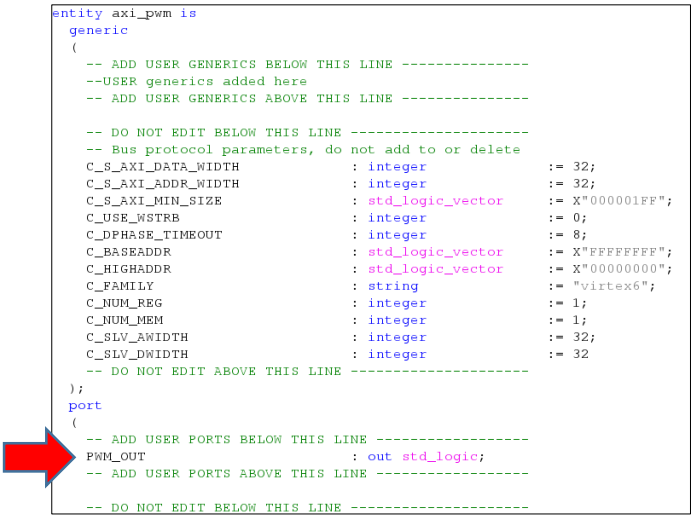
7. In USER_LOGIC_I, add the variable, direction, and type to the entity. We only want one signal out for our PWM



```
entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_NUM_REG                      : integer              := 2;
    C_SLV_DWIDTH                   : integer              := 32
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    PWM_OUT                        : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Resetn                  : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(C_SLV_DWIDTH-1 downto 0);
    Bus2IP_BE                      : in  std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
```

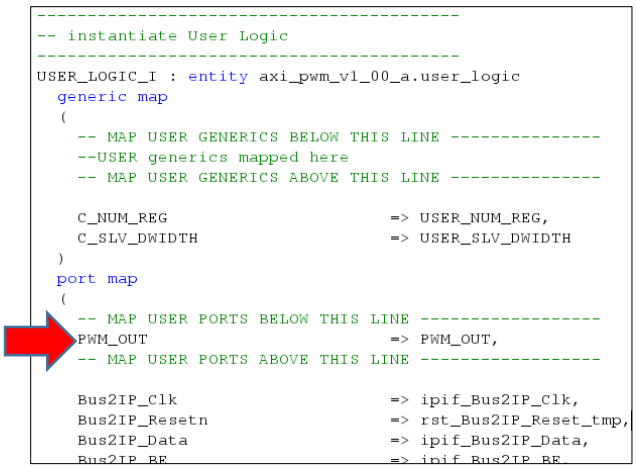
8. Save the user_logic and open up axi_pwm.vhd. Since we added a port for the user logic, we need to percolate this change up

Changes need to made in the axi_pwm entity



```
entity axi_pwm is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_S_AXI_DATA_WIDTH             : integer              := 32;
    C_S_AXI_ADDR_WIDTH             : integer              := 32;
    C_S_AXI_MIN_SIZE               : std_logic_vector     := X"000001FF";
    C_USE_WSTRB                    : integer              := 0;
    C_DPHASE_TIMEOUT               : integer              := 8;
    C_BASEADDR                     : std_logic_vector     := X"FFFFFFFF";
    C_HIGHADDR                     : std_logic_vector     := X"00000000";
    C_FAMILY                       : string               := "virtex6";
    C_NUM_REG                      : integer              := 1;
    C_NUM_MEM                      : integer              := 1;
    C_SLV_AWIDTH                   : integer              := 32;
    C_SLV_DWIDTH                   : integer              := 32
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    PWM_OUT                        : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
```

Changes also need to be made in the user logic instance within axi_pwm



```
-----------------------------------------
-- instantiate User Logic
-----------------------------------------
USER_LOGIC_I : entity axi_pwm_v1_00_a.user_logic
  generic map
  (
    -- MAP USER GENERICS BELOW THIS LINE ---------------
    --USER generics mapped here
    -- MAP USER GENERICS ABOVE THIS LINE ---------------

    C_NUM_REG                      => USER_NUM_REG,
    C_SLV_DWIDTH                   => USER_SLV_DWIDTH
  )
  port map
  (
    -- MAP USER PORTS BELOW THIS LINE ------------------
    PWM_OUT                        => PWM_OUT,
    -- MAP USER PORTS ABOVE THIS LINE ------------------

    Bus2IP_Clk                     => ipif_Bus2IP_Clk,
    Bus2IP_Resetn                  => rst_Bus2IP_Reset_tmp,
    Bus2IP_Data                    => ipif_Bus2IP_Data,
    Bus2IP_BE                      => ipif_Bus2IP_BE,
```

9. With all our ports covered within the HDL, we need to modify the MPD file that tells XPS that we have additional Ports. This file is not easily accessible in XISE, but go to File -> Open… Make sure you **change Files of type to All Files (*)**

10. Navigate to system/pcore/axi_pwm_v1_00_a/data/axi_pwm_v2_1_0.mpd and click Open

[Back to Top]

11. Scroll to the Ports section and add the following:
    `PORT PWM_OUT = "", DIR = O`
    (If the port is an input put DIR = I)

