MICROCART 2014-2015

Xilinx Tools (XPS, XSDK, and XISE) Setup and Walkthrough Author: Paul Gerver, Last updated: Feb 2015

Table of Contents

1.	Intr	oduction2
2.	Setu	
2.	1.	Coover 3050-11 and -122
2.	2.	ISU Remote Linux Servers (linux-X, research-x.ece.iastate.edu)2
2.	3.	User PC2
3.	Xilir	x Platform Studio (XPS)
3.	1.	Creating a new XPS project
3.	2.	Opening an XPS Project
3.	3.	Synthesizing an XPS project7
3.	4.	Enabling Bluetooth (UARTO) / Editing MIO settings in XPS8
3.	5.	Enabling I2C for Sensor Board Use12
4.	Xilir	x Software Development Kit (XSDK)14
4.	1.	Exported System Hardware Platform14
4.	2.	Creating a new Board Support Package (BSP)15
4.	3.	Creating a new Application Project16
4.	4.	Configuring JTAG
4.	5.	Launching an Application Project19
5.	Run	ning a MicroCART Task Example21
5.	1.	Zybo Board Terminal21
6.	Crea	ating a Custom AXI IP Core (Custom Logic Block)25
6.	1.	Getting Started25
6.	2.	Xilinx Integrated Software Environment (XISE)29
6.	3.	Simulating Logic with ModelSim or ISim
6.	4.	Synthesizing Logic Core
6.	5.	Integrating new core into XPS
6.	6.	Accessing Core Registers in Software
7.	Usir	ng a MicroSD Card to program the Zybo Board38
8.	FAC	

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 board¹, 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 <u>here</u> (~6GB)

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

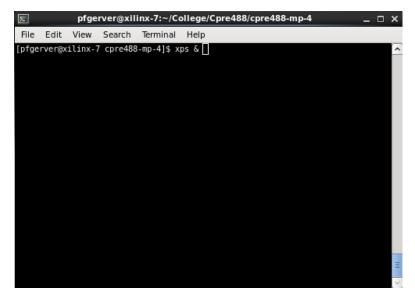
6. Ensure to install the ISE, XPS, and XSDK (The embedded package, I believe)

3. Xilinx Platform Studio (XPS)

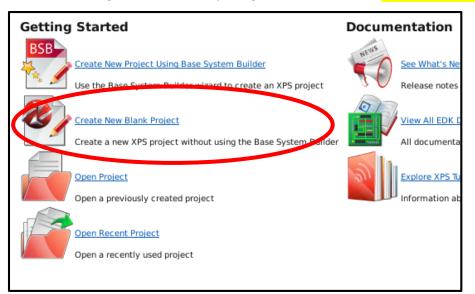
XPS is the base program for creating a hardware description file (a .bit file) that will be loaded onto the Zybo board since it contains an ARM Cortex-A9 processor and a Zynq FPGA. Without a bit file, any program that is launched on the board may not be able to communicate with all the I/O and cool peripherals 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: ${\rm xps}$ &



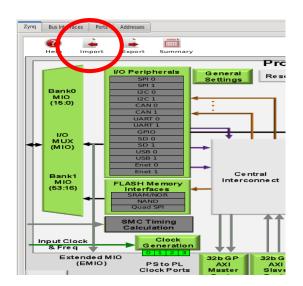
2. You should now be greeted with the opening window. Next click "Create New Blank Project"



- 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.

		<th></th>	
arget device			
<u>A</u> rchitecture	Device Size	Pac <u>k</u> age	Speed Grade
zynq	\$ xc7z010	¢ clg400	\$ -1
	File (.mhs) from existing Pro		Browse
Import Design I Set Project Peri		ath	
Import Design I Set Project Peri	pheral Repository Search Pa	ath	
Import Design I Set Project Peri	pheral Repository Search Pa ke File(instead of XPS gener	ath	Browse

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



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/"

Import Zynq Processing System Configurat	tions
Select Configuration Template	
System Template (Configurations available in the installed area) :	
ZedBoard Development Board Template ZC702 Development Board Template	
ZC706 Development Board Template	
User Template (Configurations created by User) :	
	÷
Summary of selected Configuration:	
Janmary of Science configuration.	
	OK Cancel Help

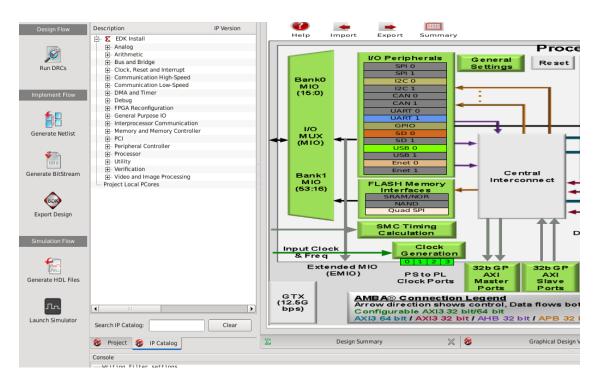
7. Once selected, click Open and click OK to start the import

? 2	Add Zynq Configuration Template File						×
Look in:	home/pfgerver/Desktop/XPS_walk	\$ 4	⇒	1	F 2		
Computer	system ZYBO_zynq_def.xml						
File name: ZYB	0_zynq_def.xml	 				<u>O</u> pen	V
Files of type: *.xm	ı	 	[•	8	<u>C</u> ance	•

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

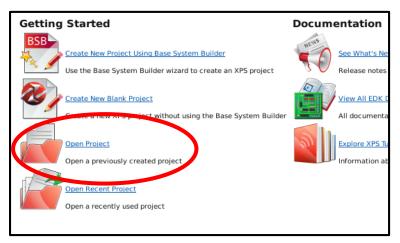


9. The default I/O controllers for the SD card, UART1 and other things will be selected and look like the following screen below.



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

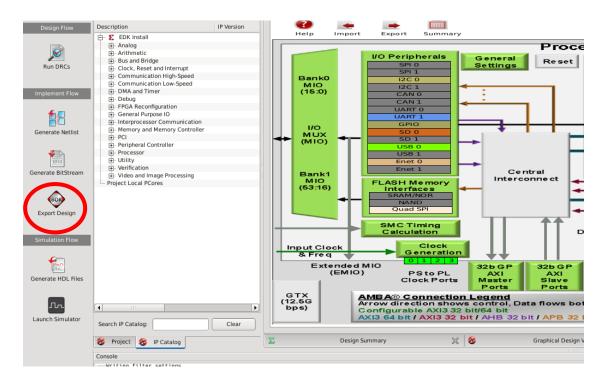


2. Navigate to the project .xmp file (i.e. task/zybo_sensor_board/system/system.xmp) and click Open

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)

	Export to SDK / Launch SDK	х
	nis dialog allows you to export hardware atform information to be used in SDK.	
✓ Includ	ie bitstream and BMM file	
	regenerate bitstream if necessary, y take some time to finish.)	
Directory	location for hardware description files	
/home/pf	fgerver/Desktop/XPS_walk/system/SDK/SDK_Export	
	ielp Export & Launch SDK Cancel Export Only	

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.

