Introduction to Zynq Hardware

Lab 3

PS Configuration Part 2 – MIO Peripherals

November 2013
Version 03

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, SD Card, and GPIO. Map these peripherals to the MIOs matching the ZedBoard or MicroZed 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/2013_3` folder. Then unzip **`Solutions\ZynqHW_Lab2_Solution.zip`** to the `2013_3` folder. If you have 7-zip installed, you can do this by right-clicking and dragging **`ZynqHW_Lab2_Solution.zip`** to the `2013_3` folder. Select **7-Zip → Extract Here**.

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 Speedway folder.

3. **Open** the Block Design.

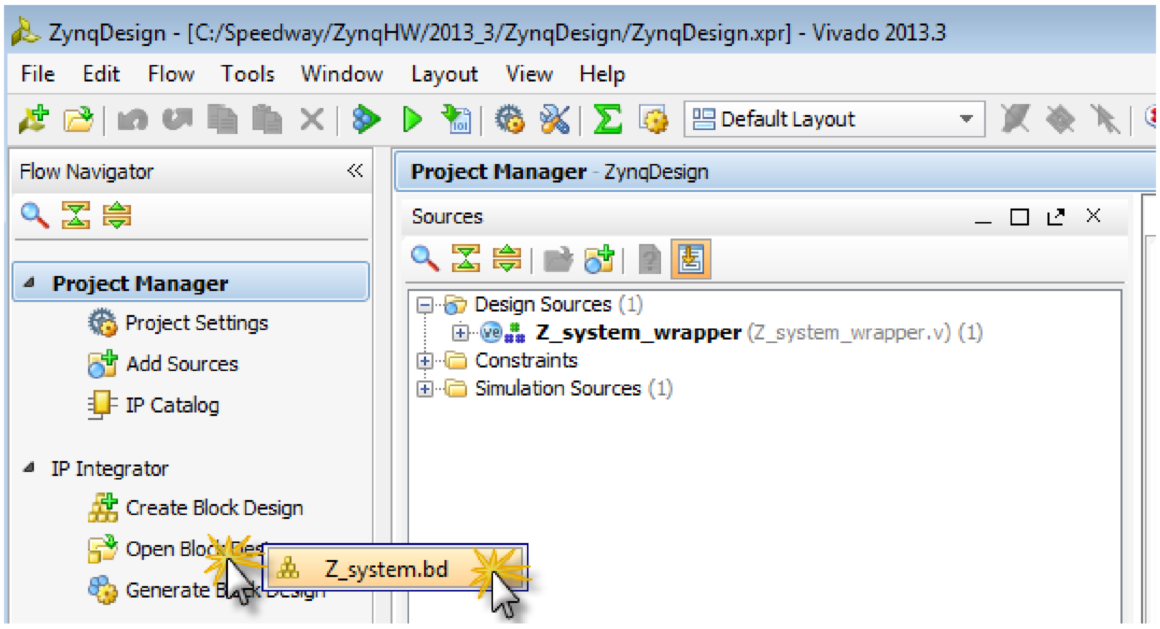


Figure 1 - Open Block Design

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.

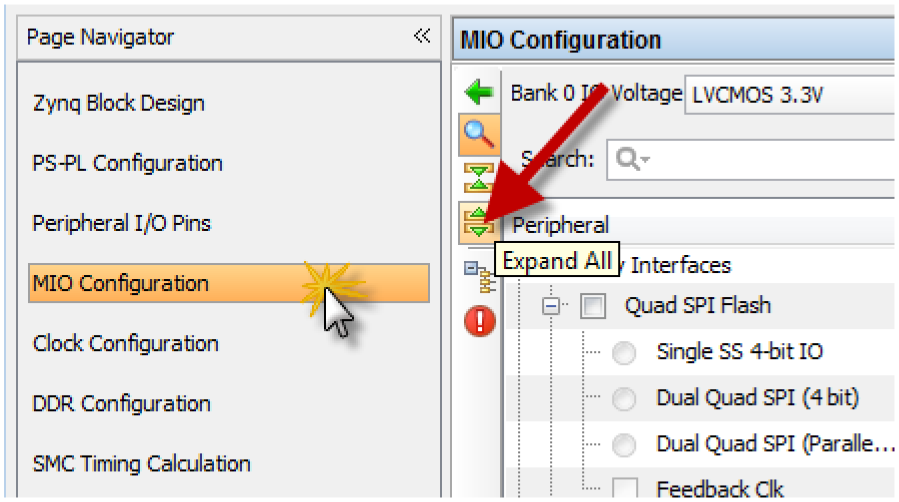
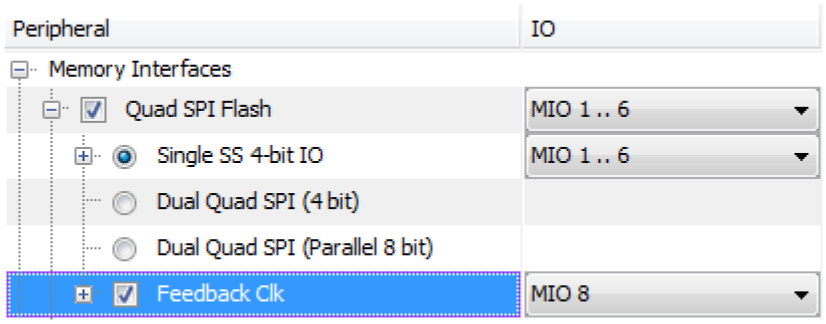


Figure 2 - MIO Configuration

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. Both MicroZed and ZedBoard feature an on-board Spansion Quad SPI Flash.

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]**.



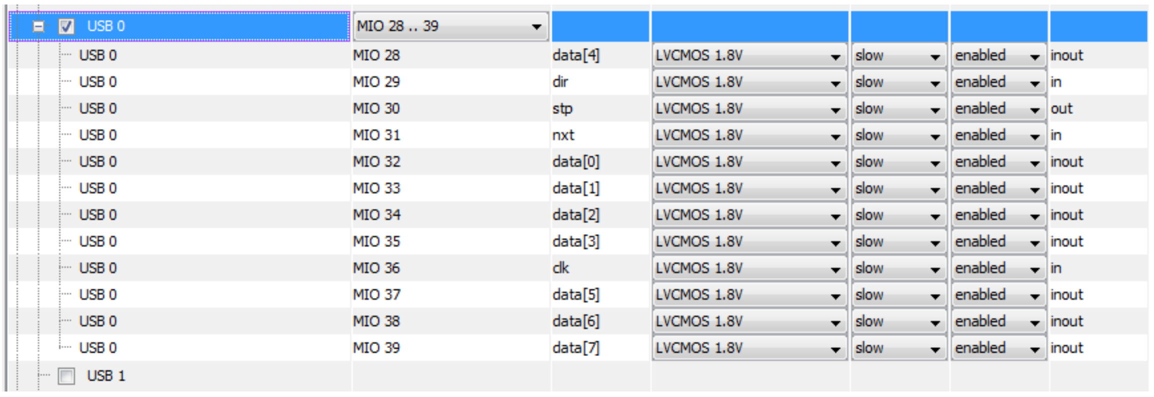
Peripheral	IO
Memory Interfaces	
✓ Quad SPI Flash	MIO 1 .. 6
● Single SS 4-bit IO	MIO 1 .. 6
○ Dual Quad SPI (4 bit)	
○ Dual Quad SPI (Parallel 8 bit)	
✓ Feedback Clk	MIO 8

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 must be mapped to MIO, just like the Flash. Ethernet can be mapped to EMIO. Therefore, we will skip the Ethernet for now and move on to USB.