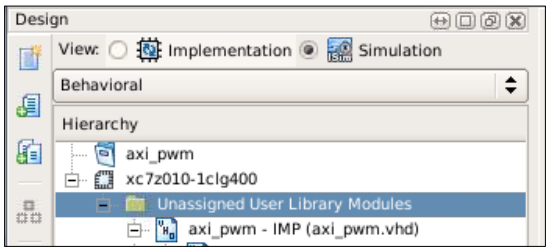
```
## Ports
PORT S_AXI_ACLK = "", DIR = I, SIGIS = CLK, BUS = S_AXI
PORT S_AXI_ARESETN = ARESETN, DIR = I, SIGIS = RST, BUS = S_AXI
PORT S_AXI_AWADDR = AWADDR, DIR = I, VEC = [(C_S_AXI_ADDR_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_AWVALID = AWVALID, DIR = I, BUS = S_AXI
PORT S_AXI_WDATA = WDATA, DIR = I, VEC = [(C_S_AXI_DATA_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_WSTRB = WSTRB, DIR = I, VEC = [((C_S_AXI_DATA_WIDTH/8)-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_WVALID = WVALID, DIR = I, BUS = S_AXI
PORT S_AXI_BREADY = BREADY, DIR = I, BUS = S_AXI
PORT S_AXI_ARADDR = ARADDR, DIR = I, VEC = [(C_S_AXI_ADDR_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_ARVALID = ARVALID, DIR = I, BUS = S_AXI
PORT S_AXI_RREADY = RREADY, DIR = I, BUS = S_AXI
PORT S_AXI_ARREADY = ARREADY, DIR = O, BUS = S_AXI
PORT S_AXI_RDATA = RDATA, DIR = O, VEC = [(C_S_AXI_DATA_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_RRESP = RRESP, DIR = O, VEC = [1:0], BUS = S_AXI
PORT S_AXI_RVALID = RVALID, DIR = O, BUS = S_AXI
PORT S_AXI_WREADY = WREADY, DIR = O, BUS = S_AXI
PORT S_AXI_BRESP = BRESP, DIR = O, VEC = [1:0], BUS = S_AXI
PORT S_AXI_BVALID = BVALID, DIR = O, BUS = S_AXI
PORT S_AXI_AWREADY = AWREADY, DIR = O, BUS = S_AXI

PORT PWM_OUT = "", DIR = O

END
```

12. Save the file and that covers adding external ports for cores. yay!

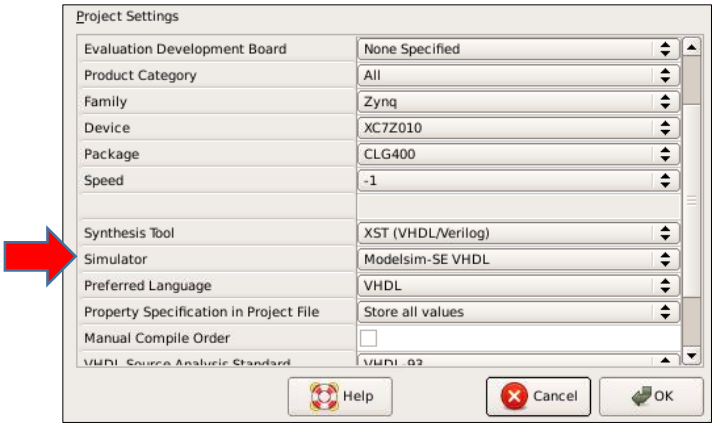
6.3. Simulating Logic with ModelSim or ISim

One of the nice things about XISE is the ability to startup a logic simulation session quickly for your user logic within the program itself. By default, XISE uses ISim to simulate, but we'll walkthrough setting it up for ModelSim and then showing how to start a session.

1. With your XISE project open, switch the view from Implementation to Simulation

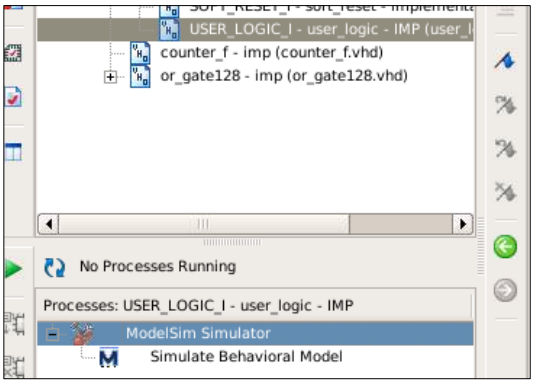


2. Click on Project -> Design Properties
3. In the list of Project Settings, find Simulator, and set it to ModelSim-SE VHDL. Click OK



[Back to Top]

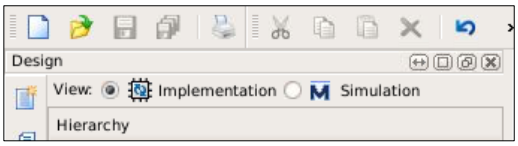
4. In the processes section, you should see your simulator of choice.
5. Highlight User_Logic_I, expand the Simulator as shown below, and double-click on ModelSim Simulator in the processes window to start the simulation.