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

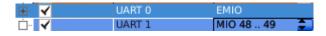
By default, UART1 is automatically configured for us when we imported the base Zybo definition file. UART1 is used on the board for receiving JTAG data and uses it as stdout and stdin for the board to communicate with the computer, when running a program. We will be enabling another UART, UART0, which 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 UARTO communication routing for a BT2PMOD chip through the Zybo's PMOD-B port.

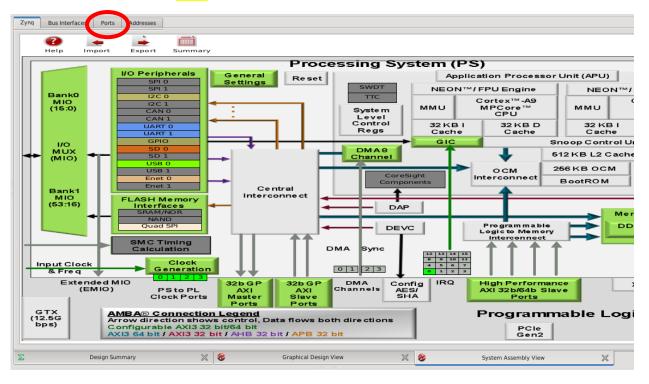
1. In XPS System View, click on the green I/O Peripherals box to bring up MIO Configuration box.

In I -	Devlahensi	10	10	Peripheral	Signal
hable	Peripheral	10	MIO 0	IGPIO	gpio[0]
<	Quad SPI Flash	MIO 1 6	MIO 1	Quad SPI Flash	gspi0 ss b
-	SRAM/NOR Flash	<select></select>	MIO 2	Quad SPI Flash	qspi0_io[0]
	NAND Flash	<select></select>	MIO 3	Quad SPI Flash	qspi0_io[1]
~	Enet 0	<select> MIO 16 27 <select> MIO 28 39 <select></select></select></select>	MIO 4	Quad SPI Flash	qspi0_io[2]
• 🖌	Enet 1	<select></select>	MIO 5	Quad SPI Flash	gspi0_io[3]
√	USB 0	MIO 28 39 🌻	MIO 6	Quad SPI Flash	qspi0_sclk
	USB 1	<select></select>	MIO 7	GPIO	gpio[7]
<	SD 0	MIO 40 45	MIO 8	Quad SPI Flash	gspi fbclk
	SD 1		MIO 9	GPIO	gpio[9]
	UART 0	<select></select>	MIO 10	GPIO	gpio[10]
 ✓ 	UART 1	MIO 48 49 EMIO <select></select>	MIO 10	GPIO	gpio[11]
. 🖌	12C 0	EMIO	MIO 12	GPIO	gpio[12]
	I2C 1	<select></select>	MIO 12 MIO 13	GPIO	gpio[12]
	SPI 0	<select></select>	MIO 14	GPIO	gpio[14]
	SPI 1	<select></select>	MIO 15	GPIO	gpio[14]
	CAN 0	<select></select>	MIO 16	Enet 0	tx clk
	CAN 1	<select></select>	MIO 17	Enet 0	txd[0]
	Trace	<select></select>	MIO 17	Enet 0	txd[1]
	Timer 0	<select></select>	MIO 18	Enet 0	txd[2]
	Timer 1	<select></select>	MIO 19 MIO 20	Enet 0	txd[3]
	Watchdog		MIO 20 MIO 21	Enet 0	tx_ctl
	PJTAG	<select></select>	MIO 21 MIO 22	Enet 0	rx clk
<	GPIO		MIO 22 MIO 23	Enet 0	rx_cik rxd[0]
			MIO 23		rxd[1]
			MIO 24 MIO 25	Enet 0 Enet 0	
			MIO 25 MIO 26		rxd[2]
			MIO 26 MIO 27	Enet 0	rxd[3] rx ctl
			MIO 27 MIO 28	Enet 0 USB 0	data[4]
			MIO 29	USB 0	dir
			MIO 30	USB 0	stp
			MIO 31	USB 0	nxt
			MIO 32	USB 0	data[0]
			MIO 33	USB 0	data[1]
			MIO 34	USB 0	data[2]
			MIO 35	USB 0	data[3]
			MIO 36	LISB 0	clk

2. Enable the UARTO 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



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

Zynq Bus Interfaces Ports	Addresses		
ame	Connected Port	Di	ire
FCLK CLK2		10	
FCLK CLK1		70	
- FCLK CLK0		10	
- FCLK CLKTRIG3 N		/ I	
- FCLK CLKTRIG2 N		I	
FCLK CLKTRIG1 N		7 I	
FCLK CLKTRIGO N		/ I	
FCLK RESET3 N		70	
FCLK RESET2 N		/ 0	
FCLK RESETI N		10	
FCLK RESETO N		/ 0	
IRQ_F2P	L to H: No Connection	Ī	
Core0 nFIQ		I	
- Core0 nIRQ		I	
Core1_nFIQ		I	
- Core1 nIRQ		I	
IRQ_P2F_QSPI		/ 0	
IRQ_P2F_GPIO		10	
IRQ_P2F_USB0		/ 0	
IRQ_P2F_ENET0		70	
		0	
···· IRQ_P2F_SDI00		10	
IRQ_P2F_I2C0		/ 0	
···· IRQ_P2F_UART0		10	
IRQ_P2F_UART1			
(IO_IF) MEMORY_0	Connected to External Ports	۲.	
(IO_IF) PS_REQUIRED_EXTERNAL_IO	Connected to External Ports		
	Connected to External Ports	٦	
···· UART0_TX	External Ports::processing_system7_0_UART0_TX_pin	/ 0	
UARTO_RX		I	
	Not connected to External Ports	٢	
± ·· (IO_IF) USBIND_0	Not connected to External Ports	٦	
clock_generator_0			
- reset 0			

