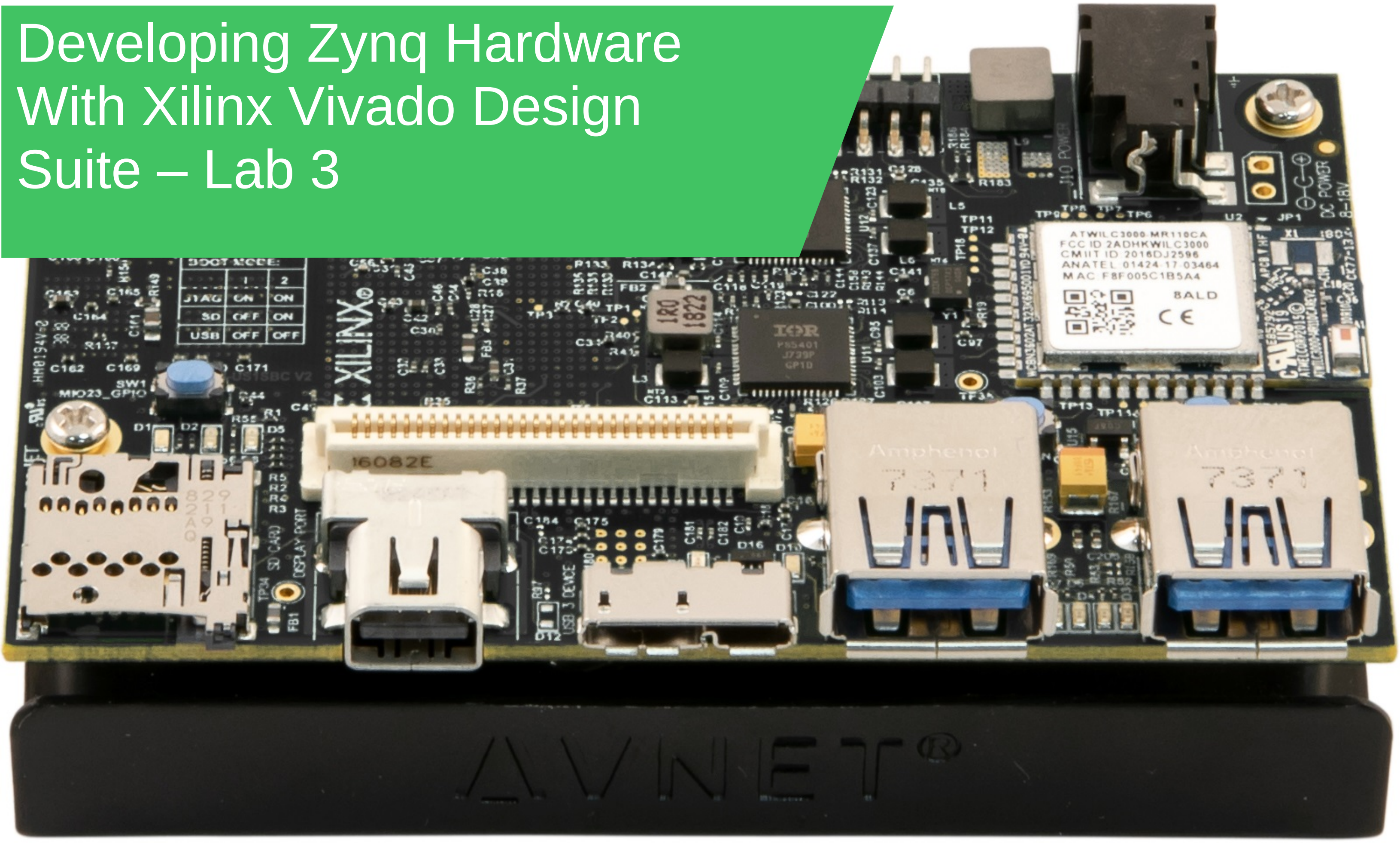
Avnet Technical Training Course

Developing Zynq Hardware With Xilinx Vivado Design Suite – Lab 3



Tools:	2019.1
Training Version:	v13
Date:	July 2019

Lab 3 Overview

Most Zynq training tutorials show you how to start with a Preset configuration for a specific development board to begin creating applications. However, the tutorials typically don't show how the Preset for that board was created. Unless you are designing an identical copy of a development board, you will need to know how to create a design without a pre-existing Preset configuration. You've been introduced to a few basics of this already in Lab 2. Now we will finish the exercise.

Additionally, it is highly recommended that all Zynq designers go through this exercise **before** committing to pins on a schematic/layout. This will help designers see what peripherals can fit onto the MIO. Only a fraction of the available peripherals will be possible to map to the MIO. Plus, the available MIO positions on which to map those peripherals is limited. Fitting the critical peripherals onto the existing MIOs and mapping the rest to EMIO is a design challenge, and IP Integrator will easily show which peripherals map where.

Lab 3 Objectives

When you have completed Lab 3, you will know how to do the following:

- Enable and map all default peripherals in IP Integrator
- Set the PS clocks for the PS peripherals and the PL
- Create and Run C programs
  - Peripheral Tests
  - Memory Test

Experiment 1: Enable and Map all PS Peripherals

This experiment shows how to map the PS Peripherals to match our evaluation board.

Experiment 1 General Instruction:

In the Vivado block design, customize the Zynq embedded ARM core to enable QSPI, Ethernet, USB, and GPIO. Map these peripherals to the MIOs matching the MiniZed PCB design.

Experiment 1 Step-by-Step Instructions:

1. <Optional> If you did not complete Lab 2 or wish to start with a clean copy, delete the `ZynqDesign` and `SDK_Workspace` folders in the `ZynqHW/2019_1` folder. Then unzip **`Solutions_Minized/ZynqHW_Lab2_Solution.zip`** to the `2019_1` folder. If you have Archive Manager installed, you can do this by right-clicking and selecting Archive Manager then extracting in to the `2019_1` folder.

2. If you closed Vivado after the last lab, open it now. Single-click on Open Project and the recent projects should appear in a pop-up list. Choose ZynqDesign in the Avnet Technical Training Course folder.

3. **Open** the Block Design.