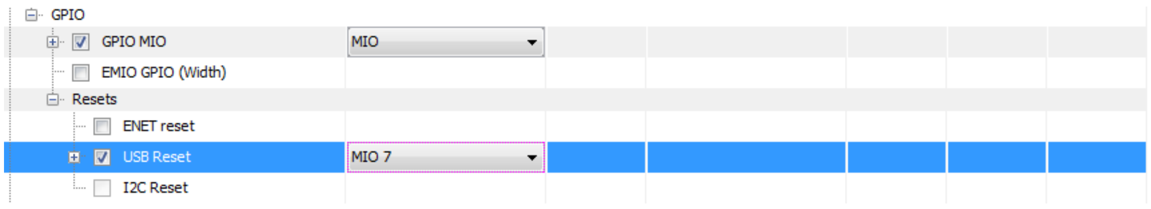
8. Enable **USB 0**. Look at the pull-down menu. Notice that **MIO[28-39]** are now mapped, set to IOSTANDARD LVCMOS 1.8V.



☑ USB 0	MIO 28 .. 39					
USB 0	MIO 28	data[4]	LVCMOS 1.8V	slow	enabled	inout
USB 0	MIO 29	dir	LVCMOS 1.8V	slow	enabled	in
USB 0	MIO 30	stp	LVCMOS 1.8V	slow	enabled	out
USB 0	MIO 31	nxt	LVCMOS 1.8V	slow	enabled	in
USB 0	MIO 32	data[0]	LVCMOS 1.8V	slow	enabled	inout
USB 0	MIO 33	data[1]	LVCMOS 1.8V	slow	enabled	inout
USB 0	MIO 34	data[2]	LVCMOS 1.8V	slow	enabled	inout
USB 0	MIO 35	data[3]	LVCMOS 1.8V	slow	enabled	inout
USB 0	MIO 36	clk	LVCMOS 1.8V	slow	enabled	in
USB 0	MIO 37	data[5]	LVCMOS 1.8V	slow	enabled	inout
USB 0	MIO 38	data[6]	LVCMOS 1.8V	slow	enabled	inout
USB 0	MIO 39	data[7]	LVCMOS 1.8V	slow	enabled	inout
☐ USB 1						

Figure 4 – USB_0 Connections

9. **For MicroZed only**, not ZedBoard, 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 exists a Resets subsection. Expand **GPIO** and **Resets**. Enable **GPIO MIO** by clicking the checkbox next to it. Then click the checkbox next to **USB_Reset**. Assign this to **MIO[7]** as this is where it is connected on MicroZed. Don't worry about the GPIO MIO assignment, we'll cover that later. On ZedBoard, this signal is connected to a PL I/O.



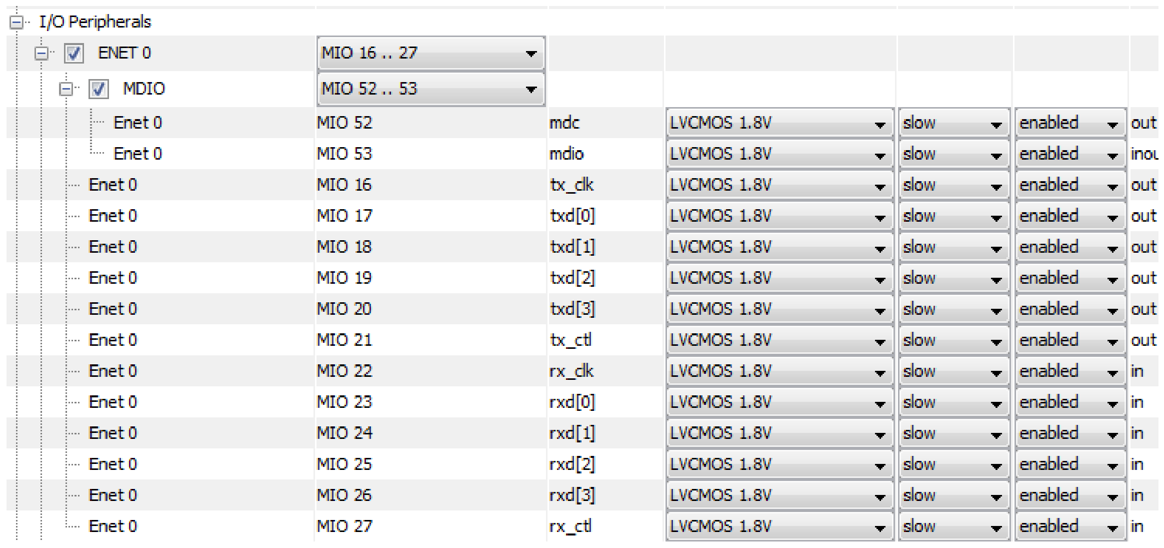
GPIO	
☑ GPIO MIO	MIO
☐ EMIO GPIO (Width)	
Resets	
☐ ENET reset	
☑ USB Reset	MIO 7
☐ I2C Reset	

Figure 5 – MicroZed USB Reset Connection

10. Next, Enable **ENET 0** by clicking on the checkbox.

11. Map to **MIO[16-27]** by clicking on the pull-down arrow and selecting MIO[16-27].

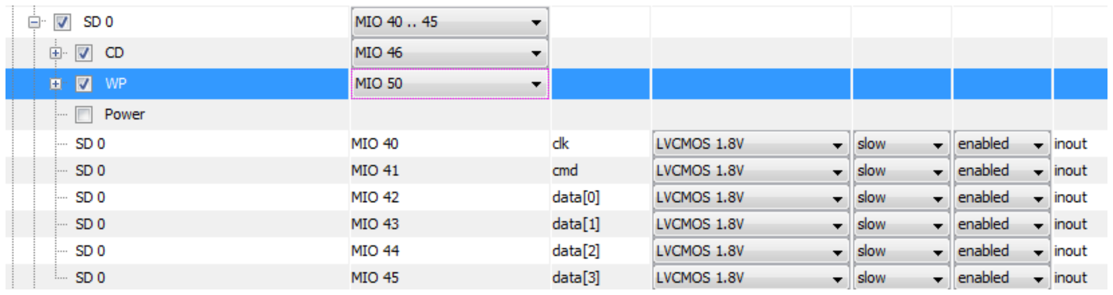
12. Expand **ENET 0** then enable **MDIO** by clicking its checkbox. Change the mapping to **MIO[52-53]**. Notice that the I/O Type is set to LVCMOS 1.8V.