- Ð **1** Port Filters >> By Interface Direction Class F ▲ Range Filters • • • • • • • • • • • • • V BUS 10 NONE / 0 ✓ 10 NONE 10 By Connection NONE / 1 Defaults Check This CLK 71 onnected NONE Unconnected / 1 NONE By Class Clocks Only / 0 CLK Clocks CLK Resets Only CLK Resets CLK Interrupts Only Interrupts Others By Direction Inputs RST Outputs / 0 RST ✓ InOuts / 0 RST 70 RST INTERRUPT Ι INTERRUPT INCOME DALLER
- 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 UARTO_TX and UARTO_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

lame	Connected Port	Direction	Ra	
External Ports				
CLK_N	clock_generator_0::CLKIN	/ I	۲	
···· CLK_P	clock_generator_0::CLKIN	7 I		
processing_system7_0_DDR_Addr	processing_system7_0::[MEMORY_0]::DDR_Addr	/ 10	(1)	
… processing_system7_0_DDR_BankAddr	processing_system7_0::[MEMORY_0]::DDR_BankAddr	7 10	12	
processing_system7_0_DDR_CAS_n	processing_system7_0::[MEMORY_0]::DDR_CAS_n	/ 10	۲	
— processing_system7_0_DDR_CKE	processing_system7_0::[MEMORY_0]::DDR_CKE	7 10		
processing_system7_0_DDR_CS_n	processing_system7_0::[MEMORY_0]::DDR_CS_n	/ 10		
processing_system7_0_DDR_Clk	processing_system7_0::[MEMORY_0]::DDR_Clk	7 10		
processing_system7_0_DDR_Clk_n	processing_system7_0::[MEMORY_0]::DDR_Clk_n	/ 10		
processing_system7_0_DDR_DM	processing_system7_0::[MEMORY_0]::DDR_DM	7 10	- Ē <i>l</i>	
processing_system7_0_DDR_DQ	processing_system7_0::[MEMORY_0]::DDR_DQ	/ 10	1	
processing_system7_0_DDR_DQS	processing_system7_0::[MEMORY_0]::DDR_DQS	7 10		
processing system7 0 DDR DQS n	processing_system7_0::[MEMORY_0]::DDR_DQS_n	/ 10	I	
processing_system7_0_DDR_DRSTB	processing_system7_0::[MEMORY_0]::DDR_DRSTB	7 10		
— processing_system7_0_DDR_ODT	processing_system7_0::[MEMORY_0]::DDR_ODT	/ 10	۲	
processing_system7_0_DDR_RAS_n	processing_system7_0::[MEMORY_0]::DDR_RAS_n	7 10		
processing_system7_0_DDR_VRN	processing_system7_0::[MEMORY_0]::DDR_VRN	/ 10	(
processing_system7_0_DDR_VRP	processing_system7_0::[MEMORY_0]::DDR_VRP	7 10		
processing_system7_0_DDR_WEB_pin	processing_system7_0::[MEMORY_0]::DDR_WEB	/ 0	۲	
processing_system7_0_MIO	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::MIO	7 10	- Ē	
processing_system7_0_PS_CLK_pin	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::PS_CLK	/ I	Test	
processing_system7_0_PS_PORB_pin	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::PS_PORB	71		
processing_system7_0_PS_SRSTB_pin	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::PS_SRSTB	/ I		
processing_system7_0_UART0_RX_pin	processing_system7_0::[UART_0]::UART0_RX	7 I	٦	
processing_system7_0_UART0_TX_pin	processing_system7_0::[UART_0]::UART0_TX	/ 0	•	

- 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 <u>Project Platform</u> 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)

Project	+00
Platform	
Project Files	
 MHS File: system.mhs 	
 UCF File: data/system.ucf 	all and a set of
···· IMPACT Command File: etc/dov	
 Implementation Options File: e Bitgen Options File: etc/bitgen. 	
Elf Files	ul
- Project Options	
Device: xc7z010clg400-1	
Netlist: TopLevel	
Implementation: XPS (Xflow)	
HDL: VHDL	
Sim Model: BEHAVIORAL	
Design Summary	

 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.



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)

- 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)

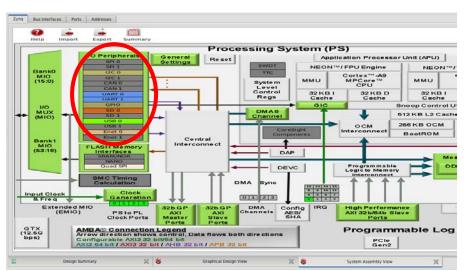
FORWARNING: The exporting process can take several minutes to complete or over an hour depending on machine load and how much logic needs to programmed.

Export to SDK / Launch SDK	×						
This dialog allows you to export hardware platform information to be used in SDK.							
✓ Include bitstream and BMM file							
(XPS will regenerate bitstream if necessary, and it may take some time to finish.)							
Directory location for hardware description files							
/home/pfgerver/Desktop/XPS_walk/system/SDK/SDK_Export							
Help Export & Launch SDK]						

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.



- 2. VERY IMPORTANT Check the Show I/O Standard Options
- 3. Chance I2CO's IO to MIO 10..11, set LVCMOS 3.3V for both Bank 0 and 1, **ENABLE** Pullup on MIO 10 and 11

nq PS Confi			MIO Configura	tandard Options	Bank 0 IO Voltage:	LVCMOS 3.3V	Bank 1 IO Voltage:	LVCMOS 3.3V
able	Peripheral	10	10	Peripheral	Signet	IO Type	Speed	Pullup
· 🖌	Quad SPI Flash	MIO 1 6	MIO 0	GPIO	gpio[0]	LVCMOS 3.3V	🗘 Islow 🗘	disabled
-	SRAM/NOR Flash	<select></select>	MIO 1	Quad SPI Flash	qspi0_ss_b	LVCMOS 3.3V	🗘 fast 🗘	disabled
-	NAND Flash	<select></select>	MIO 2	Quad SPI Flash	qspi0_io[0]	LVCMOS 3.3V	🗘 fast 🗘	disabled
- 🖌	Enet 0	MIO 16 27 📮	MIO 3	Quad SPI Flash	qspi0_io[1]	LVCMOS 3.3V	🗘 fast 🗘	disabled
	Enet 1	<select></select>	MIO 4	Quad SPI Flash	qspi0_io[2]	LVCMOS 3.3V	🗘 fast 🗘	disabled
×	USB 0	MIO 28 39 🗘	MIO 5	Quad SPI Flash	qspi0_io[3]	LVCMOS 3.3V	🗘 fast 🗘	disabled
	USB 1	<select></select>	MIO 6	Quad SPI Flash	qspi0_sclk	LVCMOS 3.3V	🗘 fast 🗘	disabled
· 🖌	SD 0	MIO 40 45 🗘	MIO 7	GPIO	gpio[7]	LVCMOS 3.3V	韋 slow 📮	disabled
	SD 1	<select></select>	MIO 8	Quad SPI Flash	qspi_fbclk	LVCMOS 3.3V	🗘 fast 🗘	disabled
	UART 0	<select></select>	MIO 9	0.10	[gpio[9]	LVCMOS 3.3V	SIOW	disabled
×		10 40	MIC 10	12C 0	scl	LVCMOS 3.3V	🗘 slow 🗘	enabled
V	12C 0	MIO 10 11 🗘	MIO 11	12C 0	sda	LVCMOS 3.3V	🔷 slow 🗘	enabled
	12C 1	<select></select>	1.0.12	GPIO	gpio[12]	LVCMOS 3.3V	🗘 slow 🗘	disabled
	SPLO	<select></select>	MIO 13	GHO		WOMOG 2 PM		disabled
	SPI 1	<select></select>	MIO 14	GPIO	gpio[14]	LVCMOS 3.3V	slow 🗘	disabled
	CAN 0	<select></select>	MIO 15	GPIO	gpio[15]	LVCMOS 3.3V	🗘 slow 🗘	disabled
	CAN 1	<select></select>	MIO 16	Enet 0	tx_clk	LVCMOS 3.3V	🛊 fast 🛊	disabled
	Trace	<select></select>	MIO 17	Enet 0	txd[0]	LVCMOS 3.3V	🗘 fast 🗘	disabled
	Timer 0	<select></select>	MIO 18	Enet 0	txd[1]	LVCMOS 3.3V	🗘 fast 🗘	disabled
	Timer 1	<select></select>	MIO 19	Enet 0	txd[2]	LVCMOS 3.3V	🗘 fast 🗘	disabled
	Watchdog	<select></select>	MIO 20	Enet 0	txd[3]	LVCMOS 3.3V	🗘 fast 🗘	disabled
	PJTAG	<select></select>	MIO 21	Enet 0	tx_ctl	LVCMOS 3.3V	🗘 fast 🗘	disabled
~	GPIO		MIO 22	Enet 0	rx_clk	LVCMOS 3.3V	🛊 fast 🗘	disabled
			MIO 23	Enet 0	rxd[0]	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 24	Enet 0	rxd[1]	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 25	Enet 0	rxd[2]	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 26	Enet 0	rxd[3]	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 27	Enet 0	rx_ctl	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 28	USB 0	data[4]	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 29	USB 0	dir	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 30	USB 0	stp	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 31	USB 0	nxt	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 32	USB 0	data[0]	LVCMOS 3.3V	🗘 fast 🗘	disabled
			MIO 33	USB 0	data[1]			disabled
			MIO 34	USB 0	data[2]			disabled
			4					•