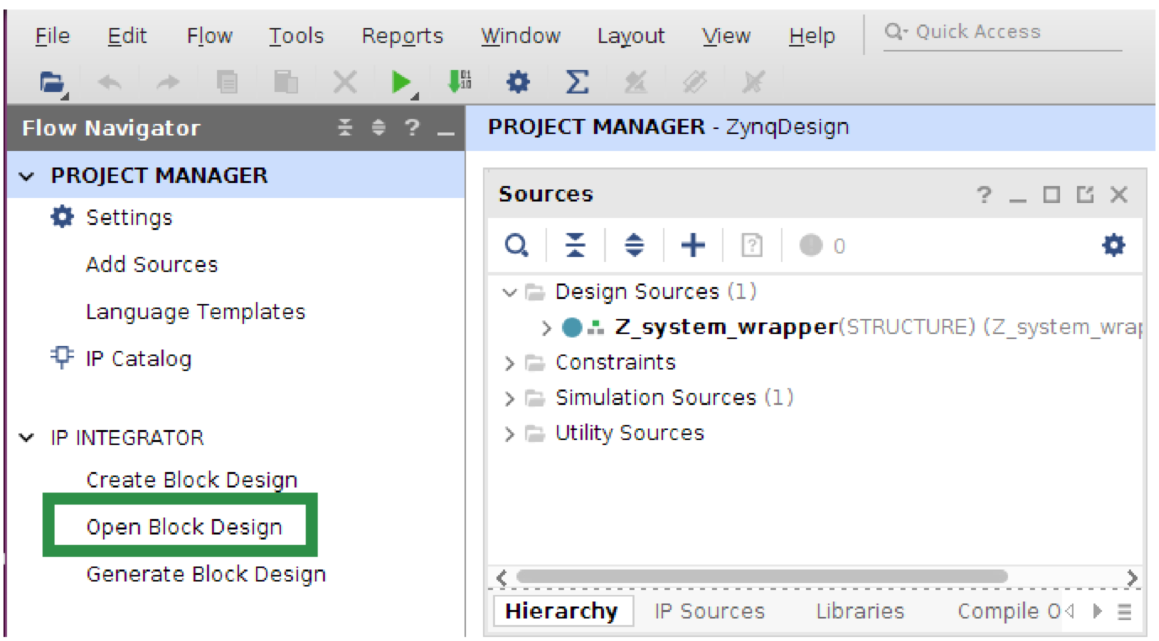


Figure 1 - Open Block Design

4. Double-click *the **ZYNQ7 Processing System*** to re-customize the processing core.

5. In the Zynq Block Design open **MIO Configuration** then click **Expand All** to view all peripherals.

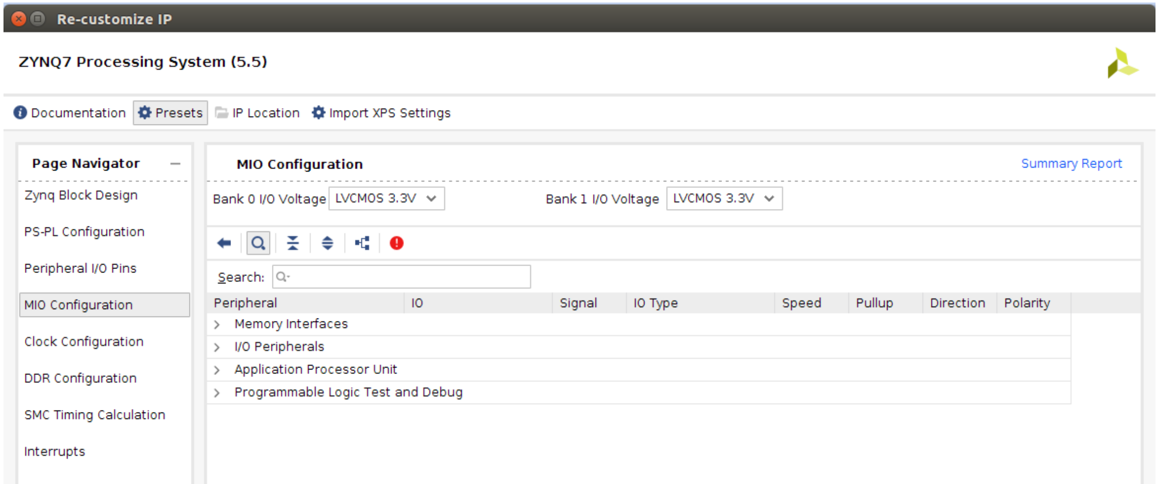


Figure 2 - MIO Configuration

Of most importance is a boot device. Zynq allows you to select just 1 of QSPI, NOR, or NAND. Note that SD Card is also a boot option and will show up lower in the list.(SD Card not available on MiniZed)

6. Check the box next to **Quad SPI Flash**.

7. Expand the QSPI to see that a **Feedback Clk** is possible, **check** the box next to it. Also, see that a 2nd QSPI can be selected for a ***Dual*** configuration (but don't do that here). Notice that the QSPI peripheral has a fixed location of **MIO[1-6, 8]**.

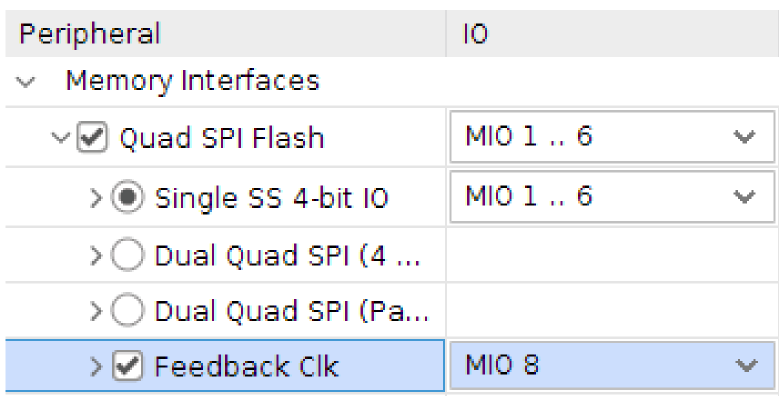


Figure 3 - QSPI Flash Connections

Notice that the SRAM/NOR Flash and NAND Flash interfaces cannot be checked. Again, this is due to the fact that only one Memory Interface is allowed from the Zynq PS.