I/O Peripherals						
✓ ENET 0	MIO 16 .. 27					
✓ MDIO	MIO 52 .. 53					
Enet 0	MIO 52	mdc	LVCMOS 1.8V	slow	enabled	out
Enet 0	MIO 53	mdio	LVCMOS 1.8V	slow	enabled	inou
Enet 0	MIO 16	tx_clk	LVCMOS 1.8V	slow	enabled	out
Enet 0	MIO 17	txd[0]	LVCMOS 1.8V	slow	enabled	out
Enet 0	MIO 18	txd[1]	LVCMOS 1.8V	slow	enabled	out
Enet 0	MIO 19	txd[2]	LVCMOS 1.8V	slow	enabled	out
Enet 0	MIO 20	txd[3]	LVCMOS 1.8V	slow	enabled	out
Enet 0	MIO 21	tx_ctl	LVCMOS 1.8V	slow	enabled	out
Enet 0	MIO 22	rx_clk	LVCMOS 1.8V	slow	enabled	in
Enet 0	MIO 23	rxd[0]	LVCMOS 1.8V	slow	enabled	in
Enet 0	MIO 24	rxd[1]	LVCMOS 1.8V	slow	enabled	in
Enet 0	MIO 25	rxd[2]	LVCMOS 1.8V	slow	enabled	in
Enet 0	MIO 26	rxd[3]	LVCMOS 1.8V	slow	enabled	in
Enet 0	MIO 27	rx_ctl	LVCMOS 1.8V	slow	enabled	in

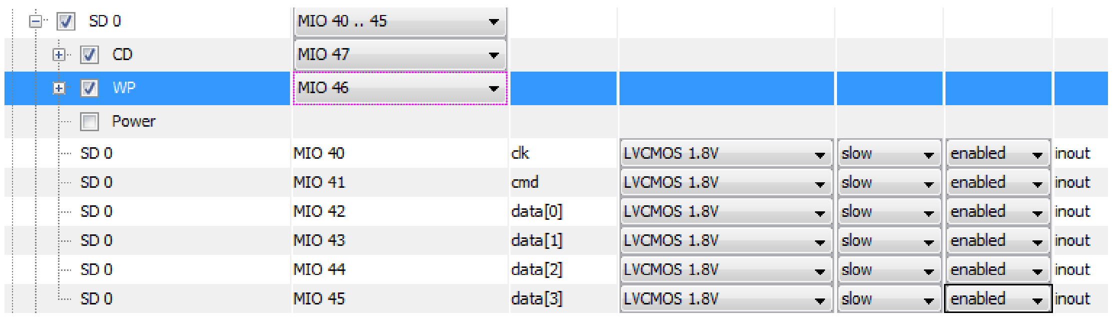
Figure 6 – Ethernet_0 Connections

13. Working our way down the list, enable **SD 0** next, mapped to **MIO[40-45]**.
    For ZedBoard: CD = **MIO 47**, WP = **MIO 46**.
    For MicroZed: CD = **MIO 46**, WP = **MIO 50.**



✓ SD 0	MIO 40 .. 45					
✓ CD	MIO 46					
✓ WP	MIO 50					
Power						
SD 0	MIO 40	clk	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 41	cmd	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 42	data[0]	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 43	data[1]	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 44	data[2]	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 45	data[3]	LVCMOS 1.8V	slow	enabled	inout

Figure 7 - MicroZed SD_0 Connections



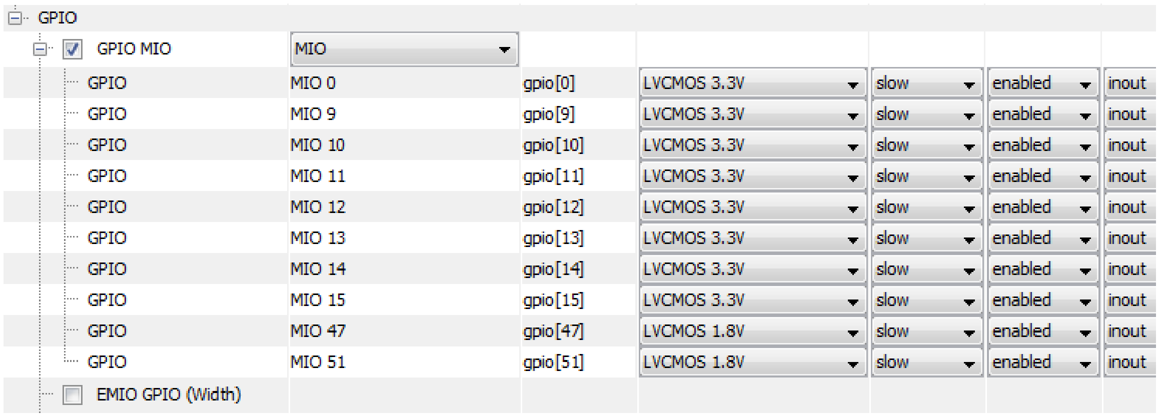
✓ SD 0	MIO 40 .. 45					
✓ CD	MIO 47					
✓ WP	MIO 46					
Power						
SD 0	MIO 40	clk	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 41	cmd	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 42	data[0]	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 43	data[1]	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 44	data[2]	LVCMOS 1.8V	slow	enabled	inout
SD 0	MIO 45	data[3]	LVCMOS 1.8V	slow	enabled	inout

Figure 8 - ZedBoard SD_0 Connections

The UART was previously selected. The remaining unused I/Os are used for the PS Pmod (8 I/Os), one LED (1 I/O), and one PB (1 I/O). The LED and PB will be GPIOs that we will set last. Now we will look at the PS Pmod.

The MicroZed and ZedBoard design team chose to assign eight MIOs to the PS Pmod. These eight I/Os were carefully mapped to MIO[0, 9-15]. A contiguous span of MIOs was desired, but this was not possible. MIO[7] and MIO[8] were not included because they are special MIO that can only be used as outputs. Thus, MIO[7] is what we connect to the LED on ZedBoard and the USB_Reset on MicroZed. MIO[8] is used for the Quad SPI Flash interface. MicroZed connects a LED to MIO[47] and a pushbutton to MIO[51]. ZedBoard provides 7 user GPIO push buttons to the Zynq-7000 AP SoC; five on the PL-side and two on the PS-side; MIO[50:51].

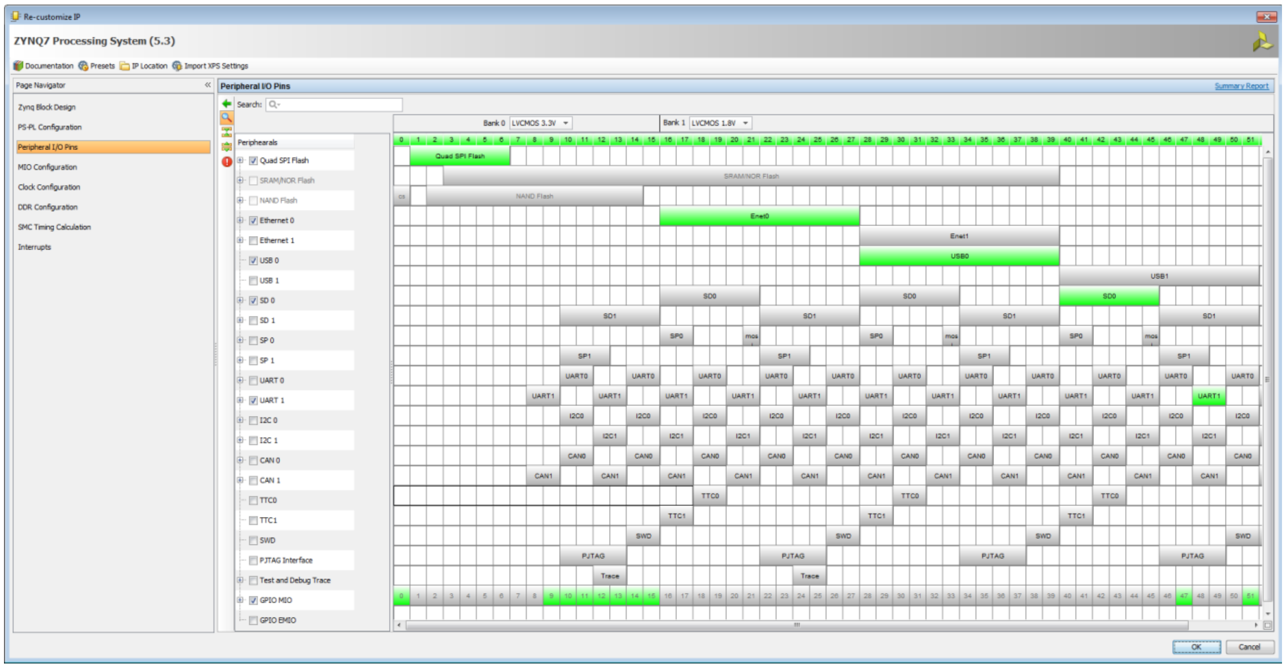
14. Earlier, we enabled GPIO for the MIO. Vivado will assign all remaining MIO to GPIO, so nothing else needs to be done.