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 UARTO configuration.

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

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.

Workspace Launcher	×
Select a workspace	
Xilinx SDK stores your projects in a folder called a workspace. Choose a workspace folder to use for this session.	
Workspace: //home/pfgerver/Desktop/XPS_walk/sw	Browse
Use this as the default and do not ask again NO!!!	
	Cancel OK

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

10	C/C++ - system_hw_platform/system.xml - Xilinx
File Edit Source Refactor Navigate	Search Run Project Xilinx Tools Window Help
) 	੶] @ ~ @ ~ ፪ ~ @ ~] ☆ ~ Q ~ Q ~] ≈] □] # ⊠] &] ⊘ / ~] □
Project Explorer 🕱 🗖	system.xml 🛛
E 😫 🔻 '	system_hw_platform Hardware Platform Specification
	Design Information
s7_init.c	Target FPGA Device: xc7z010
⊫ ps7_init.h	Created With: EDK 14.7
ps7_init.html	Created On: Mon Dec 1 17:58:55 2014
ps7_init.tcl	XPS Design Report: file:///home/pfgerver/Desktop/XPS_walk/system/SDK/SDK_Export/hw/system.html
i system.bit	
system.xml	Address Map for processor ps7_cortexa9_0
	ps7 uart 0 0xe0000000 0xe0000fff
	ps7 uart 1 0xe0001000 0xe0001ff
	ps7 l2c 0 0xe0004000 0xe0004fff
	ps7 afi 0 0xf8008000 0xf8008fff
	ps7 afi 1 0xf8009000 0xf8009fff
	ps7_afi_2_0xf800a000_0xf800afff
	ps7_aff_3 0xf800b000 0xf800bfff
	ps7_sd_0 0xe0100000 0xe0100fff
	ps7_ethernet_0_0xe000b000_0xe000bfff
	ps7_usb_0_0xe0002000_0xe0002fff
	ps7_qspi_0_0xe000d000_0xe000dfff ps7_qspi linear 0_0xfc000000_0xfcffffff
	ps/_dsp/_intear_0_extensions_extension
	ps7 qpio 0 0xe000a000 0xe00afff
	ps7 ddr 0 0xf8006000 0xf8006fff
	ps7 dev cfg 0 0xf8007000 0xf80070ff
	ps7_xadc_0_0xf8007100_0xf8007120
	pc7 complate comp 0 nuf9000000 nuf900ffff

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

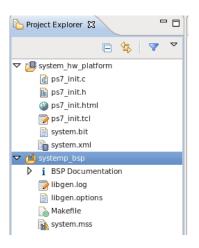
roject name:	systen	n_bsp	
🗸 Use defau	lt locati	ion	
ocation: //ho	me/pfg	gerver/Desktop/XPS_walk/sw/system_bsp	Browse
Cho	ose file	system: default 🗘	
Target Hardw	are		
Hardware Pla	tform:	system_hw_platform	0
	CPU:	ps7_cortexa9_0	0
Board Suppor	t Packa	age OS	
standalone		Standalone is a simple, low-level software layer. It provides access processor features such as caches, interrupts and exceptions as we basic features of a hosted environment, such as standard input an profiling, abort and exit.	ell as the

IMPORTANT: If UARTO 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

		Board Support Pa	ickage settings		
oard Support Package Se	ttings				n.
ontrol various settings of you	ur Board Support Package.				
Overview					
orenten					
standalone	Configuration for OS: standa	lone			
	Configuration for OS: standa	lone	Default	Туре	Description
standalone	-	lone ps7_uart_0	Default	Type peripheral	Description stdin peripheral
standalone	Name				
standalone	Name stdin	ps7_uart_0	none	peripheral	stdin peripheral

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

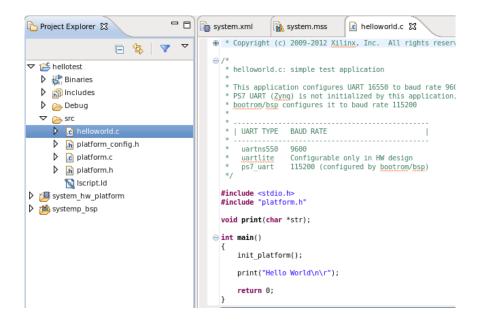
2. Enter a project name, and select the "Use existing" radio button for the BSP. We want to use our newly created one rather than create one.

