

Overview

This MicroZed System On Module (SOM) and FMC Carrier Card (FMC-CC) tutorial offers system developers an example of how to:

1. Produce files for the microSD card containing the following components needed to boot Linux on MicroZed and launch a FMC-CC User I/O Application Example:
 - a. Zynq First Stage Boot Loader (FSBL)
 - b. Zynq Programmable Logic (PL) Bitstream
 - c. U-Boot Second Stage Boot Loader (SSBL)
 - d. Linux kernel binary
 - e. Linux devicetree
 - f. Linux Root File System (RFS)
2. Copy the appropriate files to the microSD card.
3. Boot MicroZed connected to the FMC-CC using the microSD card and interact with the User I/O Application.

Objectives

When this tutorial is complete, you will be able to:

- Create a new project in Vivado, targeted at the MicroZed 7010 SOM
- Create a block based design to insert an ARM processor core
- Connect GPIO pins to FMC-CC hardware and apply appropriate pin constraints
- Build and export the hardware platform
- Create a Linux application project in SDK

Experiment Setup

Software

The software used to test this reference design is:

- Windows-7 64-bit
- Xilinx Vivado Design Suite 2013.3
- Silicon Labs CP210x USB-to-UART Bridge Driver
 - Check “Silicon Labs CP210x USB-to-UART Setup Guide” under Documentation section on MicroZed.org for detailed instructions on installing and configuring this driver
 - www.microzed.org → Documentation → MicroZed → Silicon Labs CP210x USB-to-UART Setup Guide
- Tera Term (Exact version used for this tutorial is V4.79)
- Adobe Reader for viewing PDF content
 - Adobe Reader X or later recommended

Hardware

The hardware setup used to test this reference design includes:

- Win-7 PC with a recommended 1.6 GB RAM available for the Xilinx tools to complete a XC7Z010 design¹
- Available SD card slot on PC or external USB-based SD card reader
- Avnet MicroZed (AES-Z7MB-7Z010-G) – *included in MicroZed Kit*
- USB cable (Type A to Micro-USB Type B) – *included in MicroZed Kit*
- Avnet MicroZed (AES-MBCC-FMC-G) – *included in MicroZed FMC-CC Kit*
- 12V AC/DC Adapter – *included in MicroZed FMC-CC Kit*
- JTAG Programming Cable (Xilinx Platform Cable, Digilent HS1 or HS2 cable)
 - If you don't already have a JTAG Cable, Avnet recommends the Digilent HS2 Cable
 - <http://www.em.avnet.com/en-us/design/drc/Pages/Digilent-JTAG-HS2-Programming-Cable.aspx>

¹ Refer to www.xilinx.com/design-tools/vivado/memory.htm

Experiment Files

This tutorial builds upon the concepts and lab activities of the Avnet SpeedWay trainings and workshops. Please refer back to this training material on the MicroZed community website for further information on how to configure the underlying hardware and software platforms.

MicroZed users will need to extract the contents of the accompanying **MicroZed_FMC-CC_Linux_User_IO_2013_3_01-Tutorial.zip** file archive to the following folder:

C:\Avnet\MicroZed

Note: Extracting the archives to the appropriate folder is required in order to complete the tutorial document instructions.

Experiment 1: Create a New Zynq Project in Vivado

The MicroZed Evaluation Kit includes a license voucher for Vivado Design Edition, device-locked to the Z7010 device. Vivado WebPack also supports MicroZed 7010 SOM as a target platform. This experiment will demonstrate the steps to create a new hardware platform project in Vivado.

1. Launch Vivado by selecting the **Start → All Programs → Xilinx Design Tools → Vivado 2013.3 → Vivado 2013.3** item from the Windows Start Menu.
2. Click on **Create New Project**.

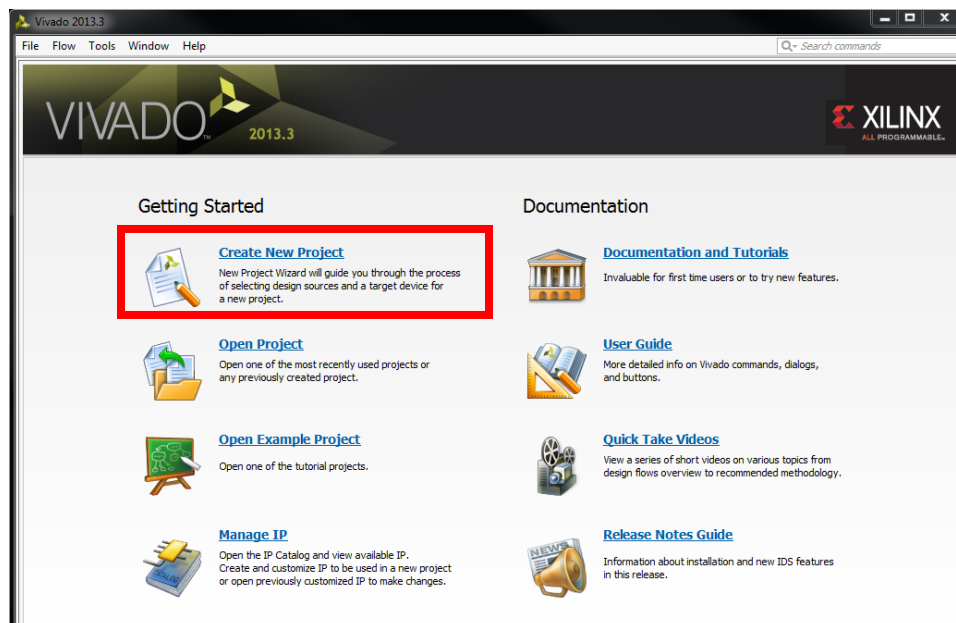


Figure 1 – Vivado Launched

3. Click the **Next >** button.

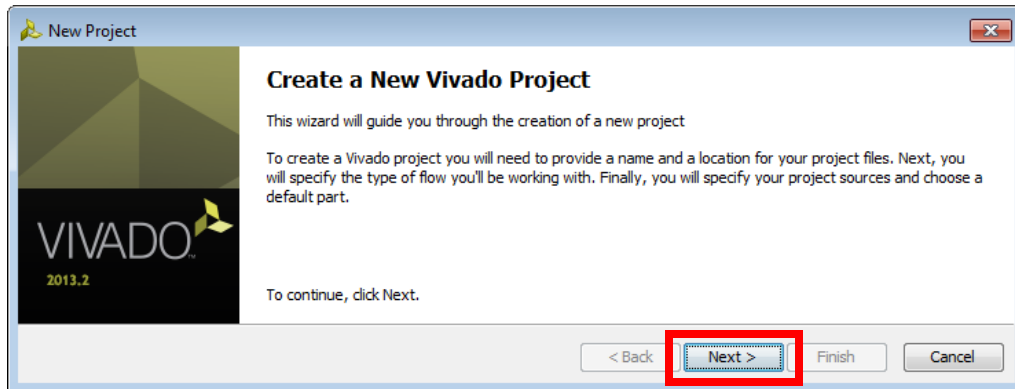



Figure 2 – New Vivado Project Wizard launched

4. Click the browse icon . Browse to set the *Project location* to **C:/Avnet/MicroZed** (or your desired project location) and click **Select**.
5. Set the *Project name* field to **MicroZed_FMC-CC_Linux_User_IO** and verify the **Create project subdirectory** checkbox is selected. Click the **Next >** button.

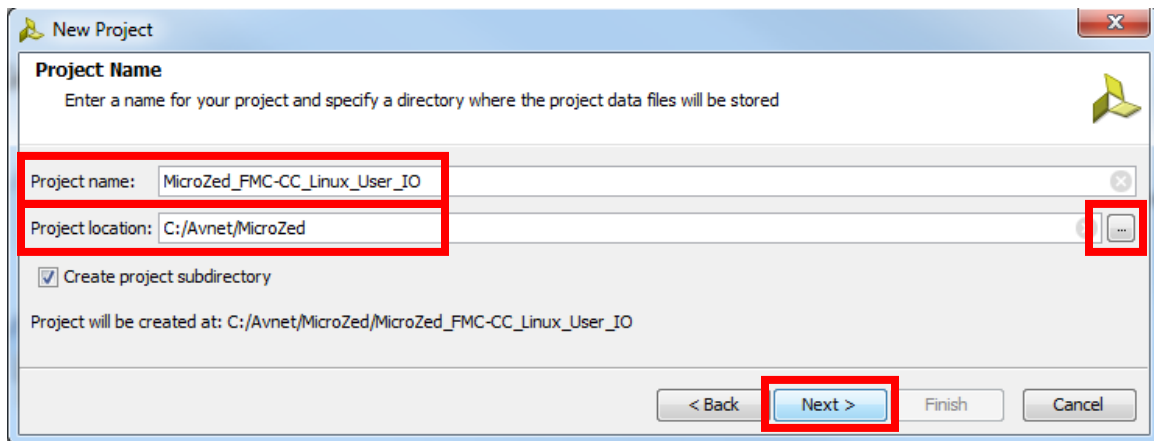


Figure 3 – Set Project Name and Location

6. The project will be RTL based, so leave the radio button for **RTL Project** selected. Since this is a brand new project, check the box for **Do not specify sources at this time**. Click the **Next >** button.