GPIO						
☑ GPIO MIO	MIO					
GPIO	MIO 0	gpio[0]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 9	gpio[9]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 10	gpio[10]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 11	gpio[11]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 12	gpio[12]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 13	gpio[13]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 14	gpio[14]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 15	gpio[15]	LVCMOS 3.3V	slow	enabled	inout
GPIO	MIO 47	gpio[47]	LVCMOS 1.8V	slow	enabled	inout
GPIO	MIO 51	gpio[51]	LVCMOS 1.8V	slow	enabled	inout
☐ EMIO GPIO (Width)						

Figure 9 – MicroZed GPIO Connections

To see what peripherals can be mapped to these 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.


Re-customize IP

ZYNQ7 Processing System (5.3)

Documentation Presets IP Location Import XPS Settings

Page Navigator
- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

Peripheral I/O Pins

Search:

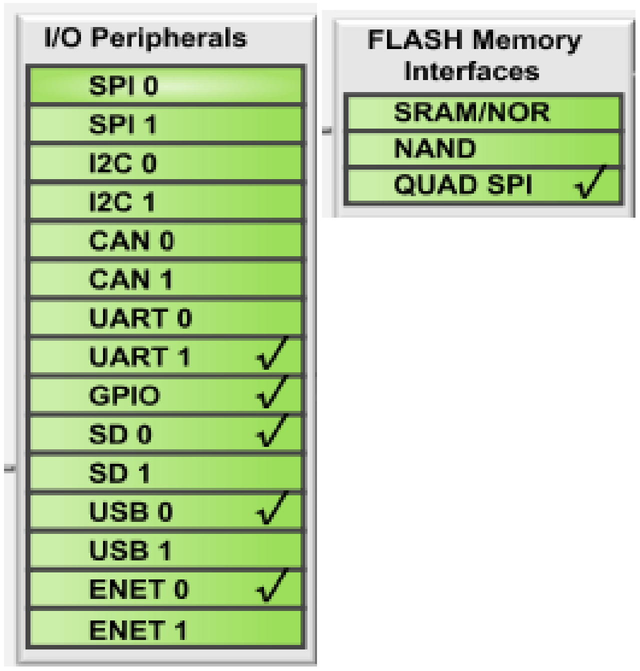
Bank 0 LVCMOS 3.3V   Bank 1 LVCMOS 1.8V

Peripheharals
- [✓] Quad SPI Flash
- [ ] SRAM/NOR Flash
- [ ] NAND Flash
- [✓] Ethernet 0
- [ ] Ethernet 1
- [✓] USB 0
- [ ] USB 1
- [✓] SD 0
- [ ] SD 1
- [ ] SP 0
- [ ] SP 1
- [ ] UART 0
- [✓] UART 1
- [ ] I2C 0
- [ ] I2C 1
- [ ] CAN 0
- [ ] CAN 1
- [ ] TTC0
- [ ] TTC1
- [ ] SWD
- [ ] PJTAG Interface
- [ ] Test and Debug Trace
- [✓] GPIO MIO
- [ ] GPIO EMIO

Summary Report

OK   Cancel

Figure 10 - All MIO Connections (MicroZed)


I/O Peripherals

SPI 0	
SPI 1	
I2C 0	
I2C 1	
CAN 0	
CAN 1	
UART 0	
UART 1	✓
GPIO	✓
SD 0	✓
SD 1	
USB 0	✓
USB 1	
ENET 0	✓
ENET 1	

FLASH Memory Interfaces

SRAM/NOR	
NAND	
QUAD SPI	✓

Figure 11 - Final I/O Peripheral Connections

Questions:

Answer the following questions:

- *Why is EMIO <u>not</u> an option for connecting USB 0?*

____

____

- *Which other peripherals would you expect <u>not</u> to have EMIO options? Why?*

____

____

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

____

- *12 peripherals can be mapped to PS Pmod. Find one that cannot?*

____

- *Is Processor JTAG available on ZedBoard? If so, how do I access it?*

____

- *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.

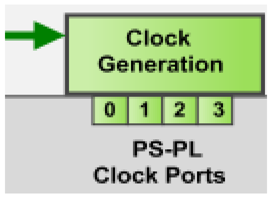
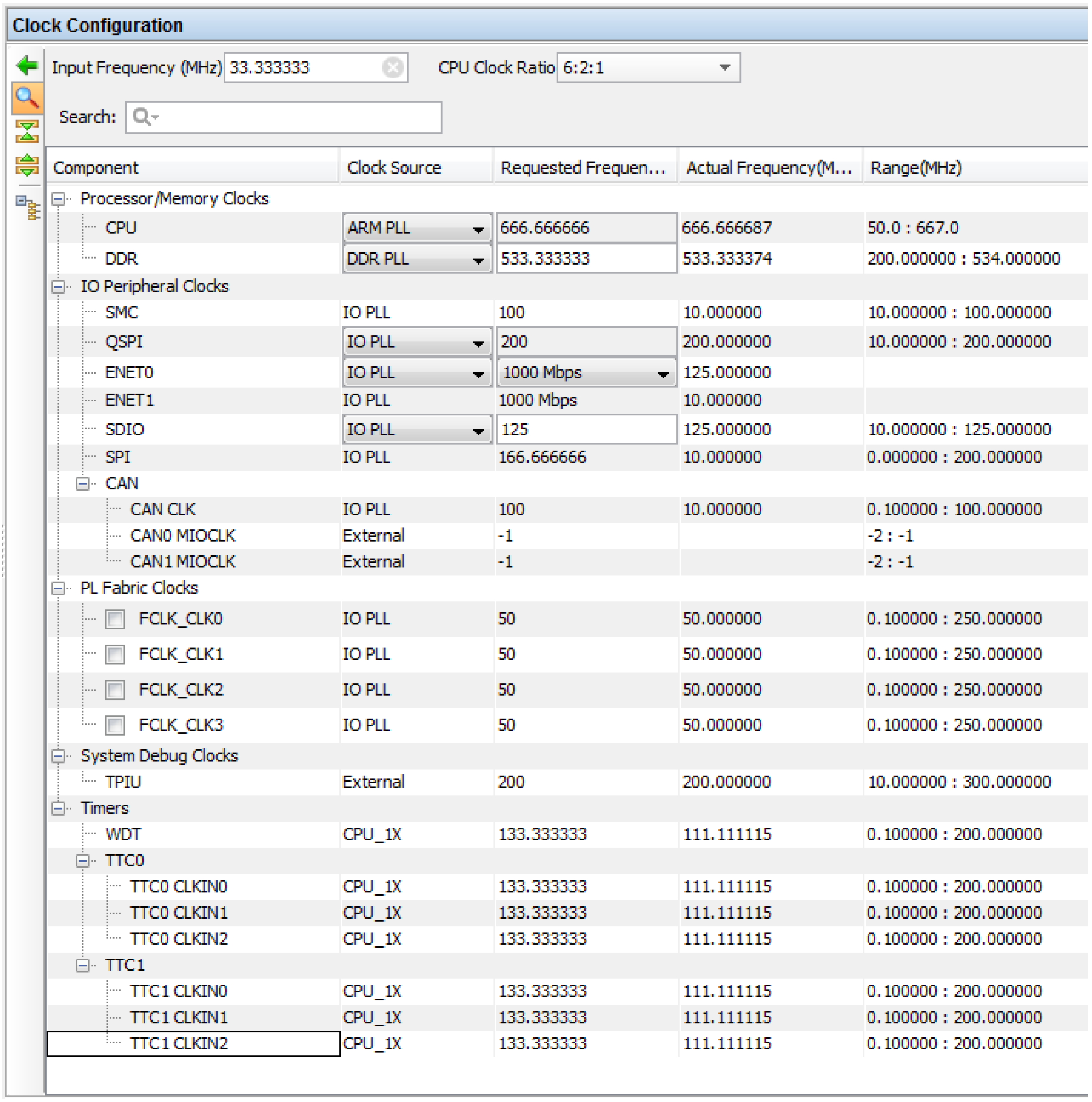