The next peripheral in the list is Ethernet. Based on what was previously stated about priority order being top-to-bottom, Ethernet would be the least flexible peripheral remaining after the Flash devices. However, it is not. The USB is actually the least flexible. It takes fewer pins but USB <u>must</u> be mapped to MIO, just like the Flash this is due to it not being able to make timing through the EMIO. Ethernet can be mapped to EMIO. Therefore, we will skip the Ethernet for now and move on to USB.

Enable USB 0. Look at the pull-down menu. Notice that MIO[28-39] are now mapped, for MiniZed set to IOSTANDARD LVCMOS 3.3V

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction
ENET 1						
USB 0	MIO 28 .. 39					
USB 0	MIO 28	data[4]	LVCMOS 3.3V	slow	enabl	inout
USB 0	MIO 29	dir	LVCMOS 3.3V	slow	enabl	in
USB 0	MIO 30	stp	LVCMOS 3.3V	slow	enabl	out
USB 0	MIO 31	nxt	LVCMOS 3.3V	slow	enabl	in
USB 0	MIO 32	data[0]	LVCMOS 3.3V	slow	enabl	inout
USB 0	MIO 33	data[1]	LVCMOS 3.3V	slow	enabl	inout
USB 0	MIO 34	data[2]	LVCMOS 3.3V	slow	enabl	inout
USB 0	MIO 35	data[3]	LVCMOS 3.3V	slow	enabl	inout
USB 0	MIO 36	clk	LVCMOS 3.3V	slow	enabl	in
USB 0	MIO 37	data[5]	LVCMOS 3.3V	slow	enabl	inout
USB 0	MIO 38	data[6]	LVCMOS 3.3V	slow	enabl	inout
USB 0	MIO 39	data[7]	LVCMOS 3.3V	slow	enabl	inout

Figure 4 – USB 0 Connections for MiniZed

8. **For MiniZed,** the USB 0 Peripheral requires one more signal, a reset. There is a place to connect this reset. In a subfolder under the **GPIO** section, there exists a **USB Reset** subsection. Expand **GPIO** and then **USB Reset**. Enable **GPIO MIO** by clicking the checkbox next to it. Then click the checkbox next to **USB Reset**. Uncheck the **I2C Reset** box if it comes up checked by default.

   Assign USB0 Reset to **MIO[7]** as this is where it is connected on MiniZed. Don't worry about the GPIO MIO assignment, we'll cover that later. On ZedBoard, this signal is connected to a PL I/O.

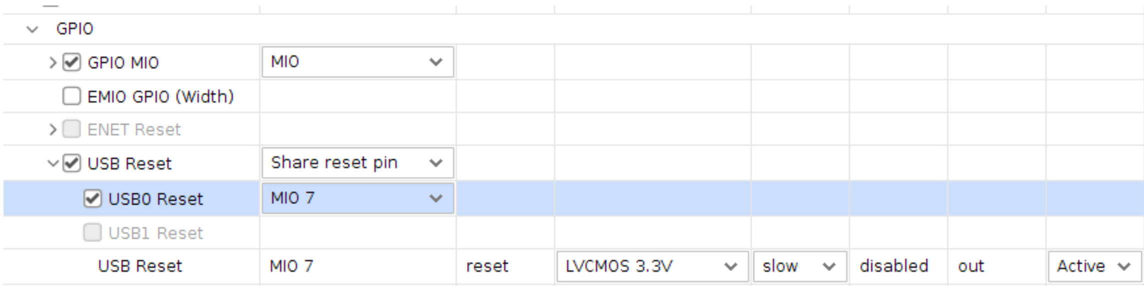


Figure 5 – MiniZed USB0 Reset Connection

9. Working our way down the list, enable **SD card or eMMC**
   For MiniZed: **SD1** mapped to **MIO[10-15]**, CD+WP leave unchecked

SD 1	MIO 10 .. 15					
CD						
WP						
Power						
SD 1	MIO 10	data[0]	LVCMOS 3.3V	slow	enable	inout
SD 1	MIO 11	cmd	LVCMOS 3.3V	slow	enable	inout
SD 1	MIO 12	clk	LVCMOS 3.3V	slow	enable	inout
SD 1	MIO 13	data[1]	LVCMOS 3.3V	slow	enable	inout
SD 1	MIO 14	data[2]	LVCMOS 3.3V	slow	enable	inout
SD 1	MIO 15	data[3]	LVCMOS 3.3V	slow	enable	inout

Figure 6 - MiniZed SD 1 Connections

10. To see what peripherals can be mapped to any remaining I/O, open the **Peripheral I/O Pins** page from the Page Navigator. Here you will see all remaining options for connecting PS Peripherals. Any MIO highlighted in green are already assigned.

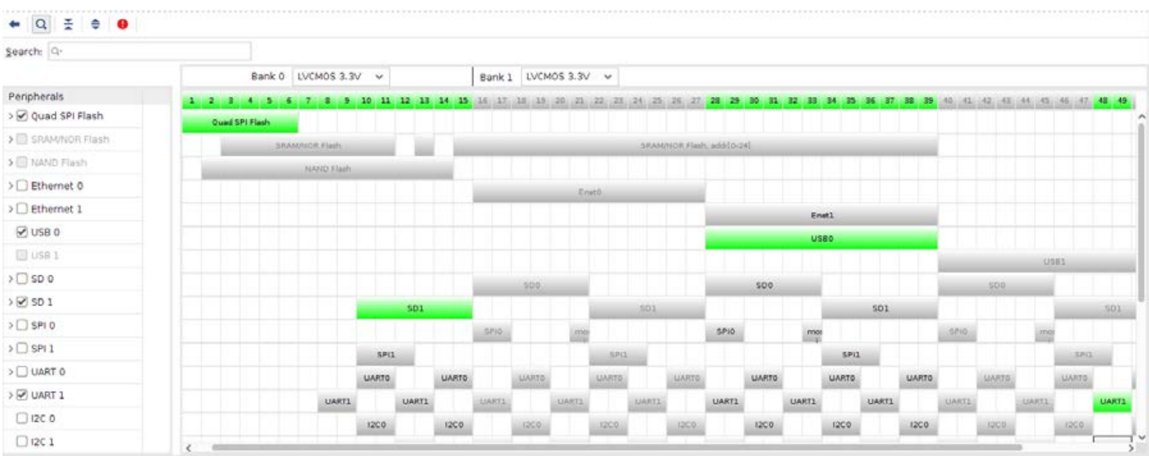


Figure 7 - All MIO Connections (MiniZed)