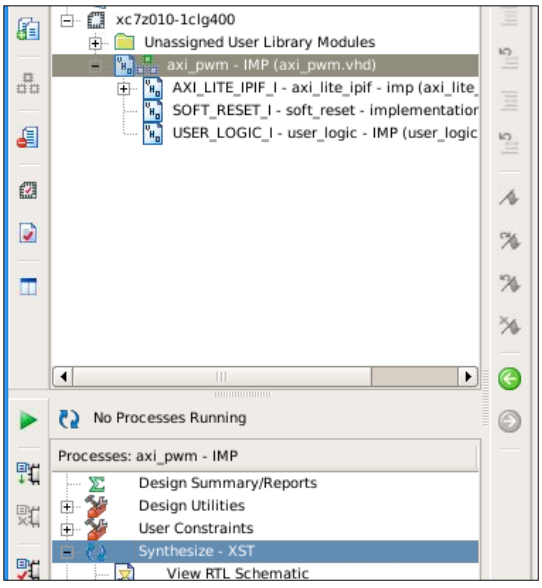
Sadly, this guide will not cover how to use ModelSim (Sorry)

6.4. Synthesizing Logic Core

1. After you've tested your logic and ready to get the core added to your project, switch to the "Implementation" view



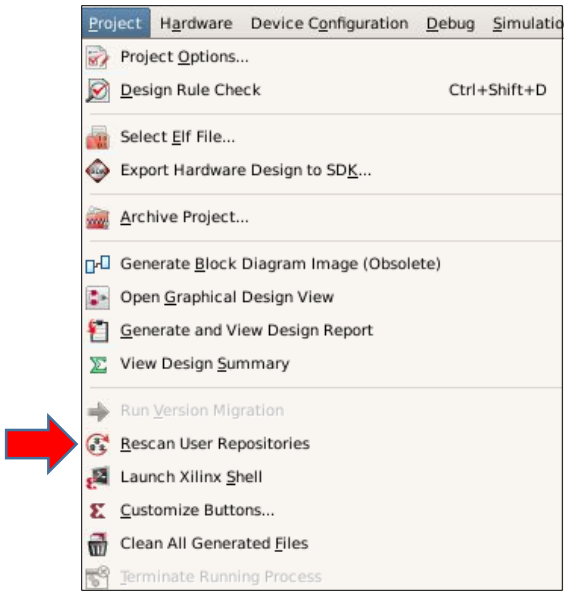
2. Select your top level HDL core (axi_pwm.vhd) and in the Processes, double-click "Synthesize – XST". That's it! Once the synthesis is completed successfully, that's it! Go ahead and close XISE.



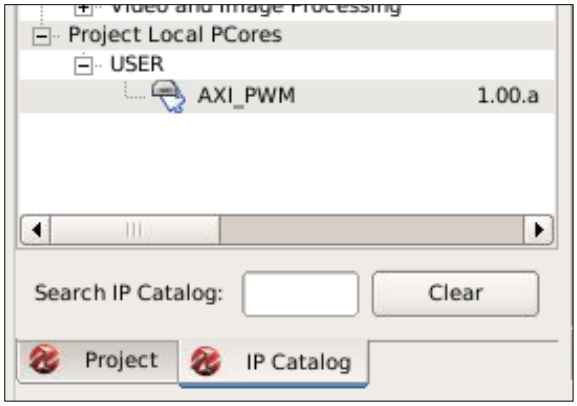
[Back to Top]

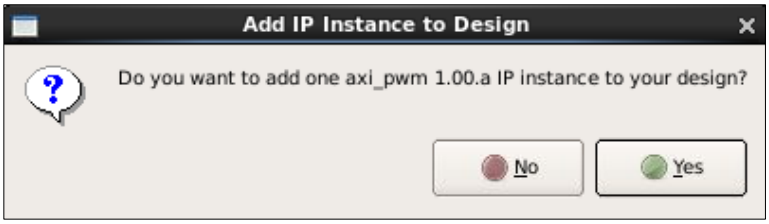
6.5. Integrating new core into XPS

1. With our core's logic synthesized, we need to add the latest version to our project. To do this open our minimized XPS project and simply click on "Project -> Rescan User Repositories"



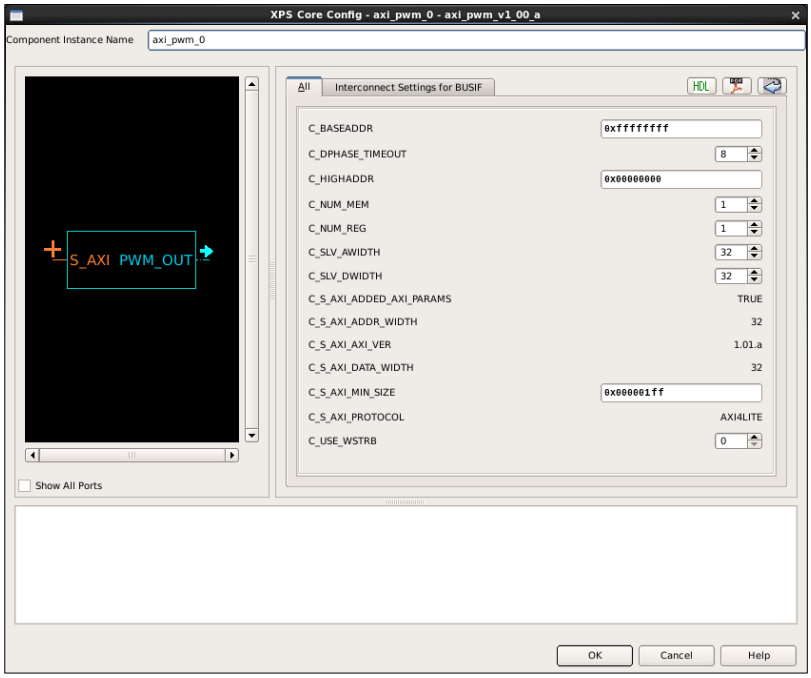
2. Under the IP Catalog, Expand "Project Local PCores" and "USER" to see our core
3. Double click on the core (axi_pwm) and click Yes so we add one core to our project (repeat as many times as needed)