Figure 12 – PS Configuration Clock Generation

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



Clock Configuration

Input Frequency (MHz) 33.333333   CPU Clock Ratio 6:2:1

Search:

Component	Clock Source	Requested Frequen...	Actual Frequency(M...	Range(MHz)
Processor/Memory Clocks				
CPU	ARM PLL	666.666666	666.666687	50.0 : 667.0
DDR	DDR PLL	533.333333	533.333374	200.000000 : 534.000000
IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.000000
QSPI	IO PLL	200	200.000000	10.000000 : 200.000000
ENET0	IO PLL	1000 Mbps	125.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	125	125.000000	10.000000 : 125.000000
SPI	IO PLL	166.666666	10.000000	0.000000 : 200.000000
CAN				
CAN CLK	IO PLL	100	10.000000	0.100000 : 100.000000
CAN0 MIOCLK	External	-1		-2 : -1
CAN1 MIOCLK	External	-1		-2 : -1
PL Fabric Clocks				
☐ FCLK_CLK0	IO PLL	50	50.000000	0.100000 : 250.000000
☐ FCLK_CLK1	IO PLL	50	50.000000	0.100000 : 250.000000
☐ FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 250.000000
☐ FCLK_CLK3	IO PLL	50	50.000000	0.100000 : 250.000000
System Debug Clocks				
TPIU	External	200	200.000000	10.000000 : 300.000000
Timers				
WDT	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0				
TTC0 CLKIN0	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0 CLKIN1	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0 CLKIN2	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC1				
TTC1 CLKIN0	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC1 CLKIN1	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC1 CLKIN2	CPU_1X	133.333333	111.111115	0.100000 : 200.000000

Figure 13 – 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 33.3333 MHz on MicroZed and ZedBoard.

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.

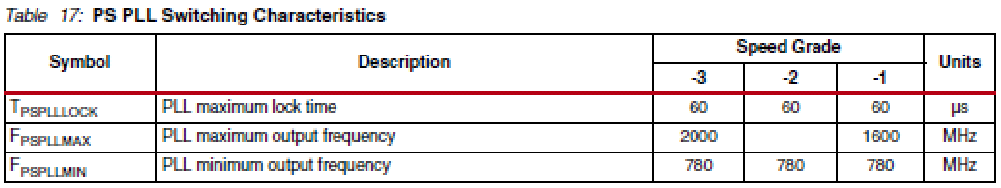


Table 17: **PS PLL Switching Characteristics**

Symbol	Description	Speed Grade			Units
		-3	-2	-1	
$T_{PSPLLLOCK}$	PLL maximum lock time	60	60	60	µs
$F_{PSPLLMAX}$	PLL maximum output frequency	2000		1600	MHz
$F_{PSPLLMIN}$	PLL minimum output frequency	780	780	780	MHz

Figure 14 – 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 = 33.33333 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.

4. 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. The most dominant one in this design is the Ethernet. Ethernet functionality is dependent on having an accurate clock. Set the ENET0 to **1000 Mbps**.

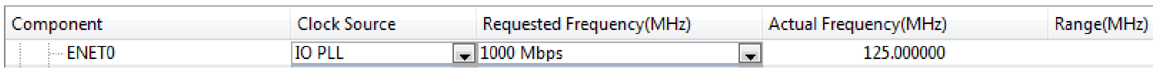
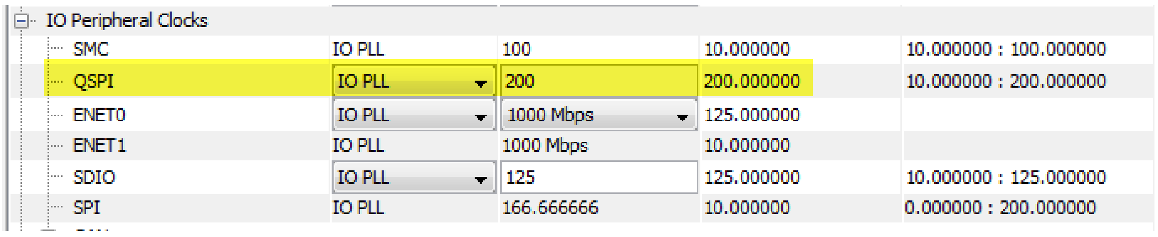

Component	Clock Source	Requested Frequency(MHz)	Actual Frequency(MHz)	Range(MHz)
ENET0	IO PLL	1000 Mbps	125.000000	

Figure 15 – ENET0 Set to 1000 Mbps

Notice that this results in an Actual Frequency of 125 MHz. If you were to work through the math, you would find that the I/O PLL output frequency must either be 1000 MHz (33.333 MHz * 30) or 1600 MHz (33.333 MHz * 45). In this situation, the I/O PLL gets set to 1000 MHz.

5. 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**.