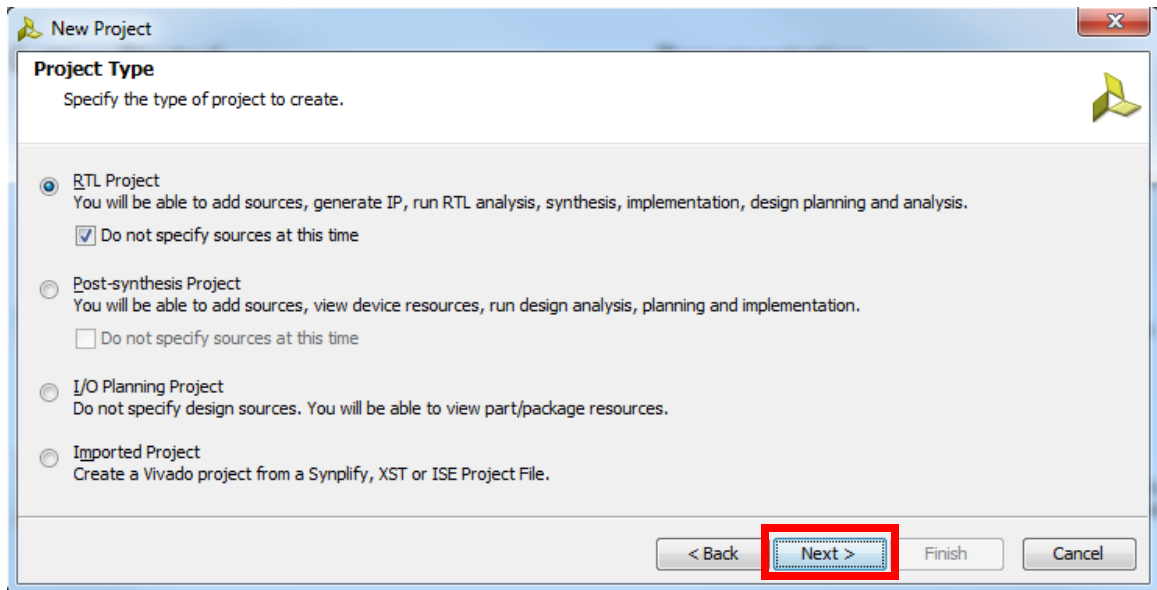


Figure 4 – Set Project Type

Next, the Default Part is selected. This can be done by specifying a specific part or by selecting a board. In Vivado 2013.3 or later, MicroZed will show up in the **Boards** list.

7. In the *Specify* pane, select the **Boards** option.
8. Set the *Board Vendor* to **em.avnet.com** to list Avnet board designs.
9. Single-click the **MicroZed Board**, Rev E. Click the **Next >** button.