•	New Project X
Application Project Create a managed make	application project.
Project nr 📭 hellotest	
 Use default location 	
Location: //home/pfger	ver/Desktop/XPS_walk/sw/hellotest Browse
Choose file sy	stem: default \$
Target Hardware	
Hardware Platform sy	/stem_hw_platform
Processor p	s7_cortexa9_0
Target Software	
OS Platform	standalone
Language	
Board Support Package	e O Create New hellotest bsp
<	Use existing systemp_bsp
?	< Back Next > Cancel Finish

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

®	New P	roject		×
Templates				-
Create one of the available temp application project.	plates to gener	ate a fully-functi	ioning	
Available Templates:				
Dhrystone		Let's say 'Hello	World' in C.	<u>^</u>
Empty Application				
Hello World				
IwIP Echo Server				
Memory Tests				
Peripheral Tests				
Zynq FSBL				
				=
				\sim
0	< Back	Next >	Cancel	Finish

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)



3. Click OK

hese settings a	affect how XMD connects	to the FPGA.		-
Type:	Auto Detect			
Hostname:				
Port:				
Frequency:				
Other Options:				
-	ain ally Discover Devices on J nfiguration of JTAG Chain			
Automatica	ally Discover Devices on J		2 i i	
 Automatica Manual Col 	ally Discover Devices on J		• 副 章 등 :	Q.
 Automatica Manual Col 	ally Discover Devices on J nfiguration of JTAG Chain			Ç.
 Automatica Manual Col 	ally Discover Devices on J nfiguration of JTAG Chain			Q.
 Automatica Manual Col 	ally Discover Devices on J nfiguration of JTAG Chain			Q.
 Automatica Manual Col 	ally Discover Devices on J nfiguration of JTAG Chain			Q.

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

<u>sop</u>	Program FPGA X
Program FI Specify the	bitstream and the ELF files that reside in BRAM memory
-Hardware C	onfiguration
Hardware S	pecification: /home/pfgerver/Desktop/XPS_walk/sw/system_hw_platform/system.xml
Bitstream:	system.bit Search Browse
BMM File:	Search Browse
Software Co	onfiguration
Processor	ELF File to Initialize in Block RAM
?	Cancel Program

- 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.



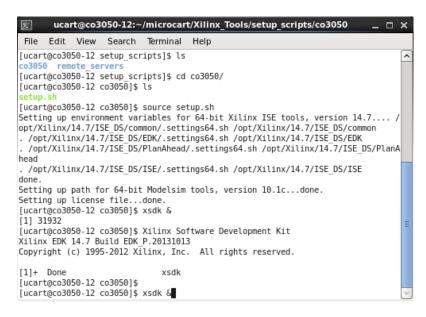
4. View the data received from the board through the COM terminal.

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.

®	Workspace Launcher	×
Select a wor	kspace	
	ores your projects in a folder called a workspace. (space folder to use for this session.	
Workspace:	local/ucart/microcart/tasks/zybo_terminal/sw	Browse
Use this a	s the default and do not ask again	
		Cancel OK

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

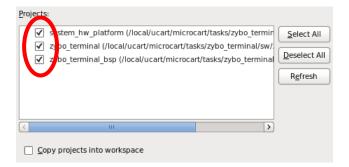
5. In the Import window, go to Existing Projects into Workspace under the General folder. Click Next.

n Import	×
Select Create new projects from an archive file or directory.	¥1
Select an import source:	
type filter text	4
Ceneral Archive Fit Existing Projects into Workspace Pite option	
Preferences C/C++ C/C++ Comparison Remote Systems Remote Systems Remote Systems Comparison Compari	
 Ruin bebug Team 	
? < Back Next > Cancel Fin	nish

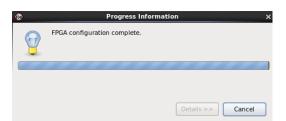
6. Choose the Select root directory and click Browse... Once again, navigate to "tasks/zybo_terminal/sw"

laces	Name	Size	Modified	
🖞 Search	system_hw_platform		11/20/2014	
Recently Used	🚞 zybo_terminal		11/20/2014	
co3050	🛅 zybo_terminal_bsp		11/20/2014	
a ucart	SDK.log	0 bytes	15:08	
🔄 Desktop				
File System				
Documents	-			
Music				
Pictures				
Videos				
Downloads				
				=
÷ —				

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



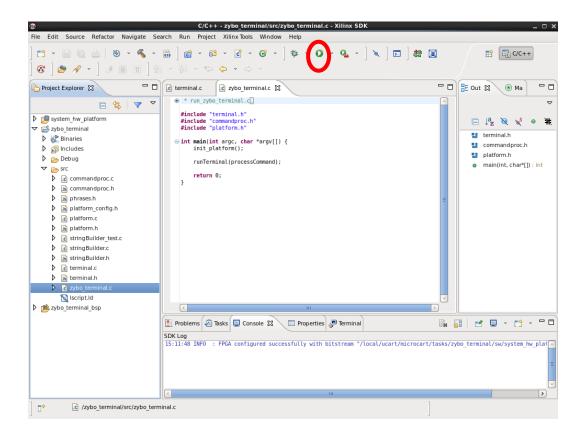
- 8. Click Finish to import the project into the workspace.
- 9. Next, configure the JTAG appropriately (Auto Detect should be adequate). Also, plug in the Zybo board if you haven't yet.
- 10. Program the FPGA by going to Xilinx Tools -> Program FPGA.



- 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:



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

Σ				[screen	n 0: ttyUSB1] _	
File	Edit	View	Search	Terminal	Help	
	ZYBOT	ERM (TI	1) =====			~
Welco \$> he						
\$> wh I am \$> le [Usin	d g defa ing LE	you ffin ma ult co	an unt of 64	1]		=

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 <u>environment setup</u>, start XPS
- 2. <u>Create</u> or <u>open</u> 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

	Create and Import Peripheral Wizard	
eripheral Flow Indicate if you want to create a new per	pheral or import an existing peripheral.	
his tool will help you create templates for a irrectory structures required by EDK will be Create Tomplates	new EDX IP, or help you import an existing EDX IP into an XPS project or EDX repository. The interfact generated. Sole(1). Sole(1). A Create genplates for a new peripheral Import generate	e files an
Implement/Verify	Flow description This tool will create HDL templates that have the EDK compliant portparameter interface need to implement the body of the peripheral.	You will
	Options Load an existing .cip settings file (saved from a previous session)	se
More Info	< Back Next >	Cance



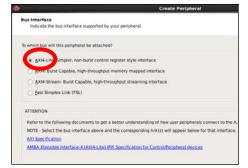
5. Ensure "To an XPS Project" button is selected. Click Next

🎊 Create Peripheral	×
Repository or Project Indicate where you want to store the new peripheral.	\$ \$
A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK reposite projects.	bry, the peripheral can be accessed by multiple XPS
\bigcirc To an EDK user repository (Any directory outside of your EDK installation path)	
Repository:	✓ Bro <u>w</u> se
To an <u>X</u> PS project	
Project: Project Proje	✓ Browse
Peripheral will be placed under:	
/local/ucart/microcart/itasks/zybo_spi//system/pcores	
More info	< <u>B</u> ack <u>N</u> ext > Cancel

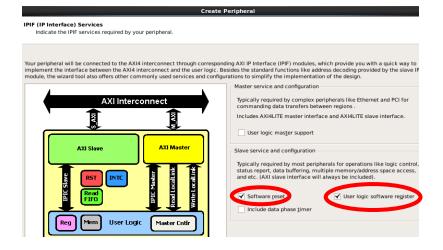
- 6. Name the custom core something unique. It is recommended to name it "axi_<name>" because the Zybo board uses AXI buses to communicate data. For our case, we'll use "axi_pwm" because we're creating a core that can generate PWMs.
- 7. Latsly, add a description if desired and click Next