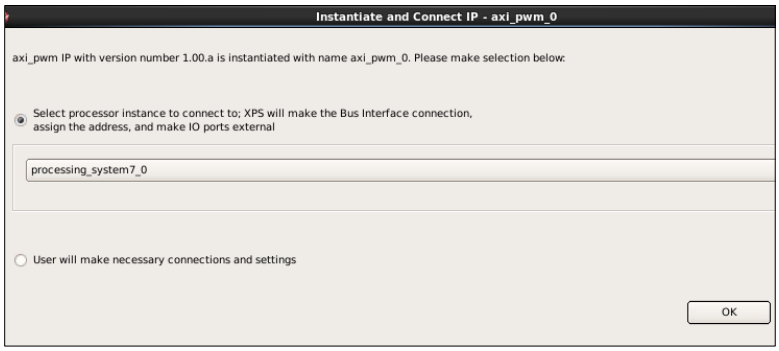


[Back to Top]

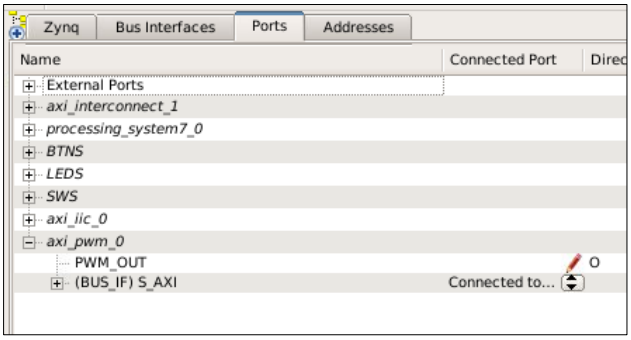
4. The internal settings for the core shouldn't need to be changed, so click OK



5. To make things easier for us, make sure "Select processor instance to connect" and ensure processor_system7_0 is selected click OK. This will automatically make the AXI Lite bus connection for us.

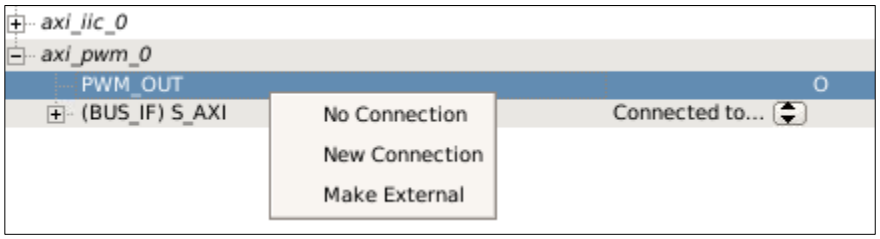


6. After the core has been added, go to the Ports tab to see our new core part of the system

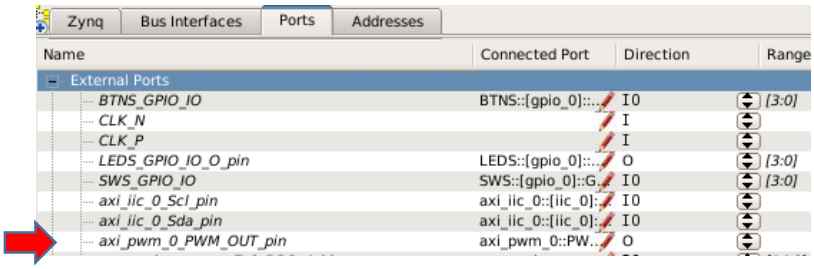


[Back to Top]

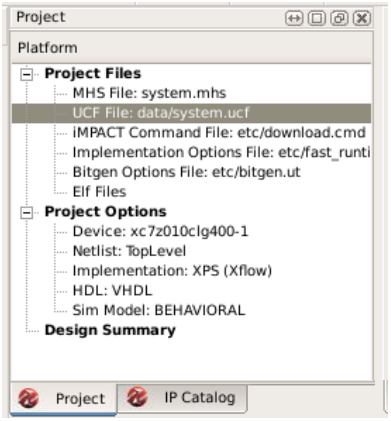
7. Since we have an external pin, we can connect it to another core if we wish, but since we're generating PWMs, let's make the pin external and connect it to a PMOD. Right-click on the PWM_OUT port and click "Make External"



8. Expand the External Ports section at the top and take not of the pin we created (axi_pwm_0_PWM_OUT_pin)



9. Switch to the Project tab and double-click the data/system.ucf file to open it up.
Pin reference (http://www.digilentinc.com/data/products/zybo/zybo_rm_b_v6.pdf)



10. Add the following line to the system.ucf file, then SAVE it

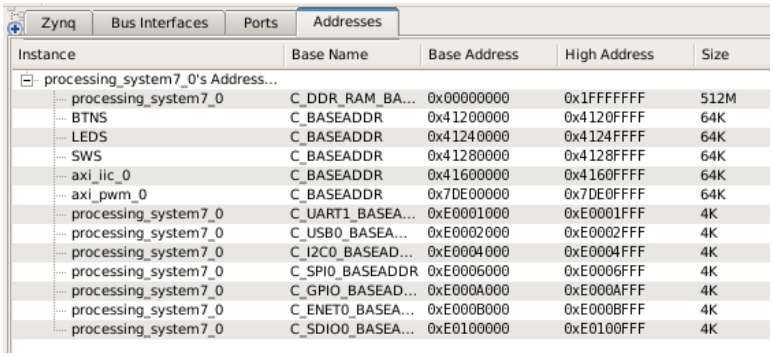
```
NET axi_pwm_0_PWM_OUT_pin LOC = T20 | IOSTANDARD=LVCMOS33; #JB1
```

11. Finally, synthesize the XPS project

[Back to Top]

6.6. Accessing Core Registers in Software

1. To access the registers stored in our core through software, we can identify what address the core is by going to the Addresses tab in XPS and see where the base address is.