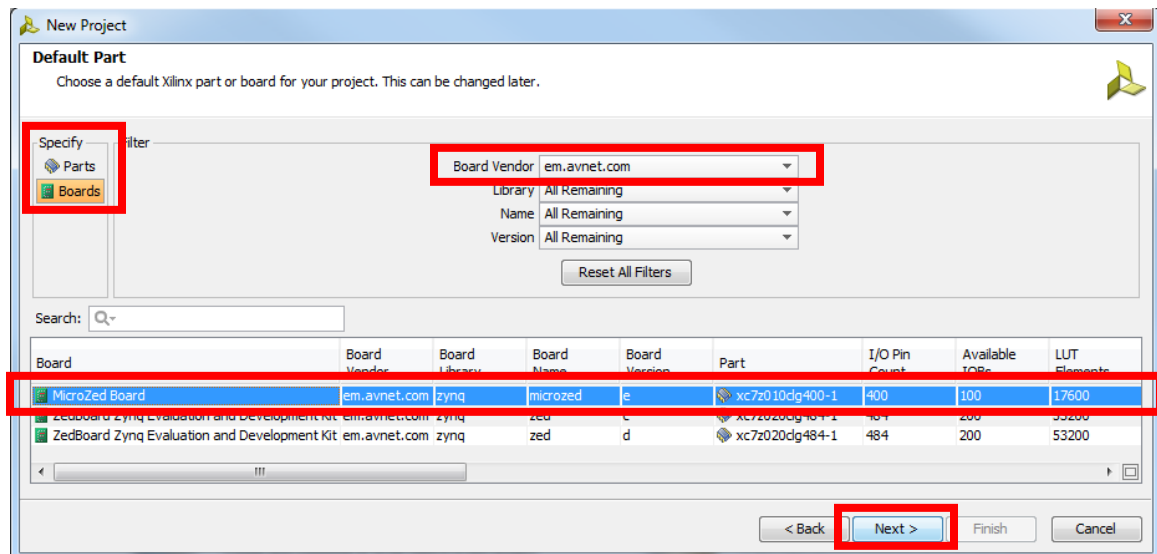


Figure 5 – Select the Target Board = MicroZed Board

10. A project summary is displayed. Click the **Finish** button. The Vivado design cockpit will be displayed.

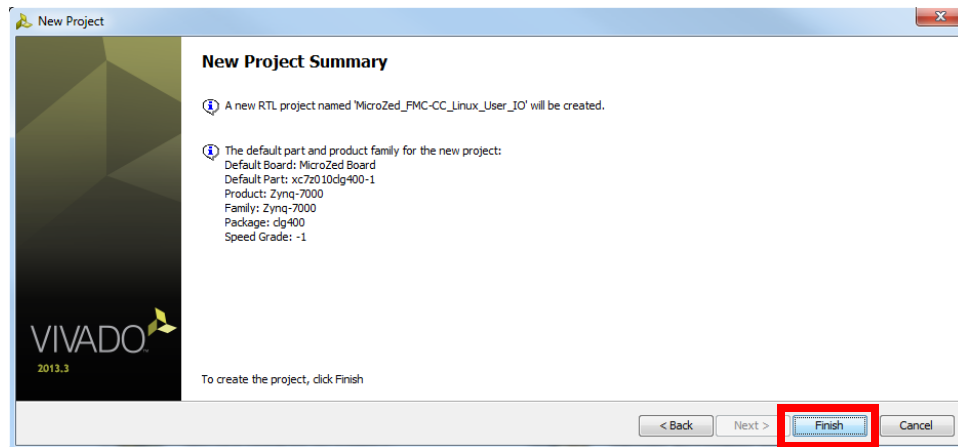


Figure 6 – New Project Summary

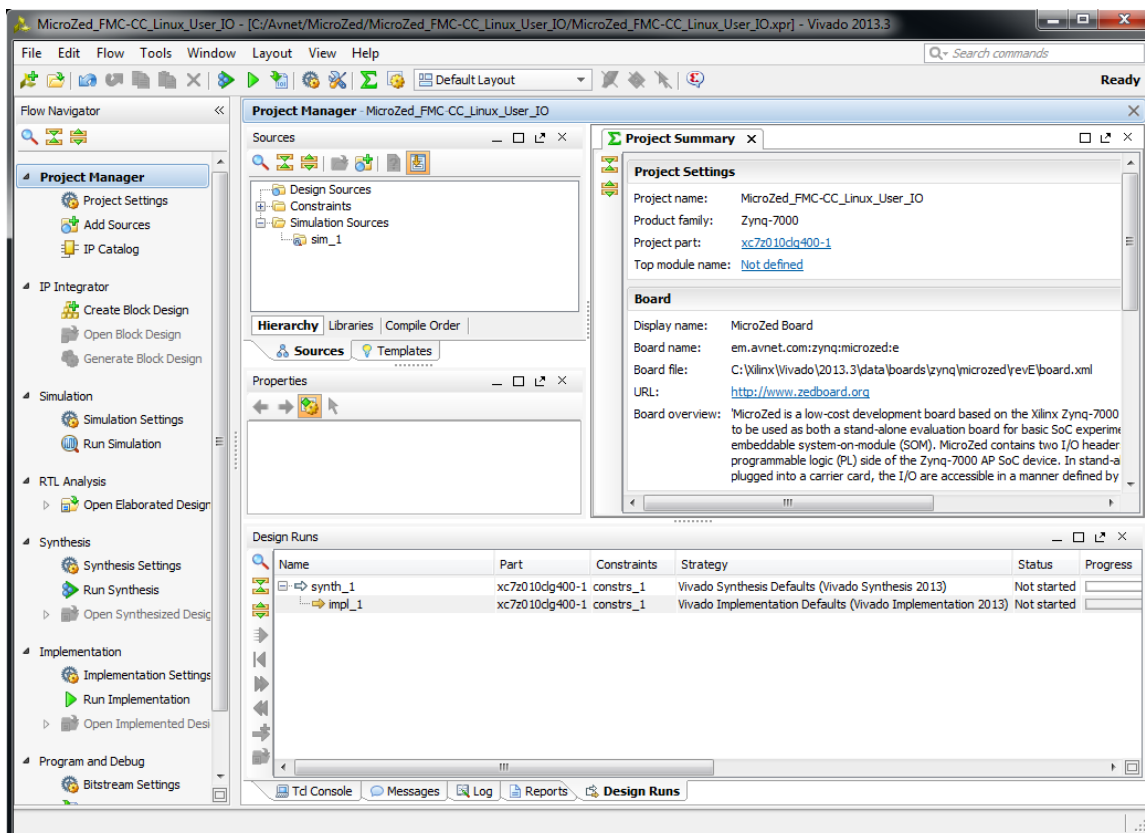


Figure 7 – Vivado Cockpit

Experiment 2: Create and Edit a Block Design

The Vivado project created in the previous experiment is blank. To access the ARM processing system, we will add an embedded source to the Vivado project using IP Integrator. This experiment will demonstrate the steps for creating a basic hardware platform in Vivado with User I/O implemented for the FMC Carrier LEDs and push buttons.

1. The recommended way to add an embedded processor is through the Block Design method via IP Integrator. Select **Create Block Design**.

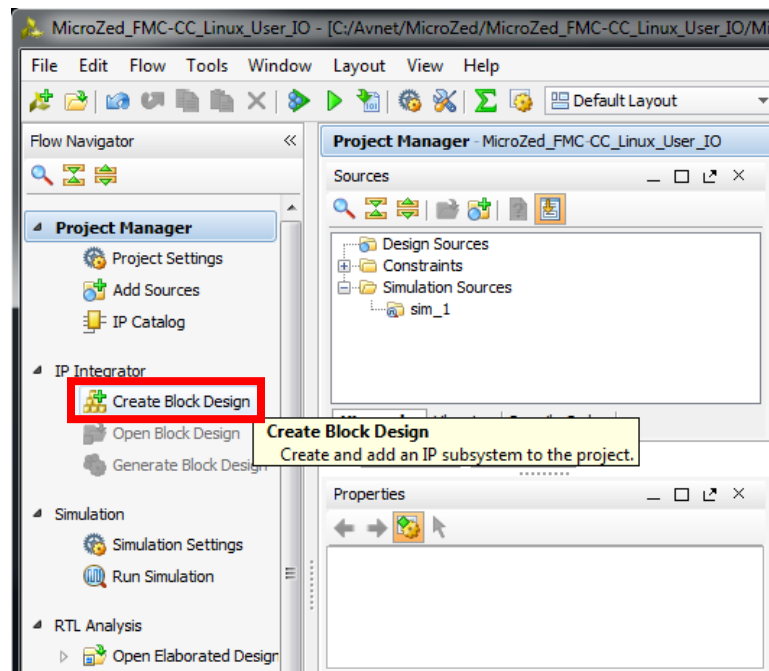


Figure 8 - Create Block Design

2. Give the Block Design a name. The name *design_1* is the default used by Xilinx. Click the **OK** button.

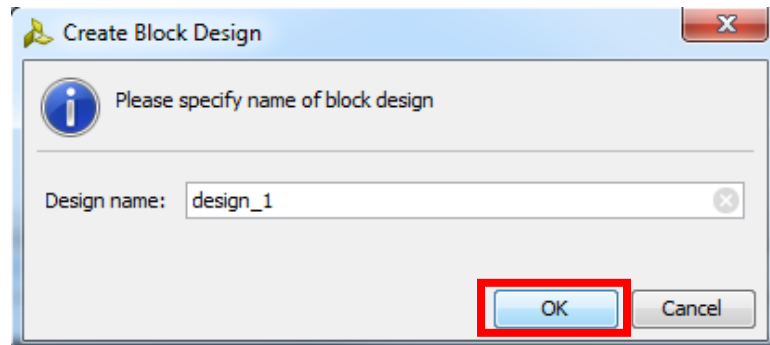



Figure 9 - Block Design Name

3. In the Diagram window, click either the **Add IP** text or icon .

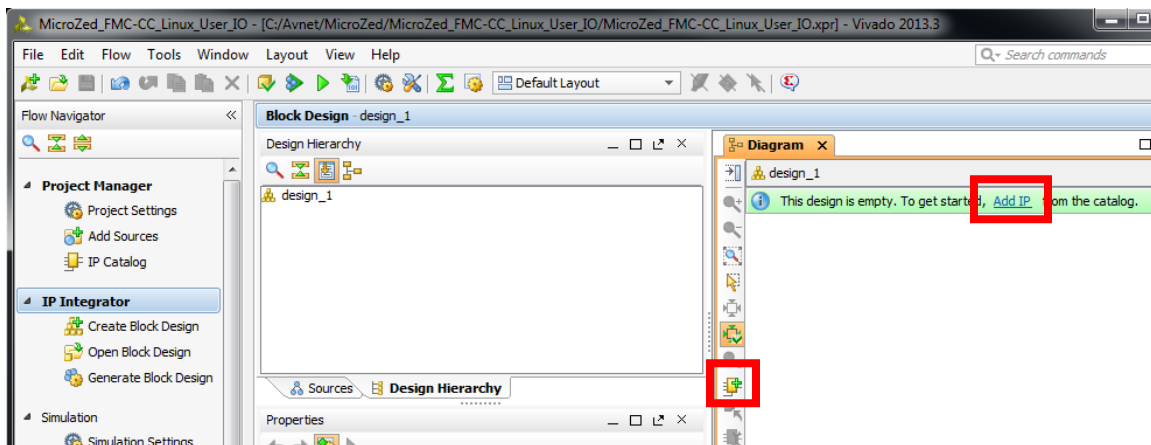


Figure 10 – Add IP to the Block Design

4. The *Add Sources* window opens. Either scroll to the bottom of the listing window or enter **zynq** into the Search field to narrow the listing results. Find the **ZYNQ7 Processing System** IP. Either double-click the IP entry or drag and drop it into the *Diagram* window.

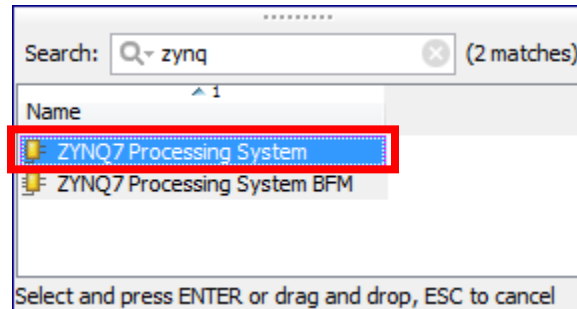


Figure 11 – Add IP Window

The Zynq Processing system will appear in the *Diagram* window. Also a new tab will appear labeled **Address Editor**.