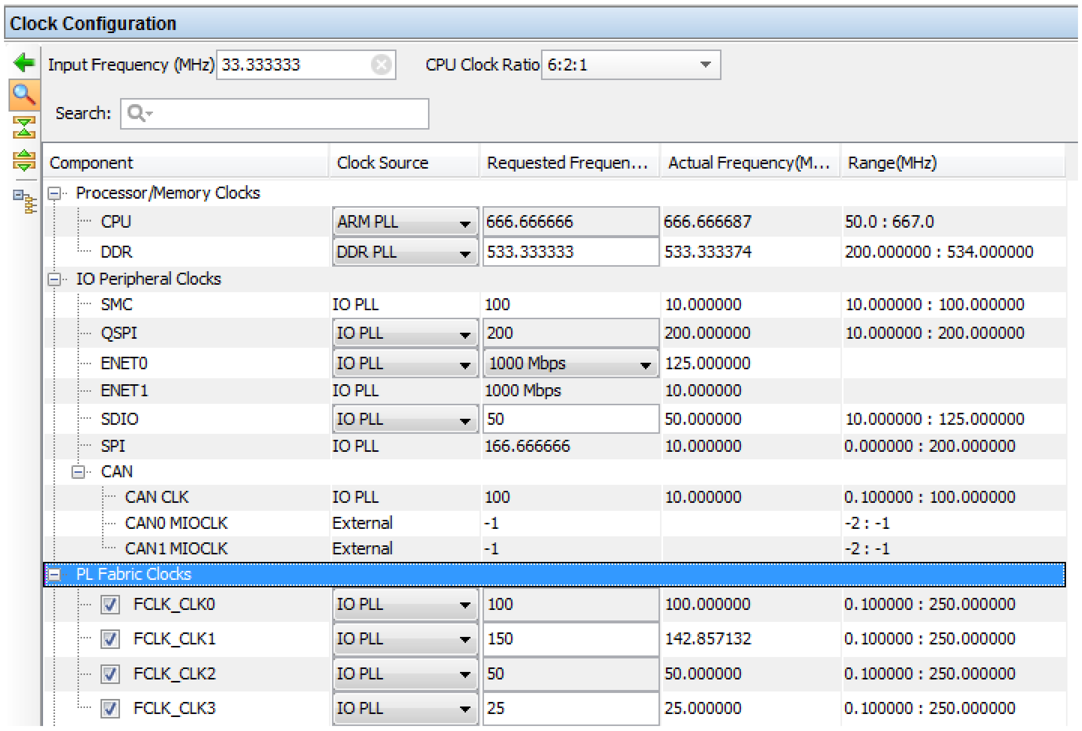


IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.000000
QSPI	IO PLL	200	200.000000	10.000000 : 200.000000
ENET0	IO PLL	1000 Mbps	125.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	125	125.000000	10.000000 : 125.000000
SPI	IO PLL	166.666666	10.000000	0.000000 : 200.000000

Figure 16 – QSPI Requested Frequency Set to 200 MHz

6. We have one more IO Peripheral Clocks to investigate. Set the **SDIO** to **50 MHz.**

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



Clock Configuration

Input Frequency (MHz) 33.333333 CPU Clock Ratio 6:2:1

Search:

Component	Clock Source	Requested Frequen...	Actual Frequency(M...	Range(MHz)
Processor/Memory Clocks				
CPU	ARM PLL	666.666666	666.666687	50.0 : 667.0
DDR	DDR PLL	533.333333	533.333374	200.000000 : 534.000000
IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.000000
QSPI	IO PLL	200	200.000000	10.000000 : 200.000000
ENET0	IO PLL	1000 Mbps	125.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	50	50.000000	10.000000 : 125.000000
SPI	IO PLL	166.666666	10.000000	0.000000 : 200.000000
CAN				
CAN CLK	IO PLL	100	10.000000	0.100000 : 100.000000
CAN0 MIOCLK	External	-1		-2 : -1
CAN1 MIOCLK	External	-1		-2 : -1
PL Fabric Clocks				
✓ FCLK_CLK0	IO PLL	100	100.000000	0.100000 : 250.000000
✓ FCLK_CLK1	IO PLL	150	142.857132	0.100000 : 250.000000
✓ FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 250.000000
✓ FCLK_CLK3	IO PLL	25	25.000000	0.100000 : 250.000000

Figure 17 –Clock Settings

Question:

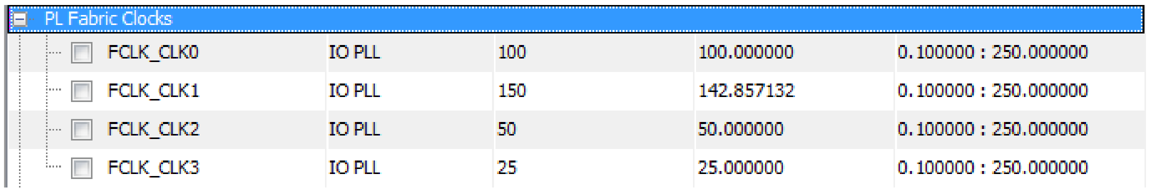
Answer the following question:

- *Notice that the FCLK_CLK1 Actual Frequency doesn't match the Requested Frequency. Why is this?*

____

____

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



PL Fabric Clocks

☐ FCLK_CLK0	IO PLL	100	100.000000	0.100000 : 250.000000
☐ FCLK_CLK1	IO PLL	150	142.857132	0.100000 : 250.000000
☐ FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 250.000000
☐ FCLK_CLK3	IO PLL	25	25.000000	0.100000 : 250.000000

Figure 18 - Clock Configuration

9. Close the Zynq Block Design by clicking **OK**.

10. **Validate** (F6) the Block Design.

11. **Save** the Block Design.

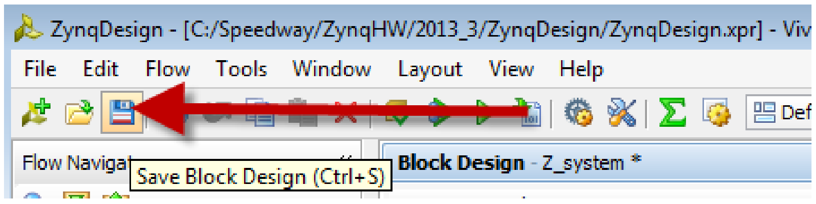

ZynqDesign - [C:/Speedway/ZynqHW/2013_3/ZynqDesign/ZynqDesign.xpr] - Viv
File
Edit
Flow
Tools
Window
Layout
View
Help
Def
Flow Navigat
Save Block Design (Ctrl+S)
Block Design - Z_system *

Figure 19 - Save Block Design

12. Select **Generate Bitstream**, this will relaunch synthesis and implementation, click **OK** to accept. This process should take only a few minutes.

13. 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:

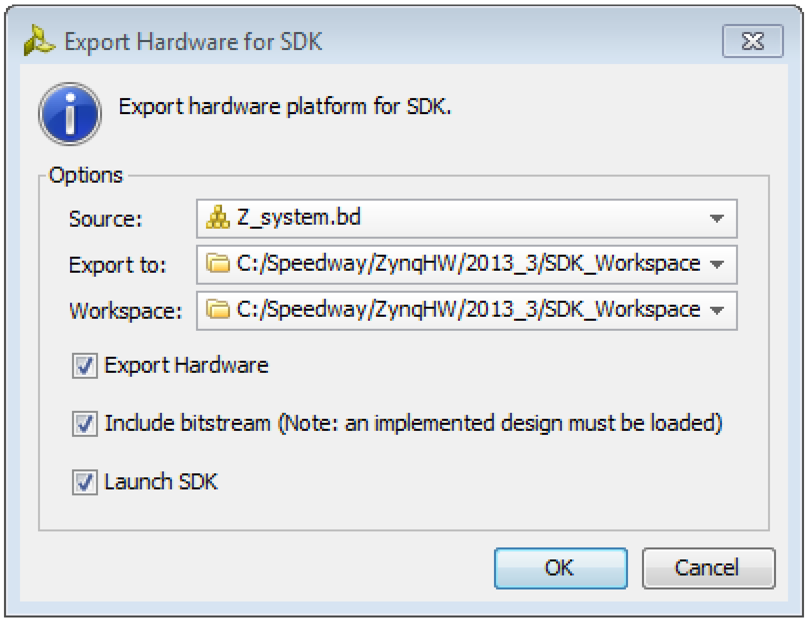
Export to SDK. Generate and run the Memory and Peripheral test applications.