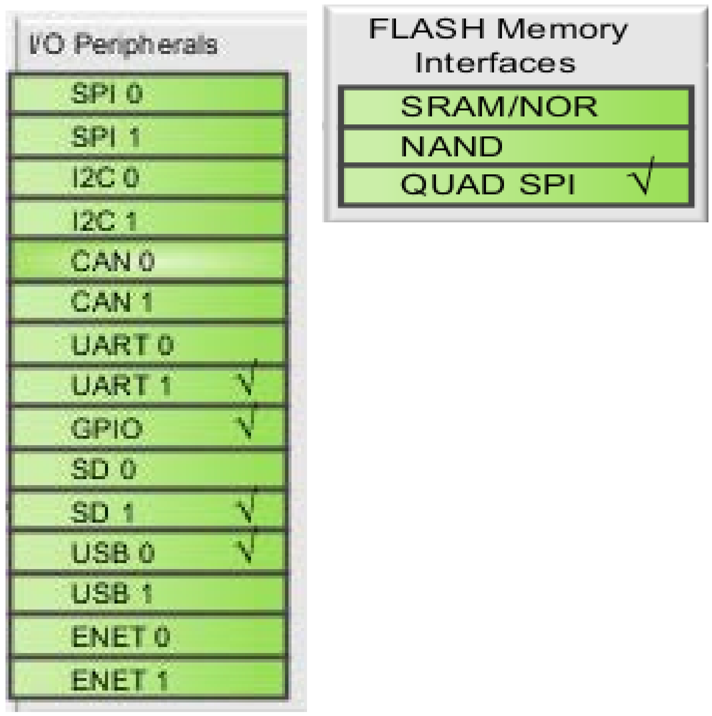


Figure 8 - Final I/O Peripheral Connections (MiniZed)

Refer to figure 8 to ensure all peripherals and your Flash Memory Interfaces have been set.

Questions:

Answer the following questions:

- *Why is EMIO not an option for connecting USB 0?*

  ____

  ____

- *Which other peripherals would you expect not to have EMIO options? Why?*

  ____

  ____

- *What is the Power pin for on the SD peripheral? (Hint: use the TRM)*

  ____

  ____

- *Name one scenario where having independent access to PJTAG would be useful.*

  ____

- *What is special about MIO[7] and MIO[8]?*

  ____

Experiment 2: Set the PS PLL Clocks

Now that we have more PS peripherals enabled, we need to go look at the clocks again. We'll also look at the clocks that the PS can pass to the PL.

Experiment 2 General Instruction:

Configure the PS Clocks for the newly enabled peripherals. Set the PL Fabric Clocks as well as CPU and DDR Clocks.

Experiment 2 Step-by-Step Instructions:

1. Click on the box for **Clock Generation** or **Clock Configuration** from the Page Navigator.

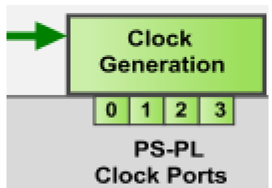


Figure 9 – PS Configuration Clock Generation

2. **Expand** all the clocks. The dialog appears below.

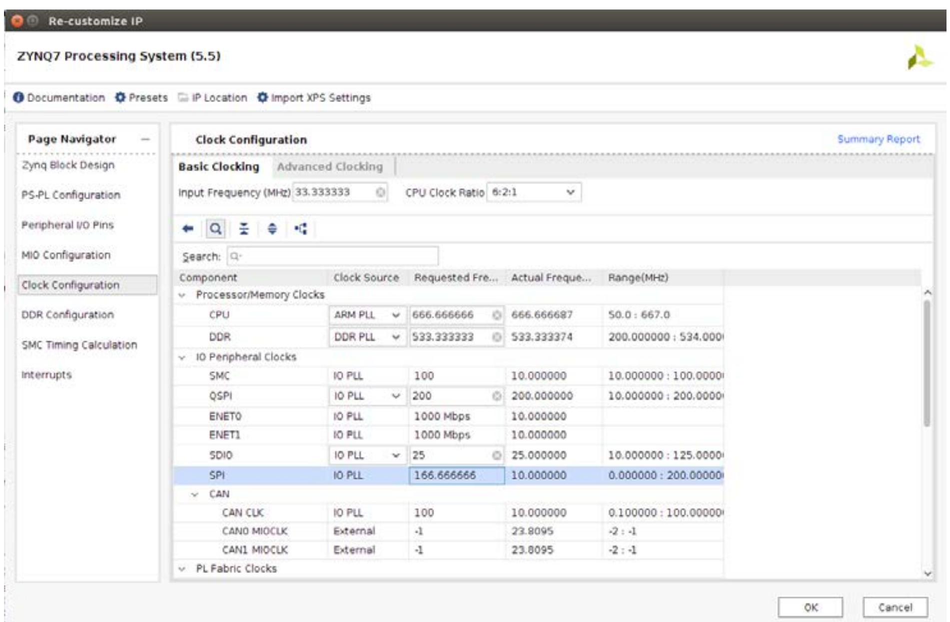


Figure 10 – PS Clock Wizard

Recall that the Zynq PS has three PLLs – ARM, DDR, and I/O. Each uses the same input reference clock, which is 33.333333 MHz on MiniZed.

Each PLL must be set to operate in a specific frequency range, as given by the datasheet. Note that for the -1 device, this range is 780 MHz to 1600 MHz.

Table 21: **PS PLL Switching Characteristics**

Symbol	Description	Speed Grade				Units
		-3	-2	-1C/-1I/-1LI	-1Q	
T_{LOCK_PSPLL}	PLL maximum lock time	60	60	60	60	µs
F_{PSPLL_MAX}	PLL maximum output frequency	2000	1800	1600	1600	MHz
F_{PSPLL_MIN}	PLL minimum output frequency	780	780	780	780	MHz

Figure 11 – PS PLL Switching Characteristics from Zynq Datasheet

Once the PLL output frequency is set, then you are limited by integer dividers to generate the clock that you want.