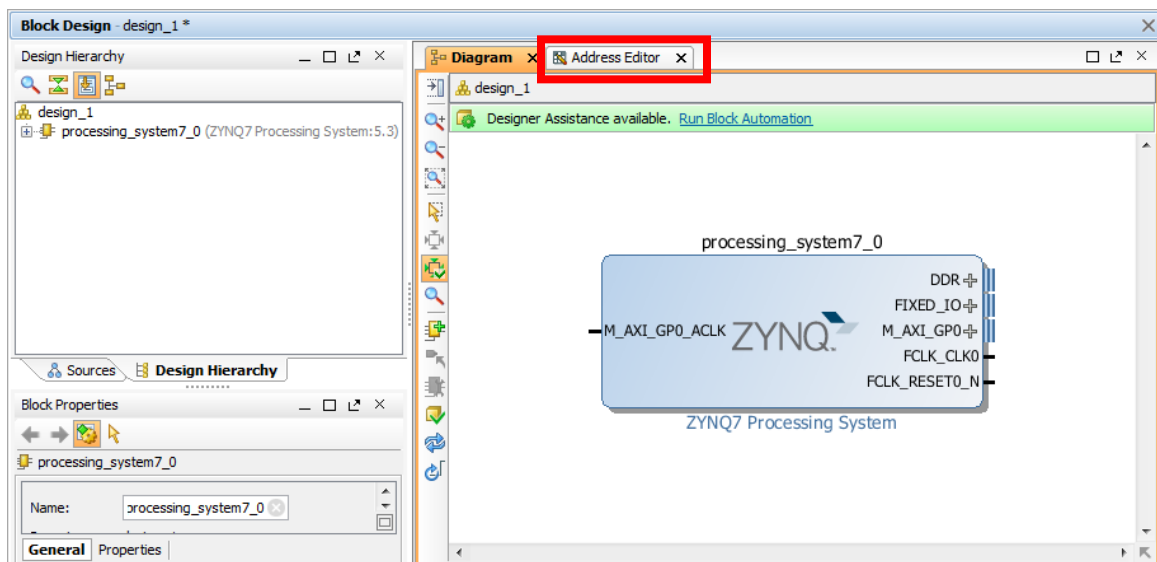



Figure 12 – Updated Block Diagram

5. To view the Zynq PS configuration do one of three things: 1) double-click the **Zynq** IP block; 2) Right-click on the **Zynq** IP block and select **Customize Block**; or 3) Select the **Zynq** IP block and click the  icon. This opens the Zynq Block Design customization window.

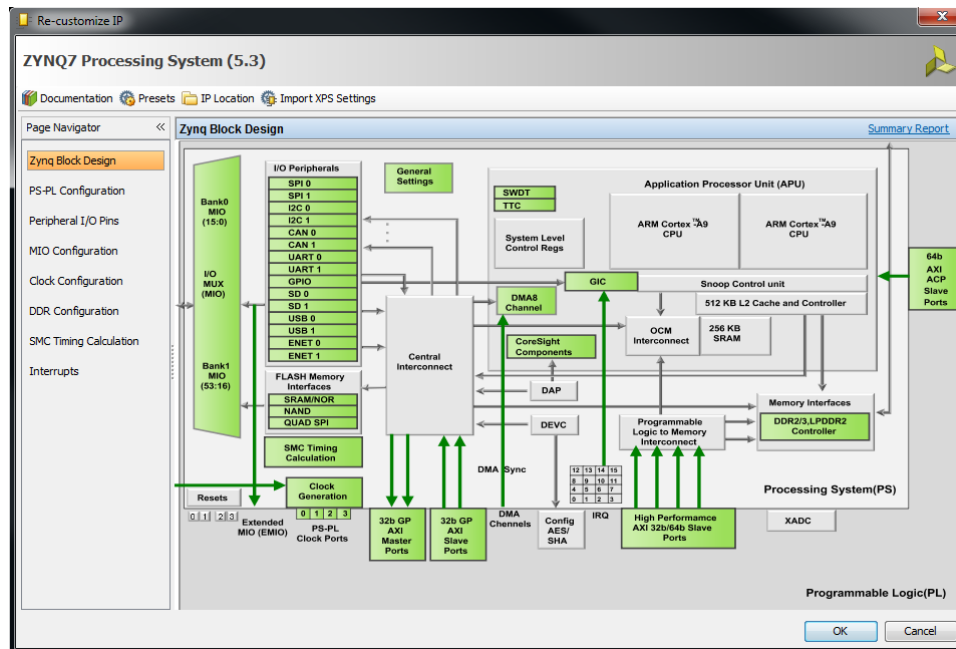


Figure 13 – Zynq Block Design Customization

6. Vivado 2013.3 contains board presets for the MicroZed 7010 SOM.

Click on the **Presets** button and select the **Microzed** option to apply the built-in board presets for the Zynq PS configuration on MicroZed.

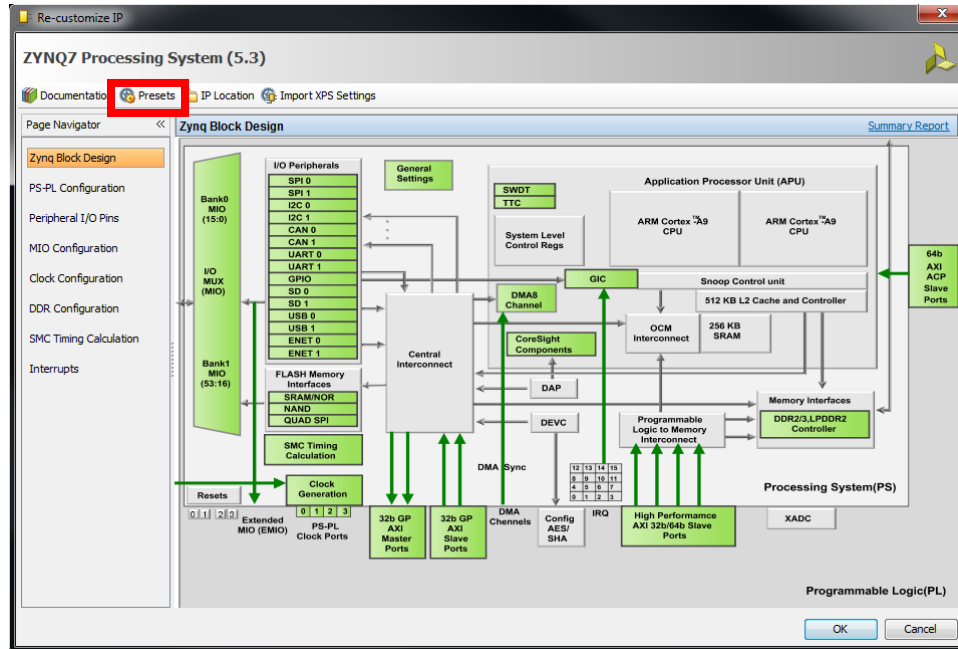


Figure 14 – Applying MicroZed Presets

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

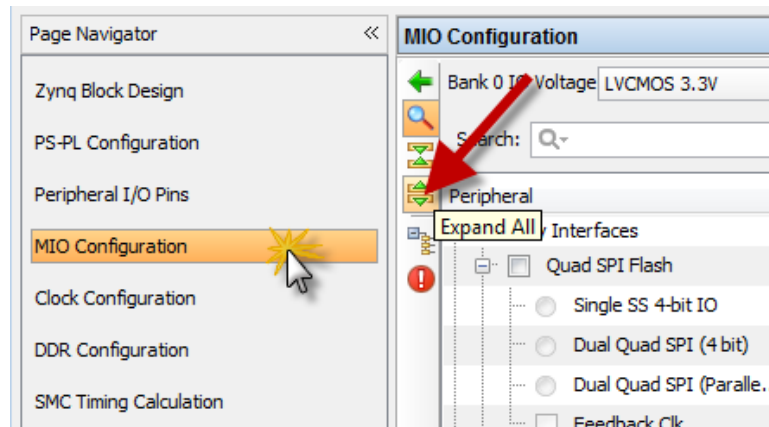


Figure 15 - MIO Configuration

MicroZed features an on-board Spansion Quad SPI Flash device with a feedback clock option which must be enabled for high speed data transfer operations.

8. Verify that the check the box next to **Quad SPI Flash** is selected.
9. Expand the **Quad SPI Flash** under the **Memory Interfaces** tree to see that a **Feedback Clk** option is possible, select the checkbox next to it and verify that the IO setting is set to the **MIO 8** option. Notice that the Quad SPI peripheral has a fixed location of **MIO[1-6, 8]**. Also verify that the **Single SS 4-bit IO** option is selected with **MIO 1 .. 6** settings.

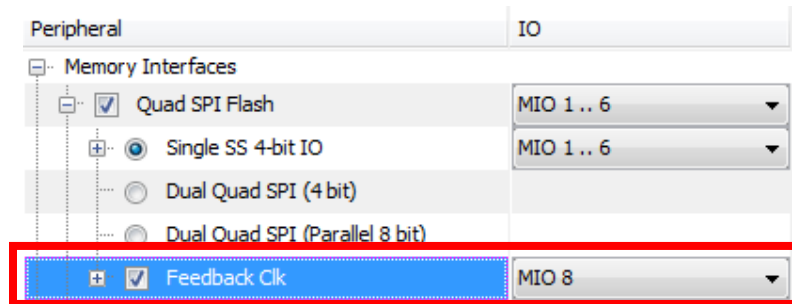


Figure 16 - QSPI Flash Connections

10. Expand the **GPIO** to see that an **EMIO GPIO** option is possible. Select the checkbox next to it and set the width to the **6** option. This EMIO width will cover the IO wires needed for the 4 user LEDs (LED1-LED4) as well as the 2 user push buttons (BTN1 & BTN2) on the FMC Carrier.