Instance	Base Name	Base Address	High Address	Size
processing_system7_0's Address...				
processing_system7_0	C_DDR_RAM_BA...	0x00000000	0x1FFFFFFF	512M
BTNS	C_BASEADDR	0x41200000	0x4120FFFF	64K
LEDS	C_BASEADDR	0x41240000	0x4124FFFF	64K
SWS	C_BASEADDR	0x41280000	0x4128FFFF	64K
axi_iic_0	C_BASEADDR	0x41600000	0x4160FFFF	64K
axi_pwm_0	C_BASEADDR	0x7DE00000	0x7DE0FFFF	64K
processing_system7_0	C_UART1_BASEA...	0xE0001000	0xE0001FFF	4K
processing_system7_0	C_USB0_BASEA...	0xE0002000	0xE0002FFF	4K
processing_system7_0	C_I2C0_BASEAD...	0xE0004000	0xE0004FFF	4K
processing_system7_0	C_SPI0_BASEADDR	0xE0006000	0xE0006FFF	4K
processing_system7_0	C_GPIO_BASEAD...	0xE000A000	0xE000AFFF	4K
processing_system7_0	C_ENET0_BASEA...	0xE000B000	0xE000BFFF	4K
processing_system7_0	C_SDIO0_BASEA...	0xE0100000	0xE0100FFF	4K

2. This address can also be found in "xparameters.h" which is automatically generated to indicate all base addresses and extra things

```
/* Definitions for peripheral AXI_PWM_0 */
#define XPAR_AXI_PWM_0_BASEADDR 0x7DE00000
#define XPAR_AXI_PWM_0_HIGHADDR 0x7DE0FFFF
```

3. When your synthesis is complete, open XSDK

4. The first register can be accessed from the base address (for example, we have 2 registers for our PWM core). For our example, our two registers are used to specify the PWM period and pulse durations.

```
// All registers are 32-bit
int* PWM_Period = (int*) XPAR_AXI_PWM_0_BASEADDR;
int* PWM_Pulse = (int*) (XPAR_AXI_PWM_0_BASEADDR) + 1; // Or (int*) (XPAR_AXI_PWM_0_BASEADDR + 4)

*PWM_Period = 200000; // Set period to 20ms
*PWM_Pulse = 15000; //Set period to 1.5ms
```

That's it! You can now access the registers of your core. These easy to access registers are very useful for specifying settings for the core or reading values that the core provides to the software. Either way, it is important to have a good logic design AND diagram so you can program the logic quickly, test accordingly, and let know users know what registers do what.

[Back to Top]

7. Using a MicroSD Card to program the Zybo Board

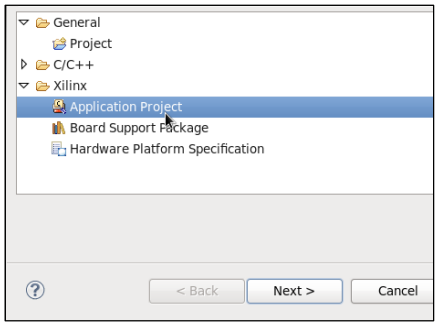
A nice feature about the Zybo board is the ability to program the FPGA and launch an application through a binary file located on a microSD card. We'll walkthrough the necessary steps to launch any program for the Zybo board using the great microSD.

PRE-REQUISITE: To program the board using the SD methods, the proper pins need to be connected. Look for the jumper that has JTAG, QSPI, and SD labeled. Move the blue jumper so it covers the SD option.

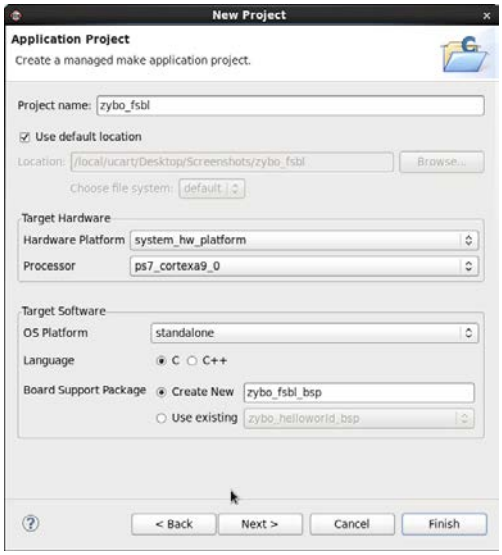
1. Open an existing project in XSDK that you wish to put on the Zybo
   a. Enter `xsdk &` in a terminal
   b. Open the workspace of the project (we'll be using zybo_helloworld)