3. Review the settings previously entered/verified for PS clocks.
    a. Input frequency = 33.333333 MHz
    b. CPU frequency = 666.666666 MHz
    c. DDR frequency = 533.333333 MHz

These settings have already dictated the PLL output frequency for the ARM and DDR PLLs, both of which must be multiples of 33.333 MHz. Since the CPU frequency must be an integer divider of the ARM PLL, we know that the ARM PLL must be set to 1333.33 MHz (33.333 MHz * 40) and the CPU clock divider must be 2. The DDR PLL output frequency could be either 1600 MHz (33.333 * 48) or 1066.667 MHz (33.333 * 32). By default the tools will set this to 1066.667 and use a divider of 2.

Everything else that uses these PLLs must now use an integer divider of the set output frequency. The same principle will apply to the I/O PLL.

Similar to the CPU and DDR being the determining factors for the ARM and DDR PLLs, the I/O PLL also has some prioritized dependencies.

4. By default, the QSPI peripheral uses the IO PLL as its clock source. The QSPI peripheral interface operates at a maximum rate of 100 MHz. The QSPI peripheral has an internal divider, which by default is set to 2, and we're not going to change that. Therefore, the QSPI input clock is set to **200 MHz**.

IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.0000
QSPI	IO PLL	200	200.000000	10.000000 : 200.0000
ENET0	IO PLL	1000 Mbps	10.000000	
ENET1	IO PLL	1000 Mbps	10.000000	

Figure 21 – QSPI Requested Frequency Set to 200 MHz

5. We have one more IO Peripheral Clocks to modify. Set the **SDIO** to **25MHz on MiniZed boards.**

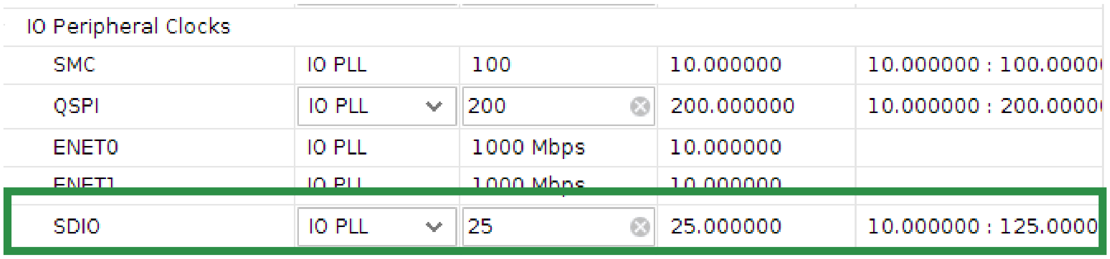


Figure 12 - SDIO Requested Frequency Set to 25

6. The PS also has the ability to drive 4 different clock frequencies into the PL. **Enable and then set** these now as follows:
   - FCLK_CLK0 to 100 MHz
   - FCLK_CLK1 to 150 MHz
   - FCLK_CLK2 to 50 MHz
   - FCLK_CLK3 to 25 MHz

7. Correct clock settings are critical for hardware peripherals to work properly in later lab activities. Please take the time now to verify that the clock settings match those shown in Figure 13 before continuing with the remainder of the experiment.

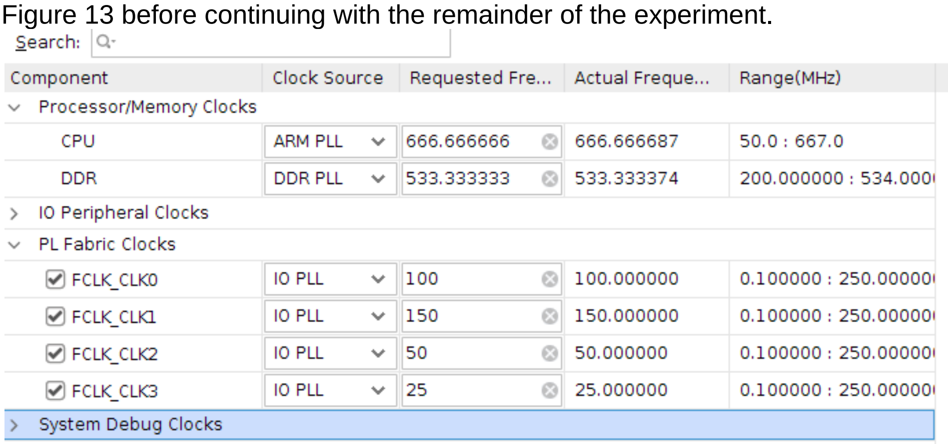


Figure 13 –Clock Settings

8. **Turn off** all Fabric Clocks as they are not needed yet. We enabled these to show the PLL capability.

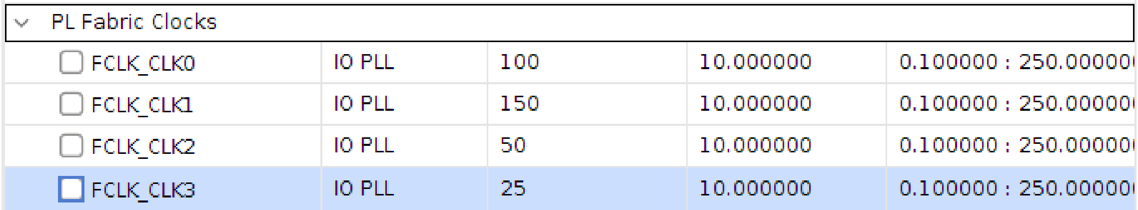


Figure 24 - Clock Configuration

9. Close the Zynq Block Design by clicking **OK**.
10. **Validate** (F6) the Block Design.
11. **Save** the Block Design.

The block design is simply a graphical representation over some HDL code that the tool is going to generate. The code and files generated from the block design is referred to as ***Output Products***. When we first generated the bitstream, the tool automatically generated the output products for us. We have now modified the Block Design. Typically, the tool should recognize that the Block Design is newer than the existing Output Products and then re-generate them. However, to be safe, we will explicitly reset and re-generate the Output Products manually.