Create Peripheral	
Name and Version Indicate the name and version of your peripheral.	2
Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design ent	ity.
Name: axi_pwm	
Version: 1.00.a	
Major revision: Minor revision: Hardware/Software compatibility revision:	
Description: PWM Generator for Quad Motors	
Logical library name: axi_pwm_v1_00_a	
All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compile above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripher repositories indicated in the XPS project settings.	
More Info Seck	<u>N</u> ext > Cancel

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



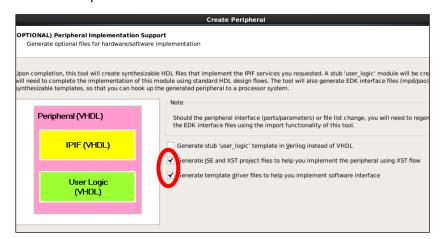
9. Make sure "Software reset" and "User logic software register" are checked. Click Next



 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 <u>twice</u> because the default IPICs are fine

ser	ic. These registers are addressable on the byte, half word, i er read/write will be included in the user-logic module gene r logic software registers may take full advantage of the sl
are programs to control and to monitor the status of your user log boundaries depending on your design. An example logic for regist eference.	ic. These registers are addressable on the byte, half-word, er read/write will be included in the user-logic module gen r logic software registers may take full advantage of the s
Bus2IP_WReq Reg 0	rice to generate CE decodes for all of the individual regis he left shows the simplest set of IPIC slave signals to rea nber of software accessible registers: 2

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.

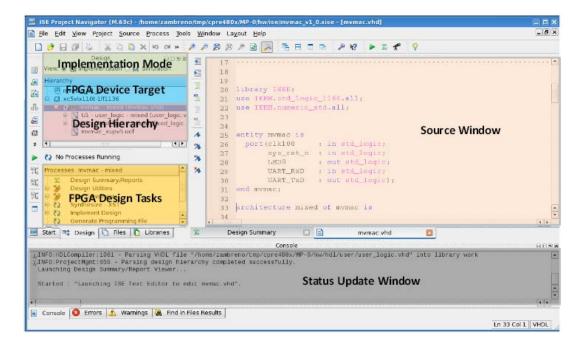
2	Create Peripheral X
	Congratulations!
	When you click Finish, HDL files representing your peripheral will be generated. You will have to implement the functionality of your peripheral in the stub 'user_logic' template file.
	IMPORTANT: If you make any interface changes to the generated peripheral (including peripheral name, version, ports and parameters), or any file changes (add or remove files), you will need to regenerate the EDK interface files by using this tool in the Import mode.
	Thank you for using Create and Import Peripheral Wizard! Please find your peripheral hardware templates under /local/ucart/microcart/tasks/zybo_spi/system/pcores/axi_pwm_v1_00_a and peripheral software templates under /local/ucart/microcart/tasks/zybo_spi/system/drivers/axi_pwm_v1_00_a respectively.
	Peripheral Summary:
	top name : axi_pwm version : 1.00.a type : AXI4LITE slave features : slave attachment
1	mir/rst register user s/w registers
i i	Address Block Summary:
	user logic slv : C_BASEADDR + 0x00000000 : C_BASEADDR + 0x000000FF soft reset : C_BASEADDR + 0x00000160 : C_BASEADDR + 0x000001FF
	File Summary
	- HDL source - /local/ucart/microcart/tasks/zybo_spi/system/pcores/axi_pwm_v1_00_a/hdl top entity : vhdl/axi_pwm.vhd
	NOTE: A *.cip settings file will be created under your peripheral's "dev!" folder. It can be loaded in a future wizard session to regenerate your peripheral.
	Click Finish to generate your peripheral.
More Info	< Back Finish Cancel

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:

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)

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

<u>></u>		
E 🗐	ile <u>E</u> dit <u>V</u> iew P <u>r</u> oject <u>S</u> ource <u>P</u> rocess <u>T</u> ools <u>W</u> indow	Layout
8 🖸		» 🛛 🏓
Desi		
T	View: 🕘 🎒 Implementation 🔘 🎆 Simulation	
-	Hierarchy	
8	🦳 🧑 axi_pwm	_
8	xc7z010-1clg400 The state of the st	Ξ
—	axi_pwm - IMP (axi_pwm.vhd)	2
00	AXI_LITE_IPIF_I - axi_lite_ipif - imp (axi_lite_	Ξ
-	SOFT_RESET_I - soft_reset - implementation	
-	🔤 🙀 USER_LOGIC_I - user_logic - IMP (user_logic	12
275		_
£2		1
		74
—		
		74
		34
		_
		0
-	• • • •	

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

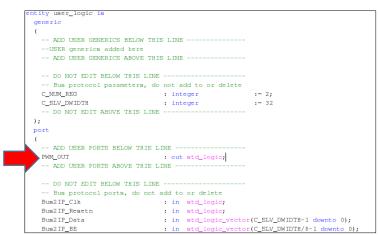
In general, there will be three main things you will need to edit.

- 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 (<u>http://class.ece.iastate.edu/cpre488/labs/MP-1.pdf</u>)

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

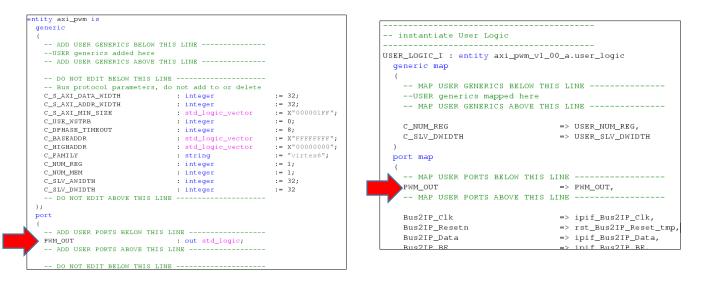
7. In USER_LOGIC_I, add the variable, direction, and type to the entity. We only want one signal out for our PWM



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

<u>Changes also need to be made in the user logic</u> instance within axi pwm



- 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

11. Scroll to the Ports section and add the following: PORT PWM_OUT = "", DIR = O

(If the port is an input put DIR = I)