2. Click File -> New Project in the XSDK IDE

3. Select a Xilinx Project, specifically an Application Project and click Next

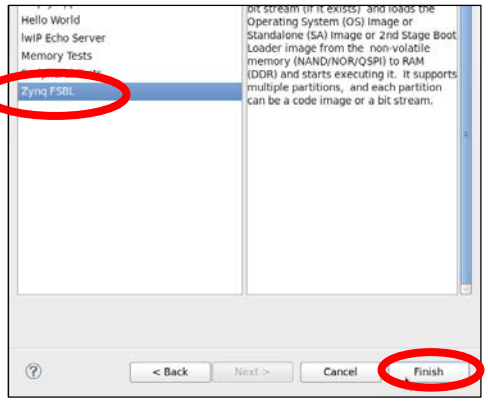


4. Type in a project name (zybo_fsbl or zynq_fsbl are recommended). Under the Board Support Package, make sure Create New is selected. Click Next

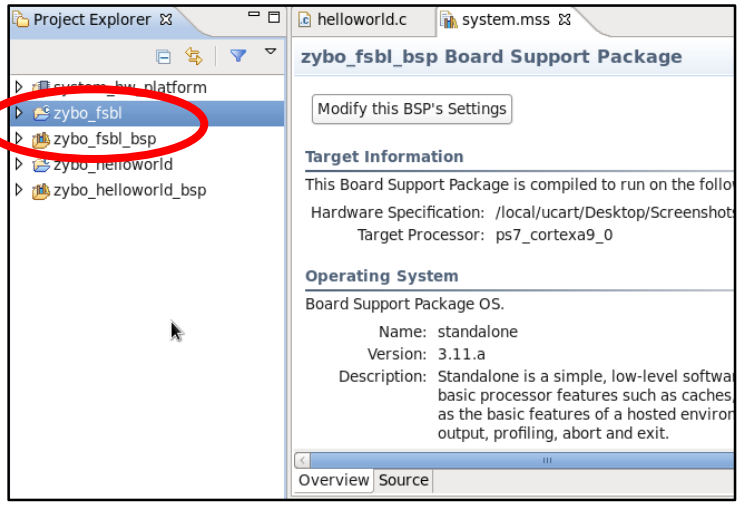


[Back to Top]

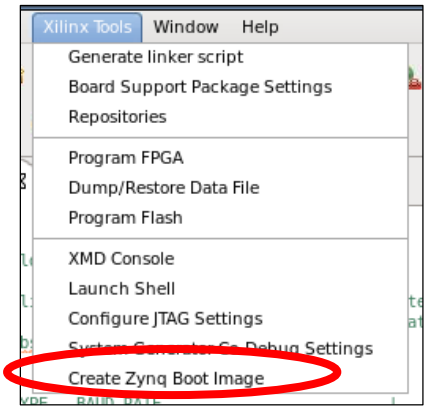
5. Select Zynq FSBL as a template (FSBL = First Stage Bootloader). Click Finish



6. You should now see the project and BSP added to the Project Explorer

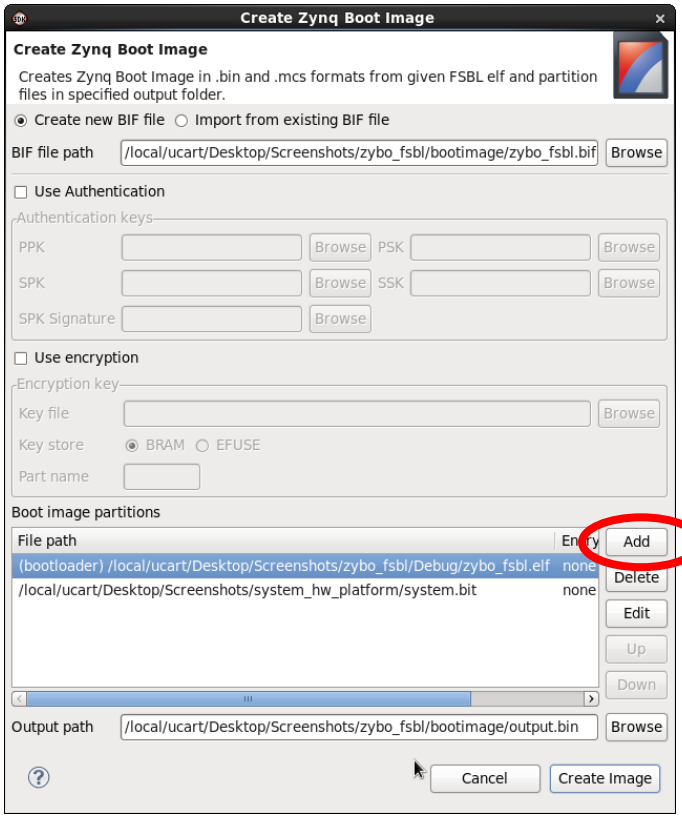


7. Make sure the zybo_fsbl project is selected and click Xilinx Tools -> Create Zynq Boot Image

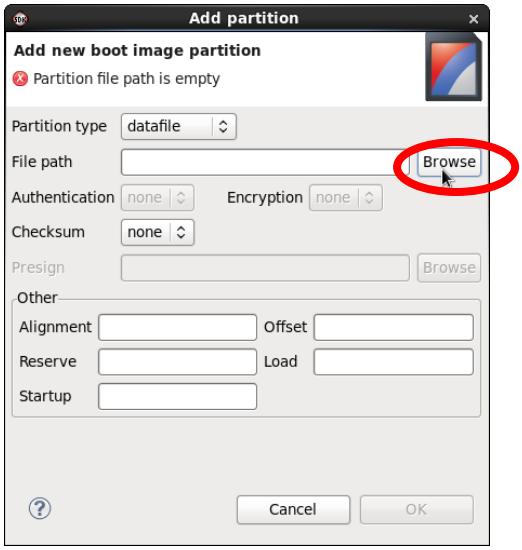


[Back to Top]