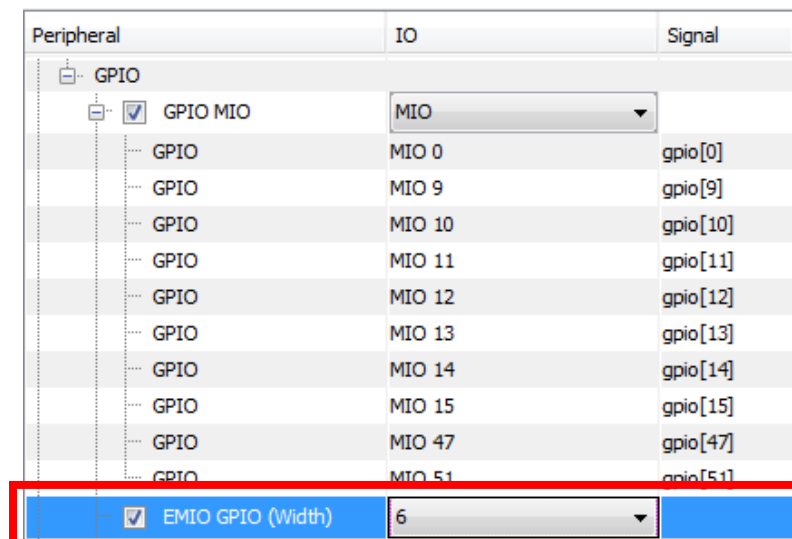


Figure 17 - EMIO Width Setting

11. When done customizing the Zynq IP block, click the **OK** button to close the Zynq IP block customization window.
12. Back in the Vivado Block Design, click the **Run Block Automation** link at the top of the window and select **/processing_system7_0** to connect all block I/O.

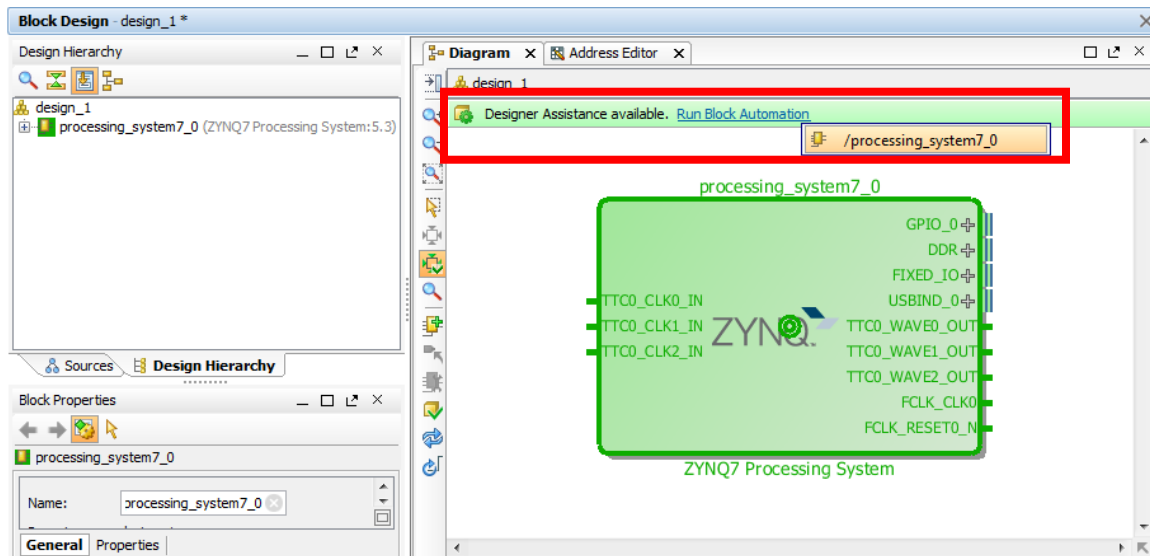


Figure 18 – Launching Designer Assistance Block Automation

13. Notice the block automation wizard has identified two sources of I/O that need to be made external. One is obvious, the **DDR** interface. The other is labeled **FIXED_IO**. FIXED_IO is basically the MIO pin connections. They are labeled FIXED_IO because you cannot change their assignments in this window.

Be sure to de-select the option **Apply Board Preset** before clicking the **OK** button. This will ensure that our work in the previous steps is not overwritten by the re-application of the MicroZed board presets.

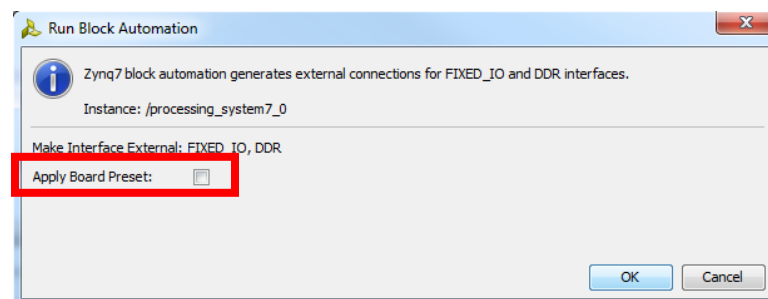


Figure 19 - Run Block Automation

14. You will now see the Zynq block with some external I/O connected. Some additional I/O must be connected in the next step.

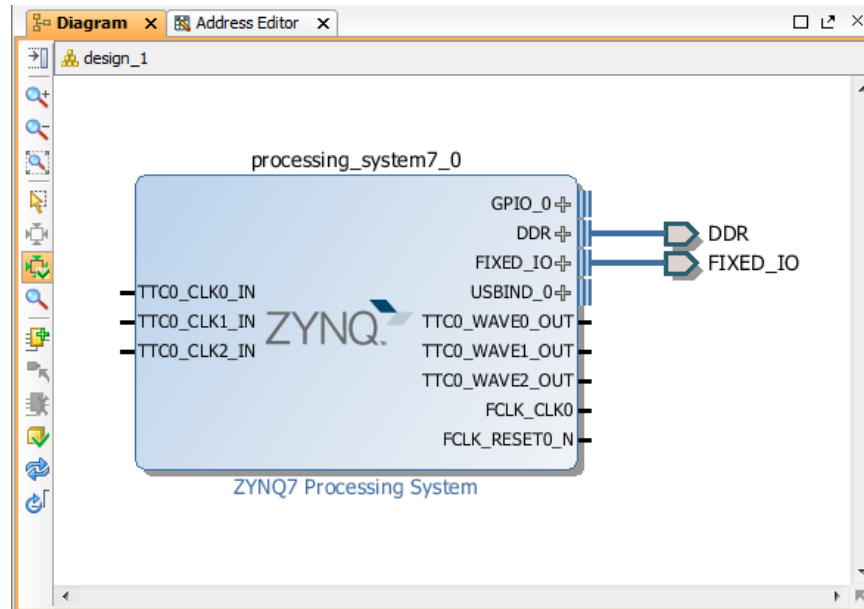


Figure 20 - Zynq Block Diagram with External I/O

15. Make the EMIO from the GPIO external to the Programmable Logic by right clicking on the **GPIO_0** interface bars and selecting the **Make External** option from the menu.

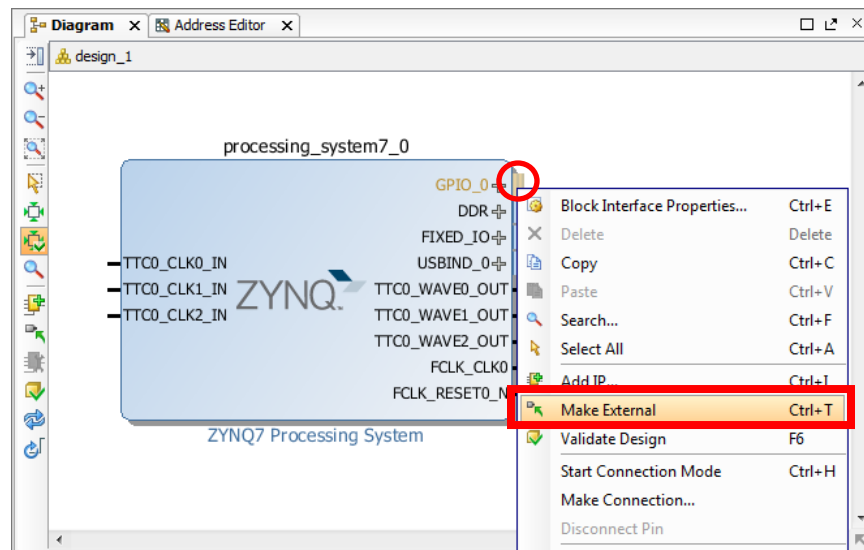


Figure 21 – Making the GPIO_0 Interface External

16. Rename the GPIO_0 external interface to **emio_user** by changing the name in the GPIO_0 properties panel then press Enter to force the changes to take effect. This name matches the wire naming found in the constraints file (added in one of the later steps) and will map the EMIO to corresponding Programmable Logic I/O pin assignments on the Zynq device.