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 for SDK...**

3. The *Export Hardware for SDK* dialog box opens. By default, the **Export Hardware** check box is checked. Check the **Launch SDK** check box. Select the SDK Workspace directory again: *C:\Speedway\ZynqHW\2013_3\SDK_Workspace* Click **OK**. Remember, it won't export the bitstream unless an implemented design is open.


Export Hardware for SDK

Export hardware platform for SDK.

Options

Source: Z_system.bd

Export to: C:/Speedway/ZynqHW/2013_3/SDK_Workspace

Workspace: C:/Speedway/ZynqHW/2013_3/SDK_Workspace

☑ Export Hardware

☑ Include bitstream (Note: an implemented design must be loaded)

☑ Launch SDK

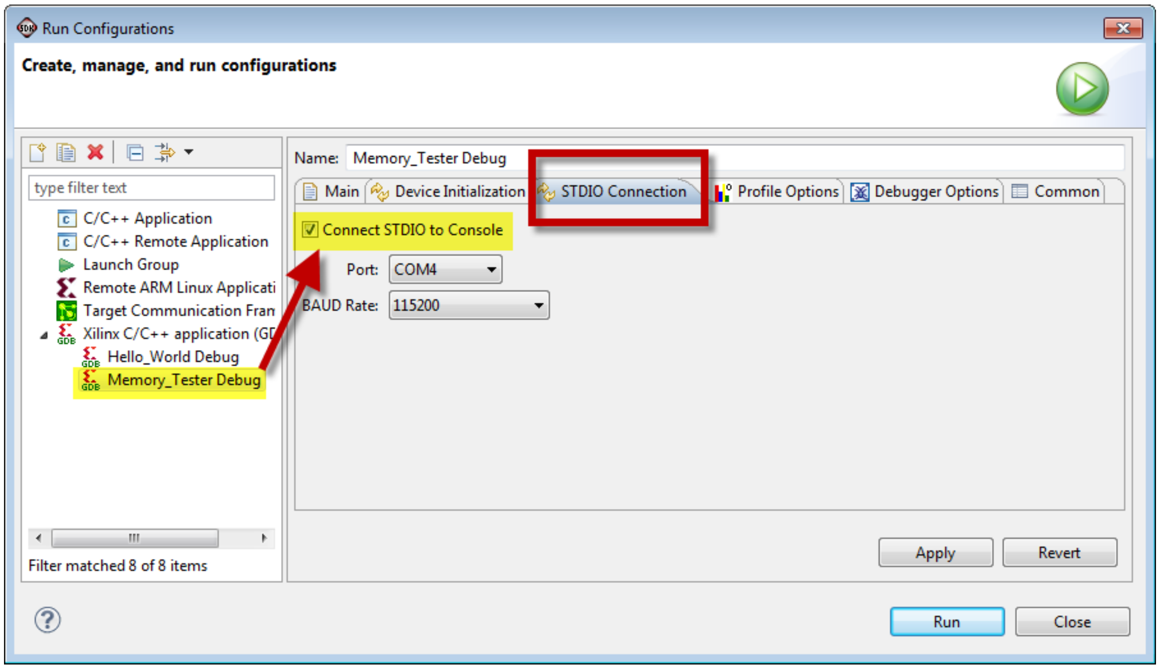
Figure 20 - Export Hardware to SDK

4. Vivado should recognize that the HW platform has changed. Click **Yes** to overwrite existing platform.

Once the HW platform updates, the BSP will rebuild, which then causes Hello World to rebuild. This may take a minute or two.

5. As a sanity check, you can re-run Hello World with the previous Run Configuration that you defined. Make sure your board is connected and powered. Right-click on hello_world_0. Select **Run As → Run Configurations**. Select **Hello_World Debug** and then click **Run**.

6. 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**, then (New Configuration)
   - Set STDIO Connection to the USB-UART COM port, 115200 baud
   - Click **Apply** and then **Run**