12. Select the *Design Sources*. Right-click on the *Z_system_i* item and select **Reset Output Products…**

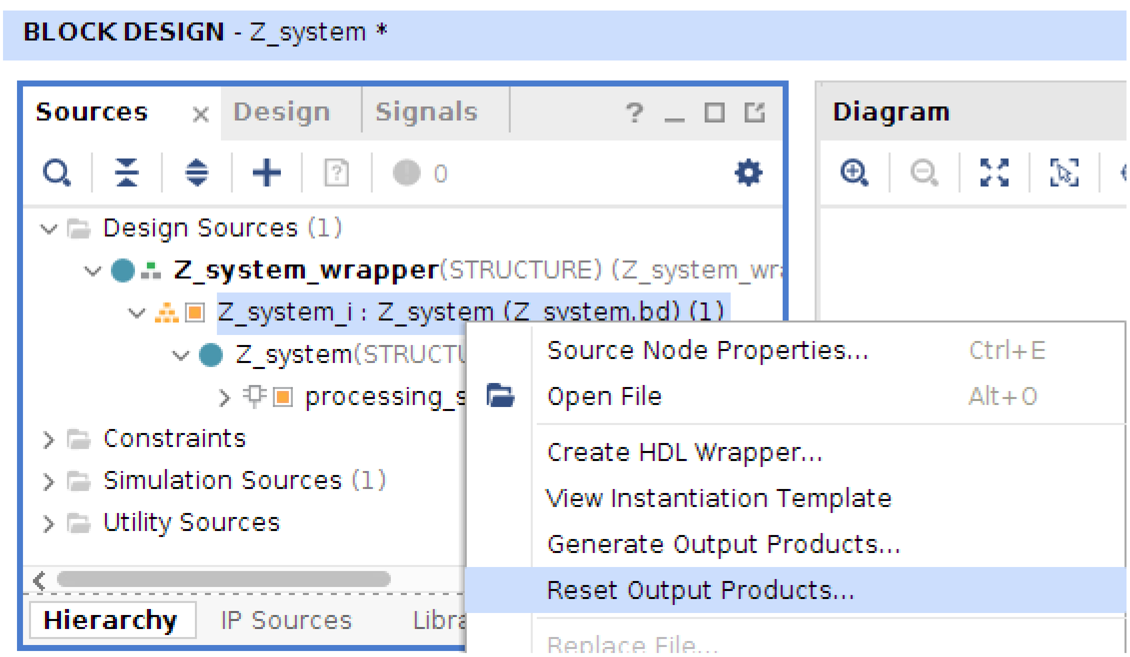


Figure 14 – Reset Output Products

13. Click **Reset** to confirm resetting of the output products.

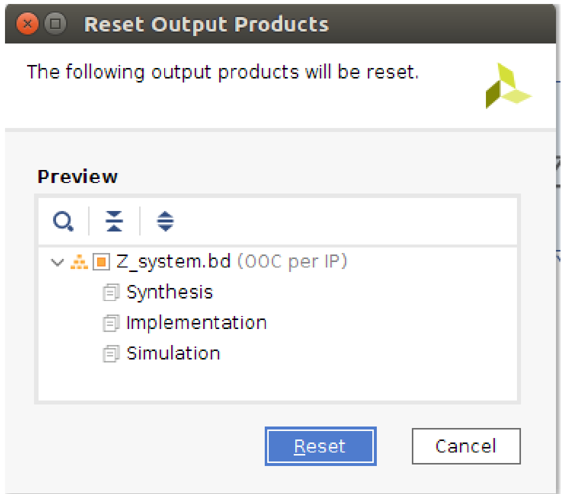


Figure 15 – Confirm Reset Output Products

14. Now right-click on *Z_system_i* again, but this time select **Generate Output Products**, make sure **Synthesis Options** is set to **Global** then click **Generate** to confirm.

15. Select **Generate Bitstream**, this will re-launch synthesis and implementation, click **Yes** to accept. If asked, save the block design. Then **Select ok** on the Launch Runs page. This process should take only a few minutes.

16. Once completed, open the implemented design. If that option does not appear, click **Cancel** in the pop-up then open the implemented design and select **Reload**.

Experiment 3: Create and Run Test Applications

With more peripherals enabled in the PS, we're in position to use a few of the more advanced test applications.

Experiment 3 General Instruction:

Export to SDK. Generate and run the Memory and Peripheral test applications.

Experiment 3 Step-by-Step Instructions:

1. SDK should be closed. If not, close it now.

2. Export to SDK as you did in Lab 2. Select **File → Export → Export Hardware...**

Remember that we previously exported a hardware platform. We let Vivado manage the location of this. We also launched SDK from Vivado, using this project-managed export location.

While this works well for the first hardware export, subsequent exports may be subject to issues. These issues typically evidence themselves as missing files in the BSP when you go to compile.

To avoid these issues, we recommend that you explicitly manage the location of your hardware export and SDK Workspace.

3. The *Export Hardware* dialog box opens. Make sure the **Include bitstream** box is checked. In the *Export to:* dialogue, select the pull-down and **Choose Location**. Browse to */home/training/AvnetTTC/ZynqHW/2019_1/ZynqDesign/ZynqDesign.lab3* and then create a directory called **ZynqDesign.lab3** using the icon. Now click **Select**. The dialogue should be displayed as below. Click **OK**.