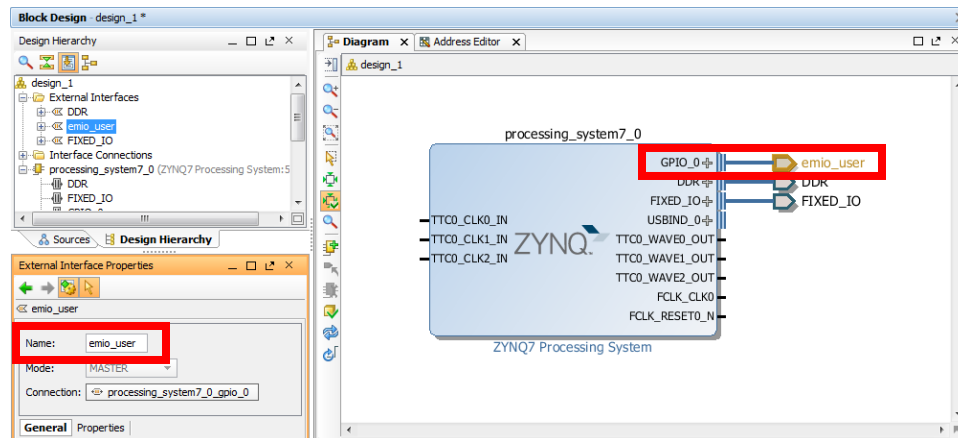



Figure 22 – Renaming the GPIO_0 External Interface

17. Save the block design by clicking the **Save Block Design** icon  or through the **File → Save Block Design** menu item.
18. In the Sources Window, right-click on **System.bd** listing and select the **Create HDL Wrapper** option from the menu.

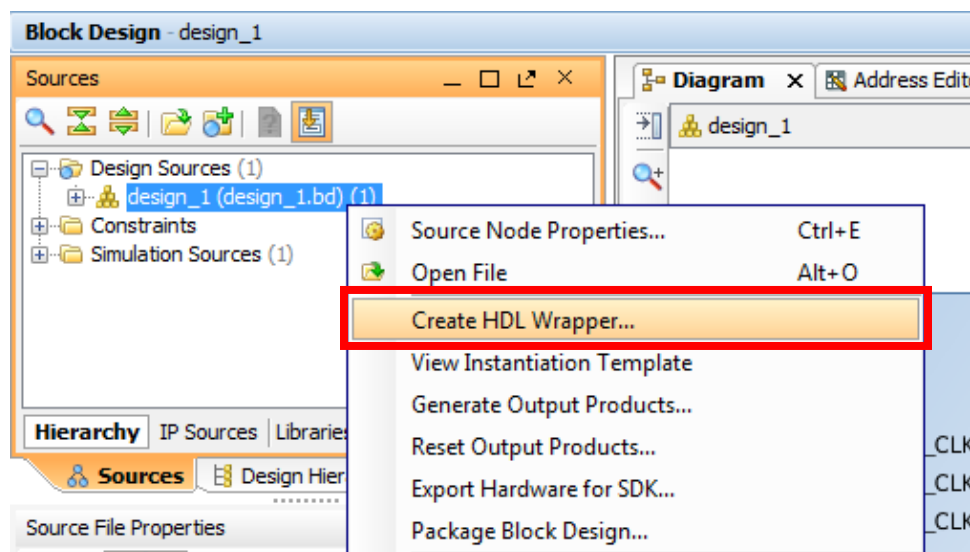


Figure 23 - Create Top Level HDL Wrapper

19. In the **Create HDL Wrapper** dialog, select the **Let Vivado manage wrapper and auto-update** option then click the **OK** button.

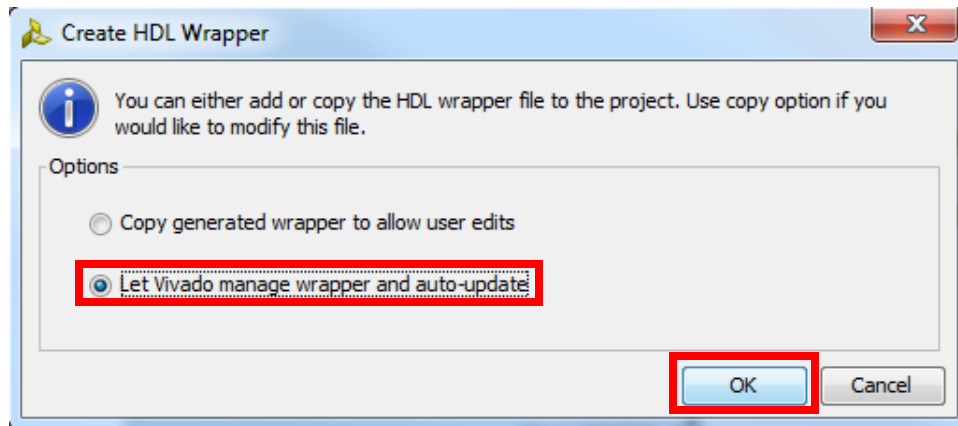



Figure 24 - Create Top Level HDL Wrapper

20. Once the top-level HDL wrapper is created, add a constraints file to the design by clicking the **Add Sources** icon  or through the **File** → **Add Sources** menu item.
21. In the **Add Sources** dialog, select the **Add or Create Constraints** option and click the **Next >** button.

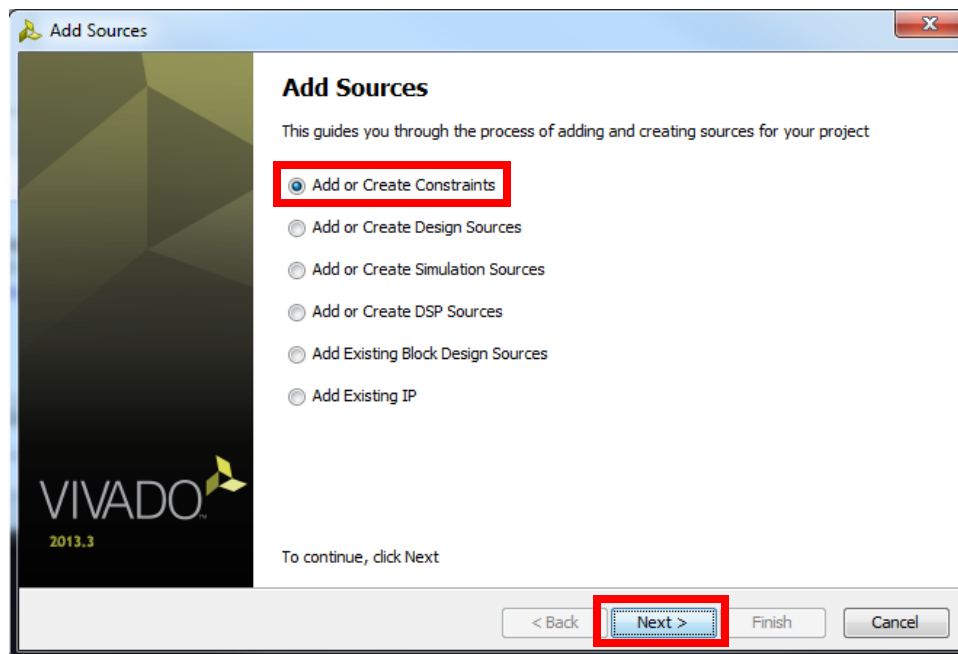


Figure 25 – Adding Sources

22. Click the **Add Files** button and browse to the **microzed_fmc_carrier.xdc** file found in the **C:\Avnet\MicroZed\boards\microzed** folder. The supplied constraints are in XDC format and match the GPIO_0 external port naming performed in an earlier step.

Make sure the **Copy constraints files into project** checkbox is selected then click the **Finish** button.