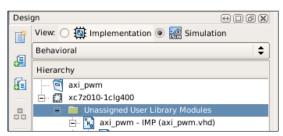
tt Po	orts
PORT	S_AXI_ACLK - "", DIR - I, SIGIS - CLK, BUS - S_AXI
PORT	S_AXI_ARESETN - ARESETN, DIR - I, SIGIS - RST, BUS - S_AXI
PORT	S_AXI_AWADDR - AWADDR, DIR - I, VEC - [(C_S_AXI_ADDR_WIDTH-1):0], ENDIAN - LITTLE, BUS - S_AXI
PORT	S_AXI_AWVALID - AWVALID, DIR - I, BUS - S_AXI
PORT	S_AXI_WDATA - WDATA, DIR - I, VEC - [(C_S_AXI_DATA_WIDTH-1):0], ENDIAN - LITTLE, EUS - 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 - 0, BUS - S_AXI
PORT	S_AXI_RDATA - RDATA, DIR - 0, VEC - [(C_S_AXI_DATA_WIDTH-1):0], ENDIAN - LITTLE, EUS - S_AXI
PORT	S_AXI_RRESP - RRESP, DIR - 0, VEC - [1:0], BUS - S_AXI
PORT	S_AXI_RVALID - RVALID, DIR - 0, BUS - S_AXI
PORT	S_AXI_WREADY - WREADY, DIR - 0, BUS - S_AXI
PORT	S_AXI_BRESP - BRESP, DIR - 0, VEC - [1:0], BUS - S_AXI
PORT	S_AXI_BVALID - EVALID, DIR - 0, BUS - S_AXI
PORT	S_AXI_AWREADY - AWREADY, DIR - O, BUS - S_AXI
PORT	PWM_OUT - "", DIR - C
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

Evaluation Development Board	None Specified
Product Category	All
Family	Zynq
Device	XC7Z010
Package	CLG400
Speed	-1
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE VHDL
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	
VUDL Source Analysis Standard	

- 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.

	SOFT_RESET_F SOR_reset - Implemente	-
	USER_LOGIC_I - user_logic - IMP (user_l	_
	counter_f - imp (counter_f.vhd)	*
	🖶 🙀 or_gate128 - imp (or_gate128.vhd)	~
		196
		100
		104
		10
		84
		29
	•	-
	No Processes Running	-
	-	0
The second	Processes: USER_LOGIC_I - user_logic - IMP	
ΞĘ.	🖮 😼 ModelSim Simulator	
-	Simulate Behavioral Model	

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

	1	F	ø	2	Ж	D	ß	\times	5	>
Desig	gn								000	K)
ľ	View:	۹ 🔯	🖡 Imp	lementa	ation 🤇	M (Simula	ation		
1	Hierar	chy								

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.

£1	🖮 🛄 xc7z010-1clg400	111
a =	🕂 💼 Unassigned User Library Modules	10
	🗕 🙀 axi_pwm - IMP (axi_pwm.vhd)	2
00	AXI_LITE_IPIF_I - axi_lite_ipif - imp (axi_lite_	
—	SOFT_RESET_I - soft_reset - implementatior	
6	USER_LOGIC_I - user_logic - IMP (user_logic	10
Ø		1.00
had		14
		74
		10
		24
		1 × 1
		74
		_
		~
	No Processes Running	0
	Processes: axi_pwm - IMP	
₽t	📡 Design Summary/Reports	
Ξü.	🕀 🏄 Design Utilities	
×ų	🕀 🌮 User Constraints	
	🕞 🚱 Synthesize - XST	
7	View RTL Schematic	

6.5. Integrating new core into XPS

 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"

	<u>P</u> rojec	t H <u>a</u> rdware	Device Configuration	<u>D</u> ebug	<u>S</u> imulatio
	Pi	roject <u>O</u> ptions.			
J	<u> </u>	esign Rule Che	eck	Ctrl+	+Shift+D
	S S	elect <u>E</u> lf File			
<	🧼 Е	xport Hardware	e Design to SD <u>K</u>		
6	<u>A</u>	rchive Project.			
	у <mark>л</mark> с	enerate <u>B</u> lock	Diagram Image (Obsole	ete)	
	b 0	pen <u>G</u> raphical	Design View		
4	<u> </u>	enerate and Vi	ew Design Report		
	ΣV	iew Design <u>S</u> ur	mmary		
	∳ R	un <u>V</u> ersion Mig	ration		
	3 R	escan User Rep	positories		
	📓 Li	aunch Xilinx <u>S</u> ł	nell		
:	<u>0</u>	ustomize Butto	ons		
1	d c	lean All Genera	ated <u>F</u> iles		
P	19 I	erminate Runni	ng Process		

- 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)

	Project Local PCores
	- USER
	AXI_PWM 1.00.a
	Search IP Catalog: Clear
	😵 Project 🎯 IP Catalog
	Add IP Instance to Design
?	Do you want to add one axi_pwm 1.00.a IP instance to your design
	No Ves

Component Instance Name axi_pwm_0	S Core Config - axi_pwm_0 - axi_pwm_v1_00_	a x
+ S_AXI PWM_OUT →	C_BASEADDR C_DPHASE_TIMEOUT C_HIGHADDR C_NUM_MEM C_NUM_REG C_SLV_AWIDTH C_SLV_DWIDTH C_S_AXI_ADDED_AXI_PARAMS C_S_AXI_ADDR_WIDTH C_S_AXI_ADDR_WIDTH C_S_AXI_ANI_VER C_S_AXI_MIN_SIZE C_S_AXI_PROTOCOL C_USE_WSTRB	Øxfffffff 8 Øx6000000 1 22 32 1.01.a 32 1.01.a 32 AXI4LITE 0
Show All Ports		
		OK Cancel Help

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.

ł	Instantiate and Connect IP - axi_pwm_0
	axi_pwm IP with version number 1.00.a is instantiated with name axi_pwm_0. Please make selection below.
	Select processor instance to connect to; XPS will make the Bus Interface connection, assign the address, and make IO ports external
	processing_system7_0
	User will make necessary connections and settings
	,
	ОК

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

Zynq	Bus Interfaces	Ports	Addresses		
Name				Connected Port	Direc
+ Extern	al Ports				
+ axi_int	erconnect_1				
+ proces	sing_system7_0				
+ BTNS					
+ LEDS					
+ SWS					
+ axi_lic	_0				
i axi_pw	/m_0				
	VM_OUT				/ O
+ (B	US_IF) S_AXI			Connected to	2



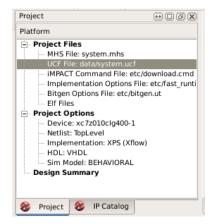
7. Since we have an external pin, we can connect it to another core if we wish, but since we're generating PWMs, let's make the pin external and connect it to a PMOD. Right-click on the PWM_OUT port and click "Make External"

⊕… axi_lic_0 ⊡… axi_pwm_0		
PWM_OUT ⊕- (BUS_IF) S_AXI	No Connection New Connection Make External	Connected to

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

Zynq Bus Interfaces Ports Address	ses
Name	Connected Port Direction Ran
External Ports	
BTNS_GPIO_IO	BTNS::[gpio_0]::/ I0 🔿 [3:0]
CLK_N	7 I 🗘
CLK_P	🖌 I 🕞
LEDS_GPIO_IO_O_pin	LEDS::[gpio_0]::
SWS_GPIO_IO	SWS::[gpio_0]::G / I0 🗐 (3:0)
axi_iic_0_Scl_pin	axi_iic_0::[iic_0]:// I0 (\$) axi_iic_0::[iic_0]:// I0 (\$)
axi_iic_0_Sda_pin	axi_iic_0::[iic_0]: 🥒 I0 🔅
axi_pwm_0_PWM_OUT_pin	axi_pwm_0::PW. 📝 O

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



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

NET axi_pwm_0_FWM_OUT_pin LOC = T20 | IOSTANDARD=LVCMOS33; #JB1

11. Finally, synthesize the XPS project

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.