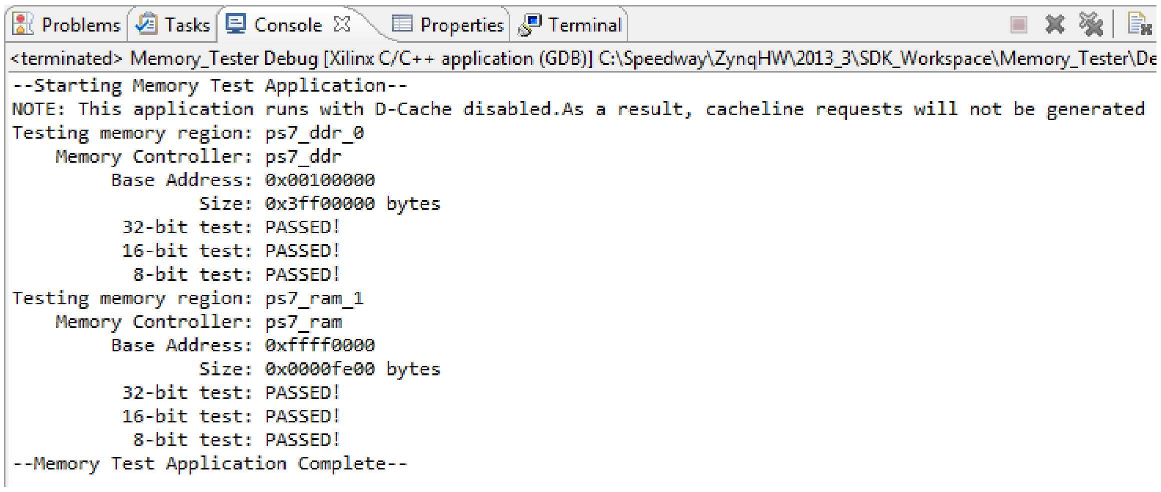
Name: Memory_Tester Debug

☑ Connect STDIO to Console

Port: COM4

BAUD Rate: 115200

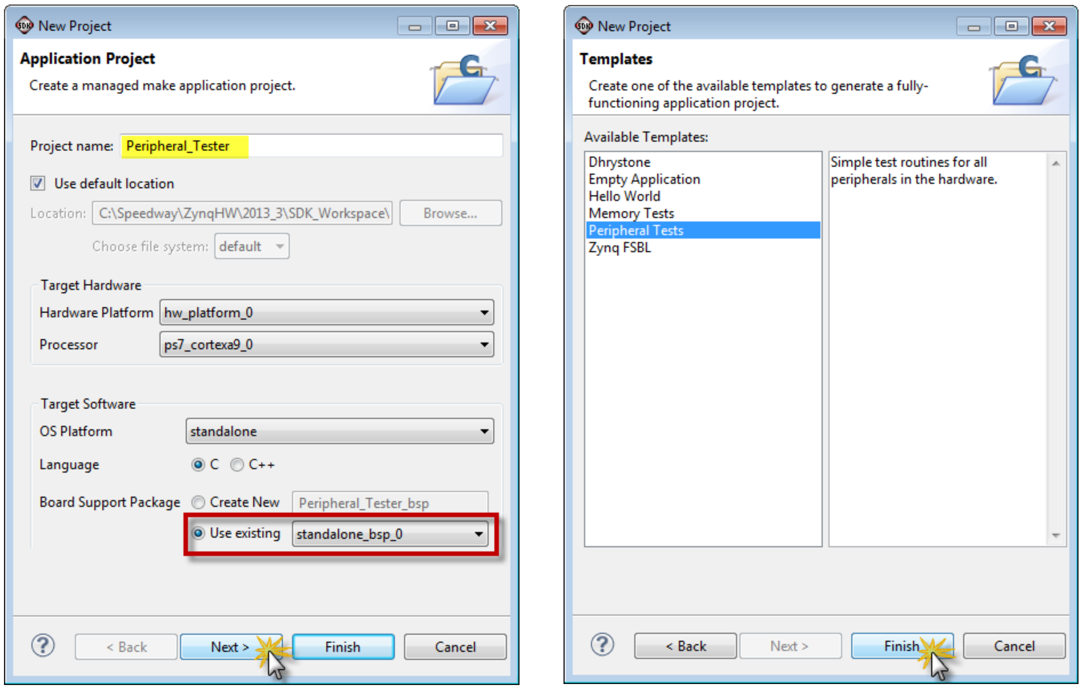
Figure 21 – New Xilinx Application Run Configuration: Memory Tests



```
<terminated> Memory_Tester Debug [Xilinx C/C++ application (GDB)] C:\Speedway\ZynqHW\2013_3\SDK_Workspace\Memory_Tester\De
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As a result, cacheline requests will not be generated
Testing memory region: ps7_ddr_0
    Memory Controller: ps7_ddr
         Base Address: 0x00100000
                 Size: 0x3ff00000 bytes
          32-bit test: PASSED!
          16-bit test: PASSED!
           8-bit test: PASSED!
Testing memory region: ps7_ram_1
    Memory Controller: ps7_ram
         Base Address: 0xffff0000
                 Size: 0x0000fe00 bytes
          32-bit test: PASSED!
          16-bit test: PASSED!
           8-bit test: PASSED!
--Memory Test Application Complete--
```

Figure 22 – Memory Test Results

7. Now repeat the same thing for **Peripheral Tests**.



Application Project

Create a managed make application project.

Project name: Peripheral_Tester

Use default location [✓]

Location: C:\Speedway\ZynqHW\2013_3\SDK_Workspace\

Choose file system: default

Target Hardware

Hardware Platform: hw_platform_0

Processor: ps7_cortexa9_0

Target Software

OS Platform: standalone

Language: [✓ C] C++

Board Support Package: Create New Peripheral_Tester_bsp [✓ Use existing] standalone_bsp_0

Templates

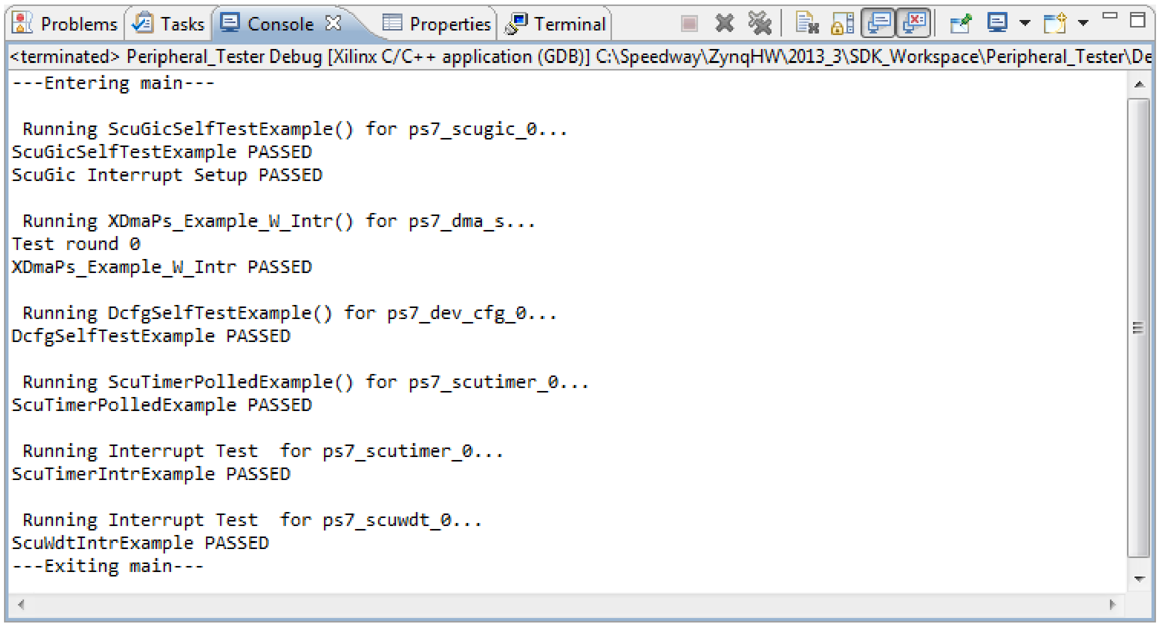
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

Dhrystone
Empty Application
Hello World
Memory Tests
Peripheral Tests [✓]
Zynq FSBL

Simple test routines for all peripherals in the hardware.

Figure 23 – New Xilinx Application Project: Peripheral Tests



```
<terminated> Peripheral_Tester Debug [Xilinx C/C++ application (GDB)] C:\Speedway\ZynqHW\2013_3\SDK_Workspace\Peripheral_Tester\De
---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 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 24 – Peripheral Tests Results

8. **Close** SDK.
9. **Close** Vivado.

Questions:

Answer the following questions:

- *Look again at Figure 22 – Memory Test Results. 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

Resources

www.microzed.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 Flash devices. 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.

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

This is a control pin output to enable/disable power to the SD Card slot.

- *12 peripherals can be mapped to PS Pmod. Find one that cannot?*

SP0 and Trace (clk, ctl, and d[3:0]; d[15:4] would have to go to EMIO).

These peripherals can be mapped to the PS Pmod: Dual Quad, SPI, SD 1, UART 0, I2C 0, I2C 1, SPI 1, CAN 0, CAN 1, Watchdog, PJTAG, GPIO.

- *Is Processor JTAG available on ZedBoard? If so, how do I access it?*

Yes, it is available. You can access ARM JTAG in cascaded JTAG mode through the shared JTAG. Independent Processor JTAG is also available through the PS Pmod. To do this will require the following steps:

- **Set boot_mode[3] to 1 for Independent JTAG before power up.**
- **A small piece of configuration code must execute to configure MIO[10-13] as PJTAG.**
- **Use the PS Pmod to access PJTAG. Avnet will offer a small Pmod with the correct mechanical connections to do this.**

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

Hardware/software simultaneous debug. The standard JTAG port is used for Vivado Analyzer while the PS Pmod PJTAG is 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 2

- *Notice that the FCLK_CLK1 Actual Frequency doesn't match the Requested Frequency. Why is this?*

The PLLs have a limited number of M/N ratios to generate the various frequencies. With the ENET set to 1000 Mbps, we have a hard requirement for a 125 MHz output clock. Starting with a 33.3333 MHz reference clock, the PLL gets set to 1 GHz, then divided by 8 to generate 125 MHz. With the PLL set to 1 GHz, you are limited now by available dividers. When 150 MHz is requested, the closest divider is 7. 1 GHz / 7 = 142.857 MHz.

Experiment 3

- *Look again at Figure 22 – 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 TRM.