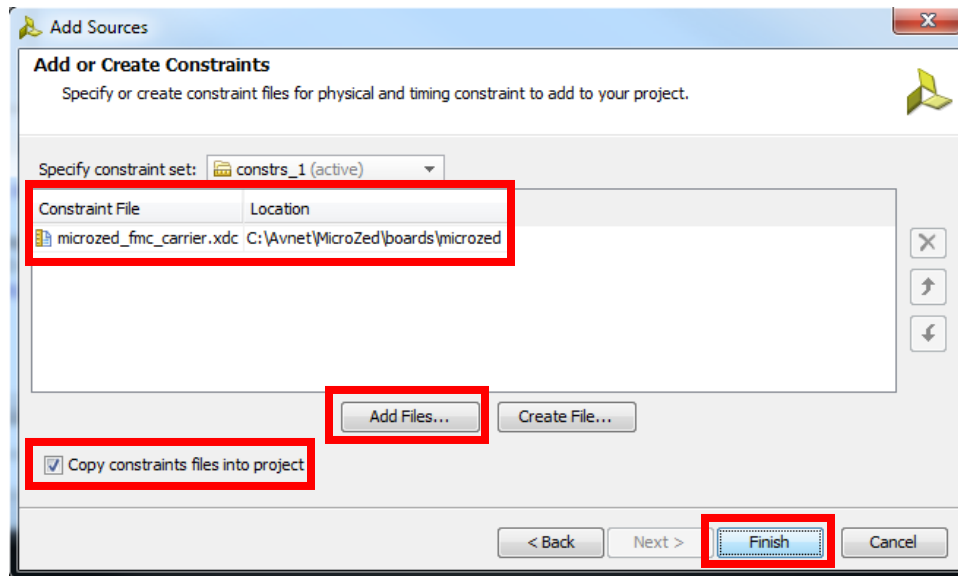


Figure 26 – Adding Constraints Sources

23. Click **Generate Bitstream** in the *Flow Navigator* window then click the **Yes** button to start Synthesis and Implementation flows. *Check the upper right-hand corner of the tool for a status bar.*

24. When Bitstream generation is completed, click the **OK** button to open the implemented design.

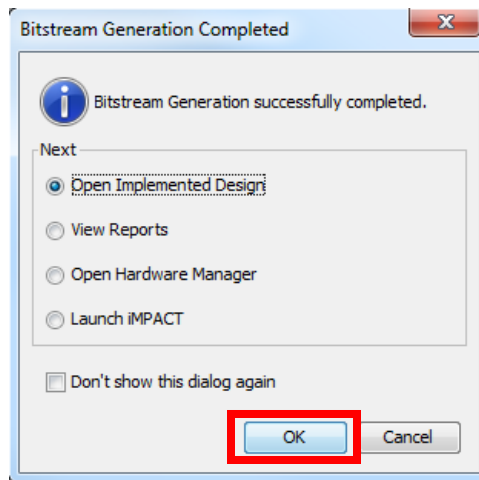


Figure 27 – Open Implemented Design

Now that we've created an embedded system hardware platform, we must make this platform available to the Software Development Kit (SDK). This is done by exporting the hardware platform from Vivado Design Suite to SDK.

25. Select the **File → Export → Export Hardware for SDK...** menu option.
26. Set the **Workspace** field to the **C:\Avnet\MicroZed\SDK_Workspace** folder and select the checkbox for **Launch SDK** to start working on the software application design right now.

Make sure the **Export Hardware** and **Include bitstream** checkboxes are selected then click the **OK** button.

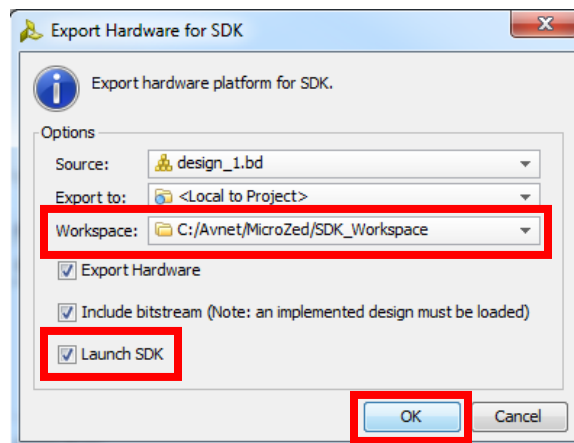


Figure 28 – Export Zynq hardware platform to SDK

27. SDK will launch containing a new Workspace with the new Hardware Platform Specification already setup as a new project. We will use SDK in the next experiments to create the Linux software application and Zynq First Stage Boot Loader or FSBL for short.

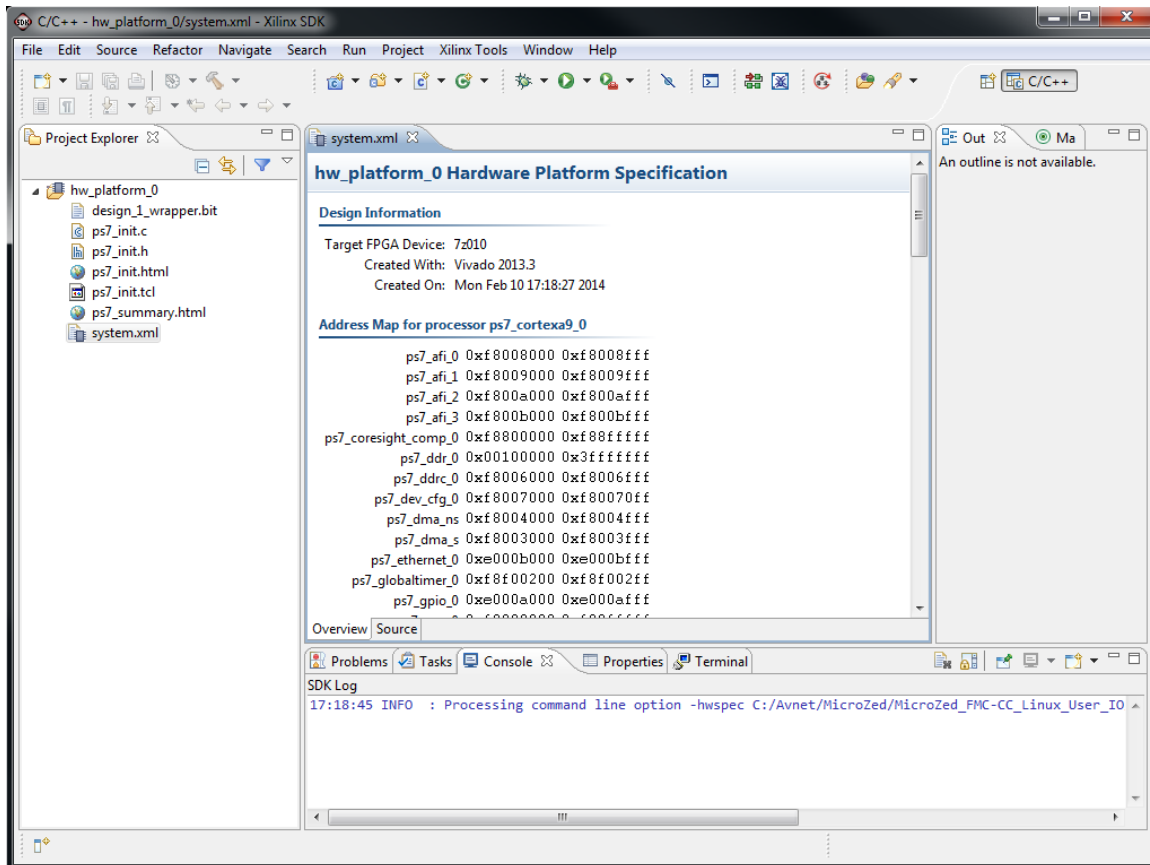


Figure 29 – SDK Window with Hardware Platform Specification

Experiment 3: Add Linux Application

This experiment will demonstrate the steps for creating a new Linux application project within SDK and how to import application source code.

1. If SDK is not already open from the previous experiment, launch SDK by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2013.3 → SDK → Xilinx SDK 2013.3** from the Windows Start menu then select the workspace **C:\Avnet\MicroZed\SDK_Workspace** folder and click the **OK** button.

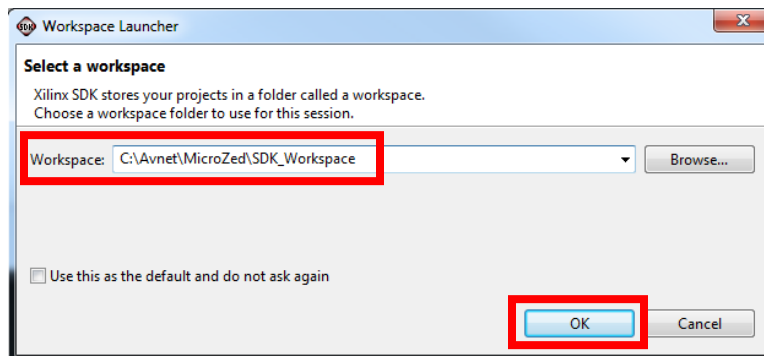


Figure 30 – SDK Workspace

2. In SDK, create a new empty Linux application project by selecting the **File → New → Application Project** menu option.