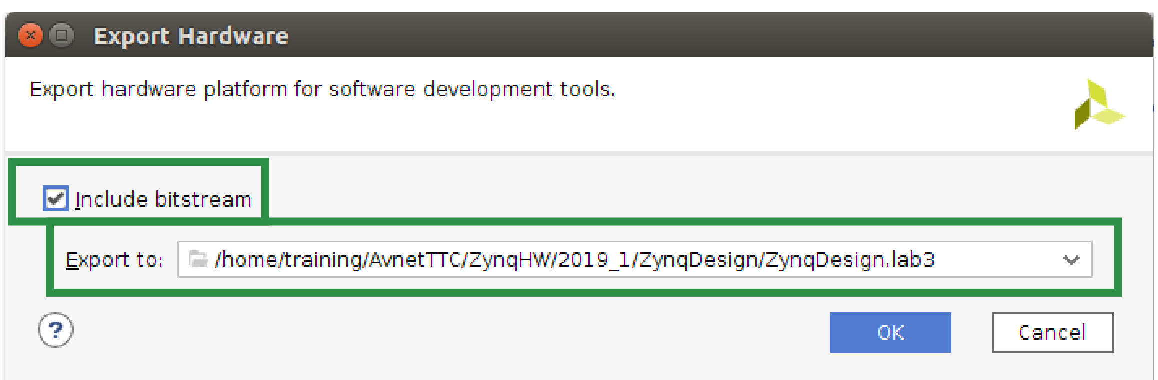


Figure 16 - Export Hardware Dialog Box

4. From the pull-down menus at the top of Vivado, select **File → Launch SDK**.

5. Change both the *Exported location* and the *Workspace* to be the new **ZynqDesign.lab3** location. Click **OK**.

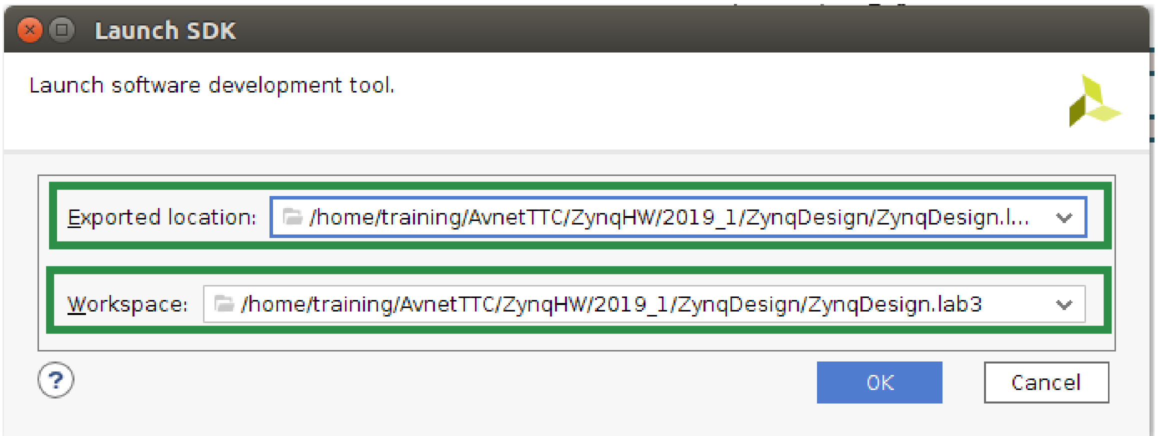


Figure 17 - Launch SDK Dialog Box

6. SDK will import the new hardware platform. Generate the Standalone BSP as was done in Lab 2 (**File → New → Board Support Package**). Also, create the Hello World application as before(**File → New → Application Project**).

7. As a sanity check, run the Hello World example again. Make sure your board is connected and powered. Right-click on hello_world_0. Select **Run As → Run Configurations**. Set up the in the **Hello_World Debug** configuration and **GTKTerm** and then click **Run**.

8. Follow the same **New Xilinx Application Project** procedure that you followed in Lab 2 for Hello World. This time, generate the Memory Test application.
   - **File → New → Application Project**
   - Enter the Project name of **Memory_Tester**
   - Select **Use existing: standalone_bsp_0**, then **Next >**
   - Select **Memory Tests**, then **Finish**
   - Right-click on **Memory_Tester**, select **Run As → Run Configurations**
   - Select **Xilinx C/C++ Application (System Debugger)**, then (New Configuration)
   - Open Terminal, such as **GTKTerm**, and set the COM port to active **COM** setting for your board and set the **Baud Rate to 115,200**
   - Click **Run**
   - If the Conflict/ FPGA Configuration pops up, click Yes to both to continue.

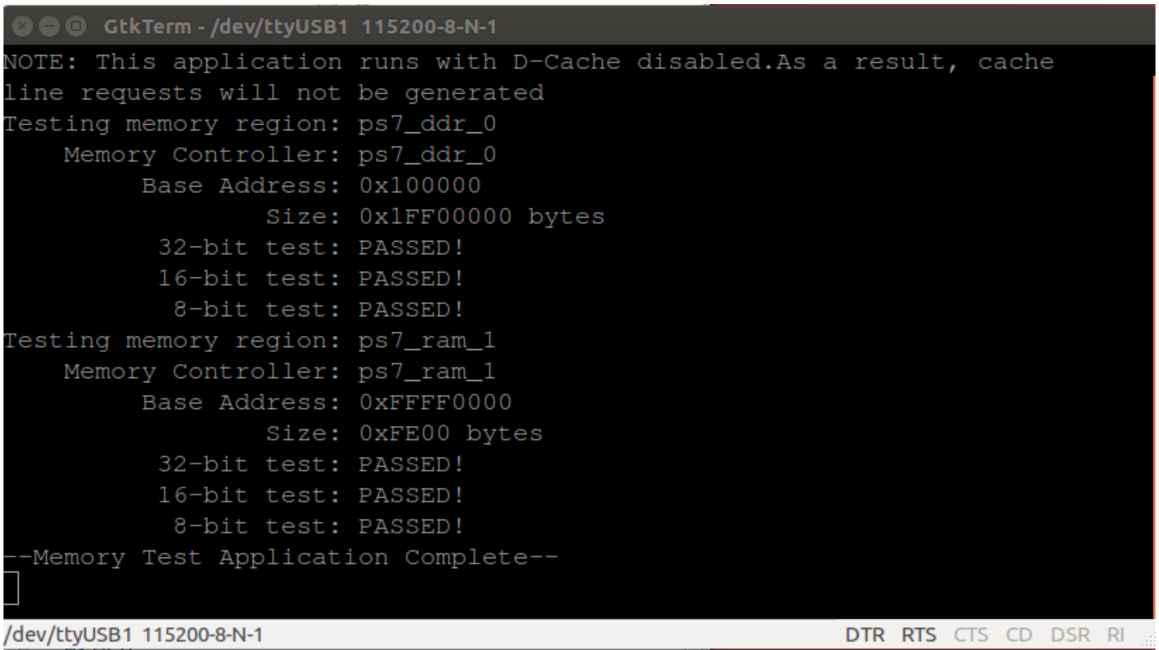


Figure 18 - Memory Test Results in GTKterm

9. Now repeat the same procedure for **Peripheral Tests**.

   Note: Occasionally, the PHY will become unresponsive as a result of some of the previous tests leaving the hardware in an unknown state which results in a false failure indication for the hardware. If you encounter this condition, you can clear this condition by power cycling the MiniZed board and re-running the Peripheral Test. One alternative to power cycling is to modify the **Run Configuration** for this application and replace the **Reset Processor** option with the **Reset Entire System** option and launch the application on the target board.