8. You should now see the "Create Zynq Boot Image" dialog box and most of the details filled in. It's important to notice the Boot image partitions and their order. Click the Add button

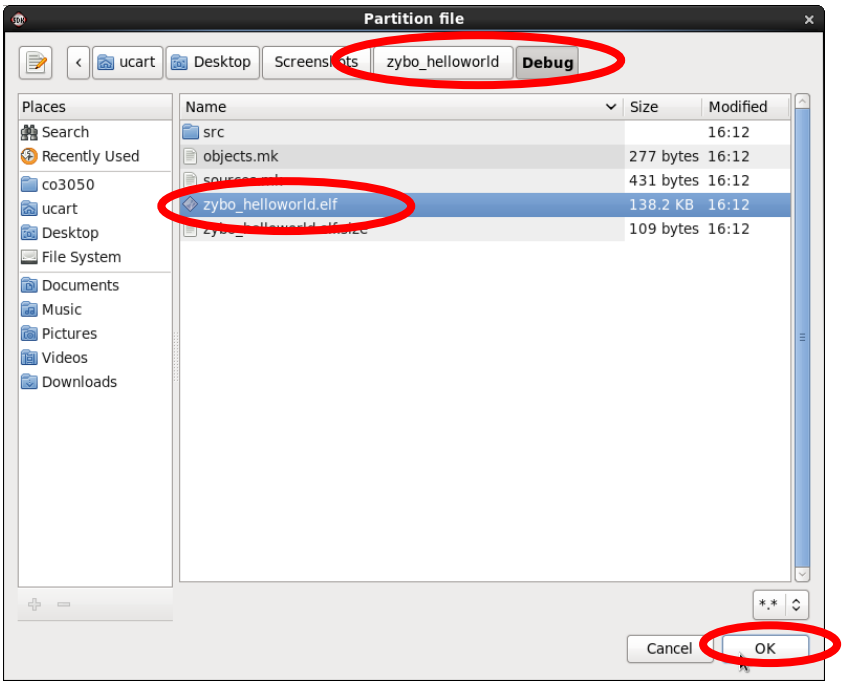


9. A small partition window will appear. Ensure the type is a datafile and click Browse

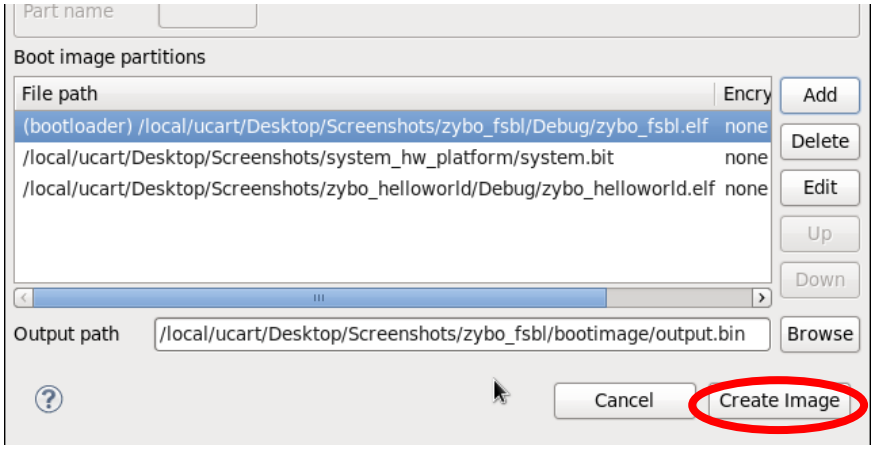


[Back to Top]

10. Navigate to the ELF file location of the application (usually located in the Debug folder of the Application project). Click Ok and Ok again to add the partition.



11. Review the updated partitions list. Take note of the Output Path location and click Create Image



NOTE: The order should always appear in the following order:

1. Zybo_FSBL.elf
2. System.bit
3. Application.elf

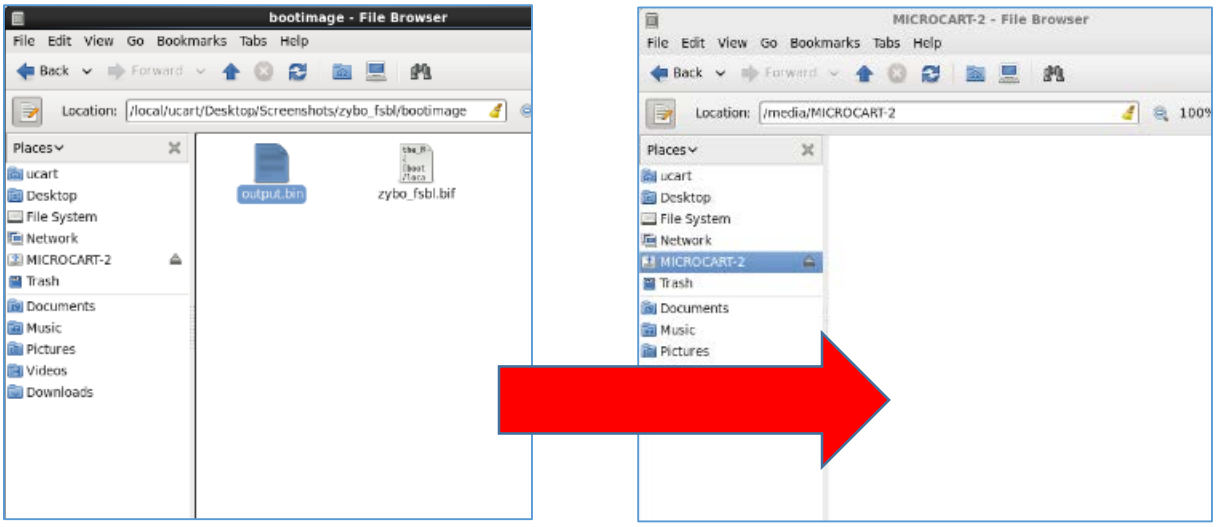
[Back to Top]