Zynq Bus Interfaces Ports	Addresses			
Instance	Base Name	Base Address	High Address	Size
- processing_system7_0's Address				
processing_system7_0	C_DDR_RAM_BA	0x00000000	0x1FFFFFFFF	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
```

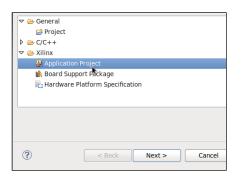
That's it! You can now access the registers of your core. These easy to access registers are very useful for specifying settings for the core or reading values that the core provides to the software. Either way, it is important to have a good logic design AND diagram so you can program the logic quickly,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

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

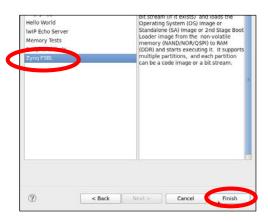


4. Type in a project name (zybo_fsbl or zynq_fsbl are recommended). Under the Board Support Package, make sure Create New is selected. Click Next

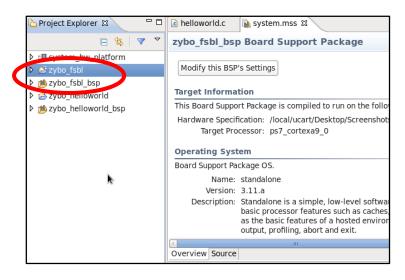
>	New Project	
pplication Project Create a managed ma	ke application project.	G
Project name: zybo_t	ísbl	
Use default location	ก	
Location: //local/ucar	/Desktop/Screenshots/zybo_fsbl	Browse
Choose file	system: default 2	
Target Hardware		
	system_hw_platform	\$
Processor	ps7_cortexa9_0	0
Target Software		
OS Platform	standalone	0
Language		
Board Support Packa	ge Create New zybo_fsbl_bsp	
	O Use existing zybo helloworld bsp	
	*	
(3)	< Back Next > Cancel	Finish



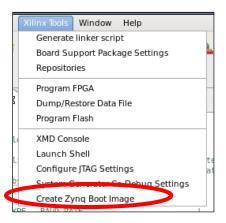
5. Select Zynq FSBL as a template (FSBL = First Stage Bootloader). Click Finish



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



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

@	Create Zynq Boot Image		×
Create Zynq	-		
Creates Zyng	Boot Image in .bin and .mcs formats from given FSBL ed output folder.	elf and partition	
	BIF file O Import from existing BIF file		
BIF file path	/local/ucart/Desktop/Screenshots/zybo_fsbl/bootima	ge/zybo_fsbl.bif Browse	e
Use Auther	ntication		
Authentication			
РРК	Browse PSK	Browse	
SPK	Browse SSK	Browse	
SPK Signature	Browse		
Use encryption ke			
Key file	·	Browse	
Key store	BRAM O EFUSE		
Part name			
Boot image pa	rtitions		
File path		En ty Add	٦
	/local/ucart/Desktop/Screenshots/zybo_fsbl/Debug/zyb	o_fsbl.elf none Delete	2
/local/ucart/D	besktop/Screenshots/system_hw_platform/system.bit	none	۲,
		Up	
<	m	Down	
Output path	/local/ucart/Desktop/Screenshots/zybo_fsbl/bootima	ge/output.bin Browse	e
?	Car	cel Create Image]

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

®	Add partition ×
Add new boo	t image partition
😣 Partition file	path is empty
Partition type	datafile 😂
File path	Browse
Authentication	none 🗘 Encryption none 🗘
Checksum	none \$
Presign	Browse
Other	
Alignment	Offset
Reserve	Load
Startup	
?	Cancel OK

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.

6		P	artition file				×
< 🗟 ucart	📷 Desktop	Screensl ots	zybo_helloworld	Debug			
Places	Name			~	Size	Modified	$ \land $
🏥 Search	📋 src					16:12	
Recently Used	objects.r	nk			277 bytes	16:12	
💼 co3050	Sources				431 bytes	16:12	
🗟 ucart	🔷 zybo_hel	loworld.elf			138.2 KB	16:12	
📷 Desktop 🔜 File System	- 2y	lower de la faire e			109 bytes	16:12	
🛅 Documents							
🛅 Music							
Pictures							=
Tideos							
🔯 Downloads	8						
							~
4 -						*.*	٢
					Cancel	, ок	
						28	

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

Part name		
Boot image pa	rtitions	
File path	Encry	Add
	/local/ucart/Desktop/Screenshots/zybo_fsbl/Debug/zybo_fsbl.elf none Desktop/Screenshots/system_hw_platform/system.bit none	Delete
/local/ucart/D	Desktop/Screenshots/zybo_helloworld/Debug/zybo_helloworld.elf none	Edit
		Up
<	III ()	Down
Output path	/local/ucart/Desktop/Screenshots/zybo_fsbl/bootimage/output.bin	Browse
?	Cancel	Image

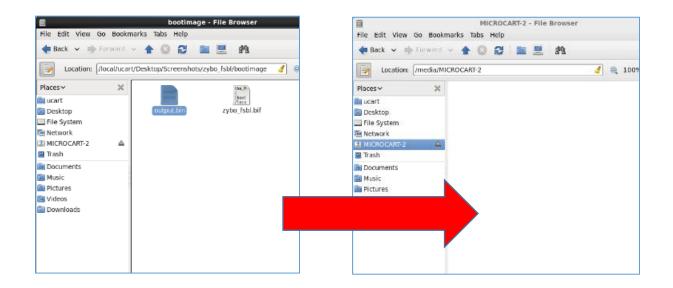
NOTE: The order should always appear in the following order:

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

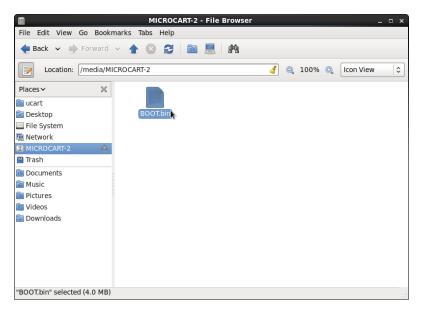
- 12. Congrats! You've now created a binary file that can be used on the Zybo. However, we still need to get the file onto a microSD and into the board itself.
- 13. Grab a microSD card (size doesn't matter) and plug it into an adapter (a USB adapter as shown for example)



- 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



16. Rename the output.bin to BOOT.bin. This <u>MUST</u> 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

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.