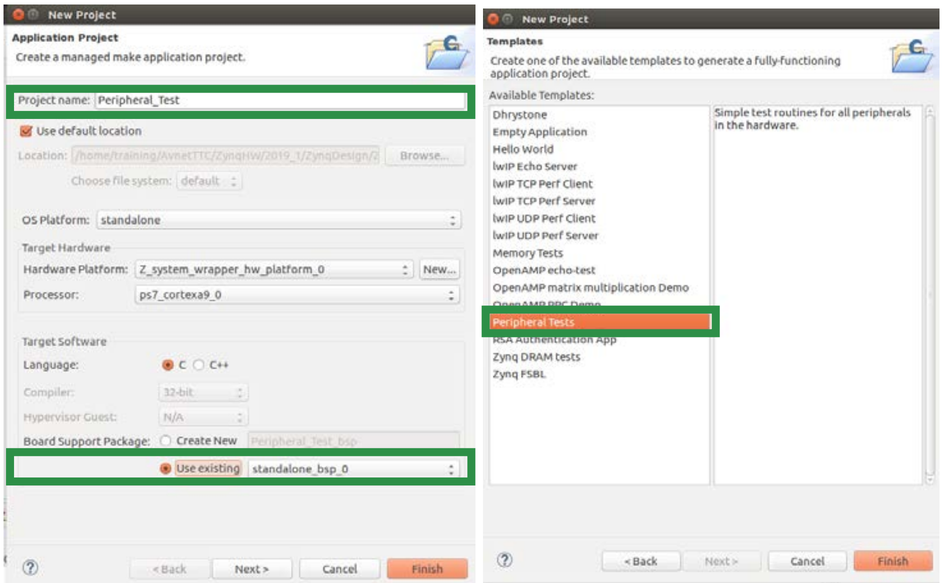
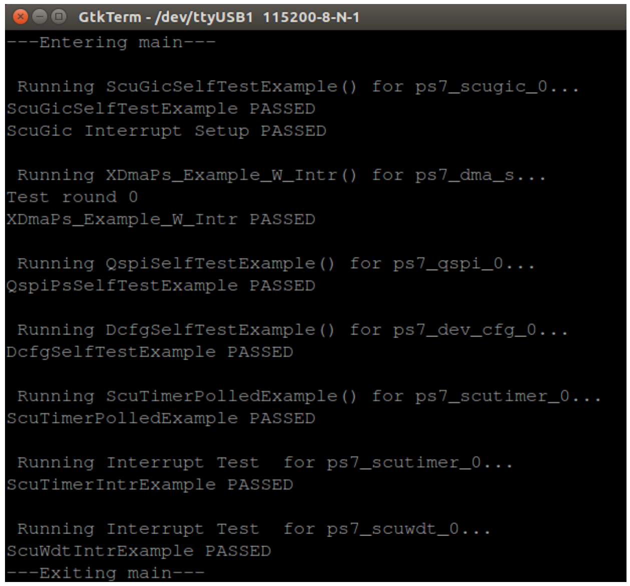


Figure 19 – New Xilinx Application Project: Peripheral_Test



```
GtkTerm - /dev/ttyUSB1  115200-8-N-1
---Entering main---

 Running ScuGicSelfTestExample() for ps7_scugic_0...
ScuGicSelfTestExample PASSED
ScuGic Interrupt Setup PASSED

 Running XDmaPs_Example_W_Intr() for ps7_dma_s...
Test round 0
XDmaPs_Example_W_Intr PASSED

 Running QspiSelfTestExample() for ps7_qspi_0...
QspiPsSelfTestExample PASSED

 Running DcfgSelfTestExample() for ps7_dev_cfg_0...
DcfgSelfTestExample PASSED

 Running ScuTimerPolledExample() for ps7_scutimer_0...
ScuTimerPolledExample PASSED

 Running Interrupt Test  for ps7_scutimer_0...
ScuTimerIntrExample PASSED

 Running Interrupt Test  for ps7_scuwdt_0...
ScuWdtIntrExample PASSED
---Exiting main---
```

Figure 20 – Peripheral Tests Results in GTKterm

10. **Close** SDK.
11. **Close** Vivado.

Questions:

Answer the following questions:

- *Look again at* Figure 18 - Memory Test Results in GTKterm*. What is the Base Address for the memory test?*

____

- *Why doesn't the DDR start at address 0x0?*

____

____

____

Exploring Further

If you have more time and would like to investigate more…

- Modify the Memory Test to test the entire memory space. *Hint – look in the test_memory_range function in memorytest.c*

This concludes Lab 3.

Revision History

Date	Version	Revision
6 Nov 13	02	Initial Draft
19 Nov 13	03	Pilot Updates
30 Oct 14	04	Updated for Vivado 2014.3
5 Jan 15	05	Updated for Vivado 2014.4
04 Mar 15	06	Finalize for Vivado 2014.4
17 Mar 15	07	Minor edits for release
Oct 2015	08	Updated to 2015.2
July 2016	09	Updated to 2016.2
May 2017	10	Updated to 2017.1 and added MiniZed board support
June 2017	11	Updated to 2017.1 for MiniZed Only
Nov 2017	12	Minor Update
Jan 2018	13	Updated to 2017.4
July 2019	14	Updated to 2019.1

Resources

www.minized.org

www.microzed.org

www.picozed.org

www.zedboard.org

www.xilinx.com/zynq

www.xilinx.com/sdk

www.xilinx.com/vivado

Answers

Experiment 1

- *Why is EMIO not an option for connecting USB 0?*

Due to the ULPI interface standard for the USB peripherals, timing could not be met by going through the EMIO. Therefore, USB peripherals MUST be connected to MIO or they are lost.

- *Which other peripherals would you expect not to have EMIO options? Why?*

The 3 primary boot Flash options (QSPI Flash in this case). All other peripherals can be mapped to EMIO. The Flash cannot be mapped to EMIO because we depend on the Flash to be the boot device.

- *Name one scenario where having independent access to PJTAG would be useful.*

Hardware/software simultaneous debug between non-Xilinx tools. The standard JTAG port is used for Vivado Logic Analyzer while the PJTAG is routed out to a Pmod (Using EMIO) and then used with an ARM DS-5 or DSTREAM for processor debug.

- *What is special about MIO[7] and MIO[8]?*

They are output only.

Experiment 3

- *Look again at figure 18 – Memory Test Results. What is the Base Address for the memory test?*

0x00100000

- *Why doesn't the DDR start at address 0x0?*

This is where OCM resides. You can remap DDR to reside at address 0x0, but this is typically done during a 2nd-stage bootloader. See Section 29.4.1 of the UG585.