12. Congrats! You've now created a binary file that can be used on the Zybo. However, we still need to get the file onto a microSD and into the board itself.

13. Grab a microSD card (size doesn't matter) and plug it into an adapter (a USB adapter as shown for example)

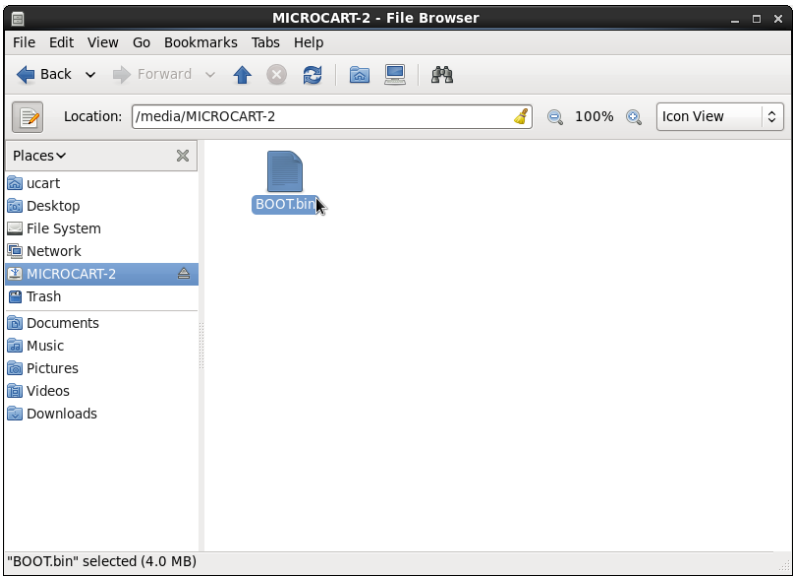
14. Plug the adapter into your current working machine. Open a file browser and navigate to your application project (i.e. zybo_helloworld) and to `zybo_fsbl/bootimage/output.bin`

15. Once located, put the output.bin file onto the SD card

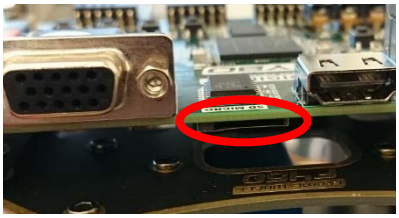


[Back to Top]

16. Rename the output.bin to BOOT.bin. This **MUST** be done since the Zybo will only read a file named BOOT.bin to configure the FPGA and start the program



17. Safely eject the microSD adapter and take the microSD card over to the Zybo board. The microSD card slot on the board can be seen below as well as the proper way to insert the card into the board



18. Once the card is in, simply turn ON the board and you should see the green Done LED come on (indicating the FPGA was configured successfully) and the program will start executing a few milliseconds after

[Back to Top]

8. FAQ

Q: How do I get Linux on the Zybo, or how do I launch from an (micro)SD card?

A: Excellent question! You'll need to create a new application project and select the FSBL template. This is a First Stage Bootloader program that will allow us to program the FPGA from the SD card and then it'll handle over the reigns to your software program (considering you put it inside the project). The walkthrough for this kind of setup may or may not be in this guide. If it is not, there are plenty of resources online to help, or ask Dr. Zambreno for the proper 488 lab guide that will explain how to do it.

Q: Some C libraries aren't working or aren't declared?

A: The Zybo has no real operating system that may handle the date or processes that we often take for granted. If you run into an issue that an operating system would be great for, you may want to look into putting Linux on the Zybo since it comes with some board drives and eases the pain of bare metal operations. Otherwise, look for alternative means that Xilinx may provide for us.

Q: Whenever I export an XPS project XSDK and clean the project, why doesn't the BSP have my newly added cores or the latest up-to-date addresses for cores?

A: Xilinx tools are complex beasts that have yet to be fully mastered. My best advice is to remove both the system_hw_platform and any BSPs in the XSDK project, and re-export the project from XPS (once the project has been synthesized once, this process is extremely quick). A new system_hw_platform should appear in the project. Create a new BSP and all the additional cores you added should now be in there.

Q: Where can I find the master UCF file for the Zybo?

A: You can find the file in the MicroCART repository under "Xilinx_Tools/XPS_files"

Q: I have an error when synthesizing or compiling. What do I do?

A: This is bound to happen eventually. Please follow the error messages provided to debug connections or code. It's ok. It happens. Try the best you can do identify the root of the problem and walk through steps. A necessary port may not be connected or a semicolon may be missed.

Q: Who should I consult about weird problems in any of the Xilinx programs that aren't already discussed?

A: Dr. Jones and Dr. Zambreno are excellent resources for debugging problems, especially in XPS, since they have quite a bit of experience with the development kits.

[Back to Top]