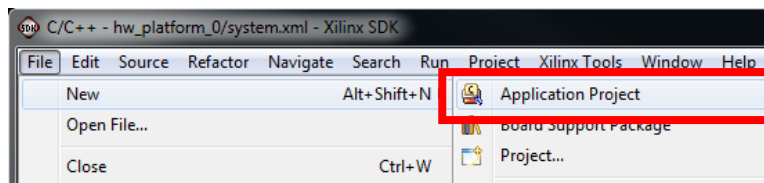


Figure 31 – Creating a New Linux Application Project

3. In the **Project Name** field enter the **linux_user_io** name. Set **OS Platform** to the **linux** option in the drop-down menu. Click the **Next >** button.

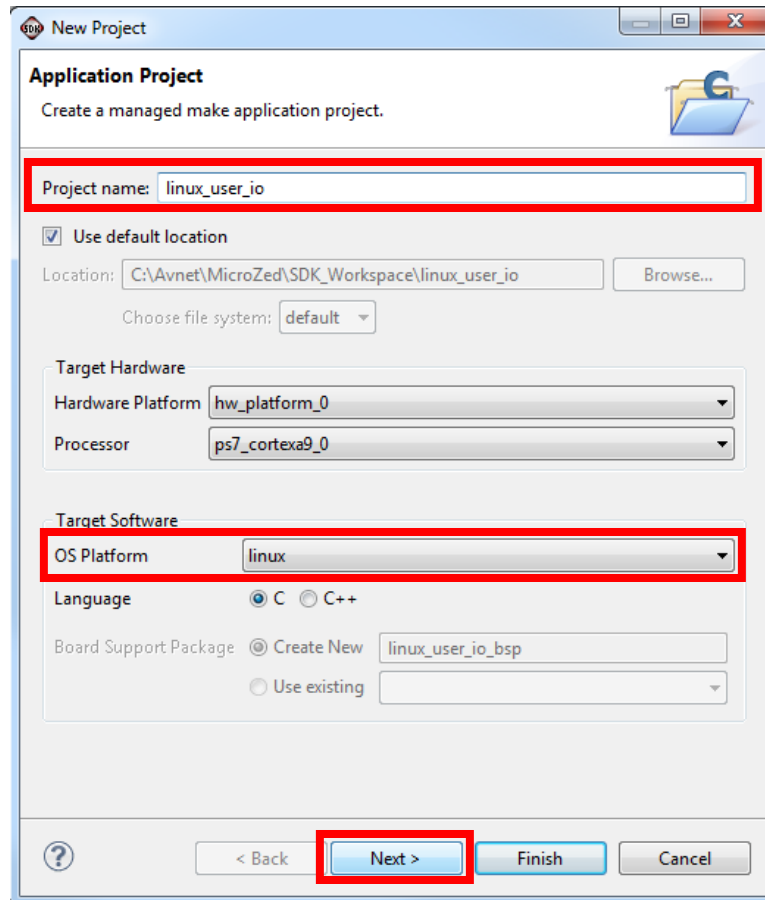


Figure 32 - New Application Wizard

4. Select **Linux Empty Application** from the *Available Templates* panel. Click the **Finish** button.

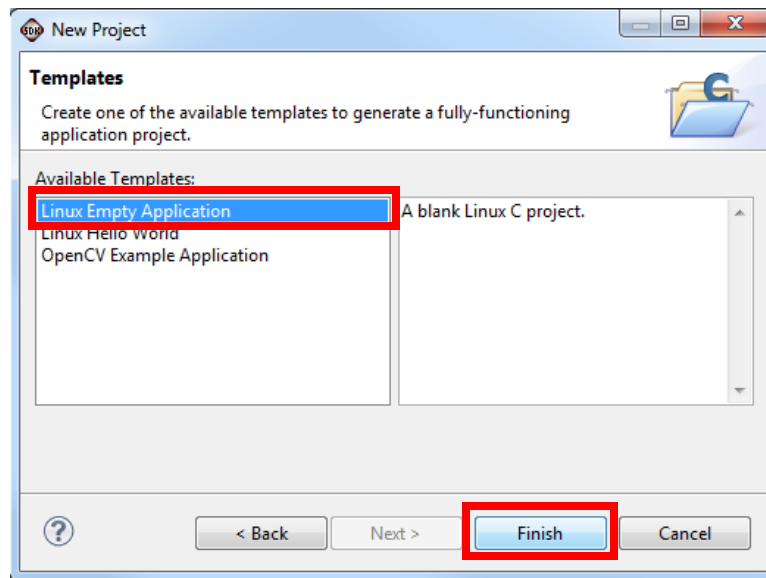


Figure 33 – New Project Template: Linux Empty Application

5. Notice that the **linux_user_io** application is now visible in the *Project Explorer* panel along with the hw_platform_0 Hardware Platform Specification. By default, SDK will attempt build the application automatically after it is added but there is nothing to build since application source code has not been added yet.

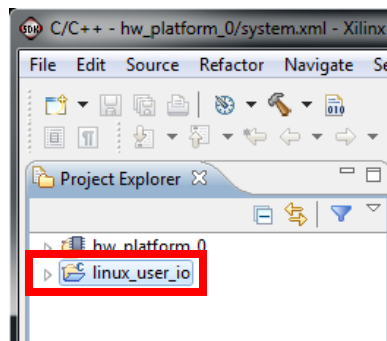


Figure 34 – Project Explorer View with linux_user_io C Application Added

6. In the **Project Explorer** tab, expand **linux_user_io** and right-click on the **src** folder. Click on the **Import** option in the pop up menu.

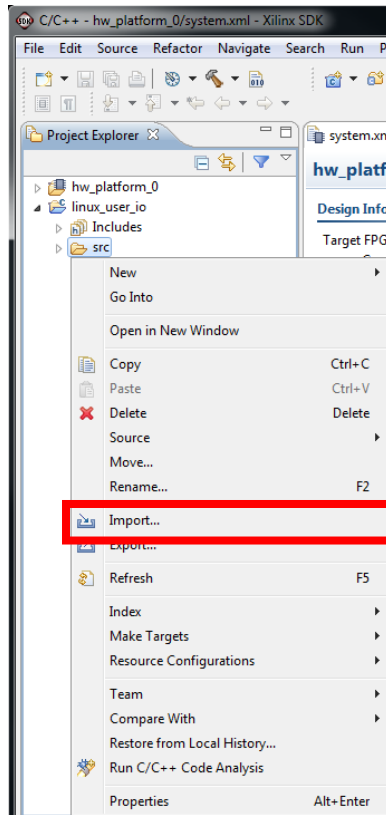


Figure 35 – Import Application Code

7. In the **Import** window, expand the **General** item, select the **File System** option, and click the **Next** button.

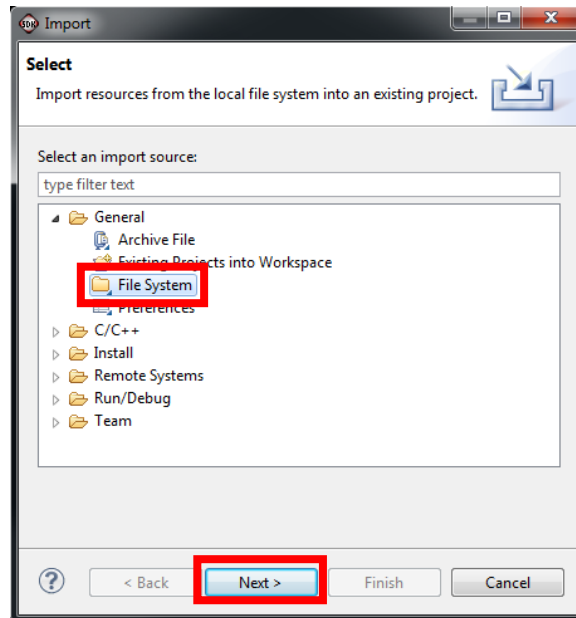


Figure 36 – Importing from a File System

- Click on the **Browse** button and select the following folder which contains the application code that we wish to run under Linux:

C:\Avnet\MicroZed\sw\linux_user_io

After this folder is selected within the **Browse** dialog, click the **OK** button to search the folder for files to import into the application project. Select the **linux_user_io.c** file and then click the **Finish** button to complete the **Import** operation.

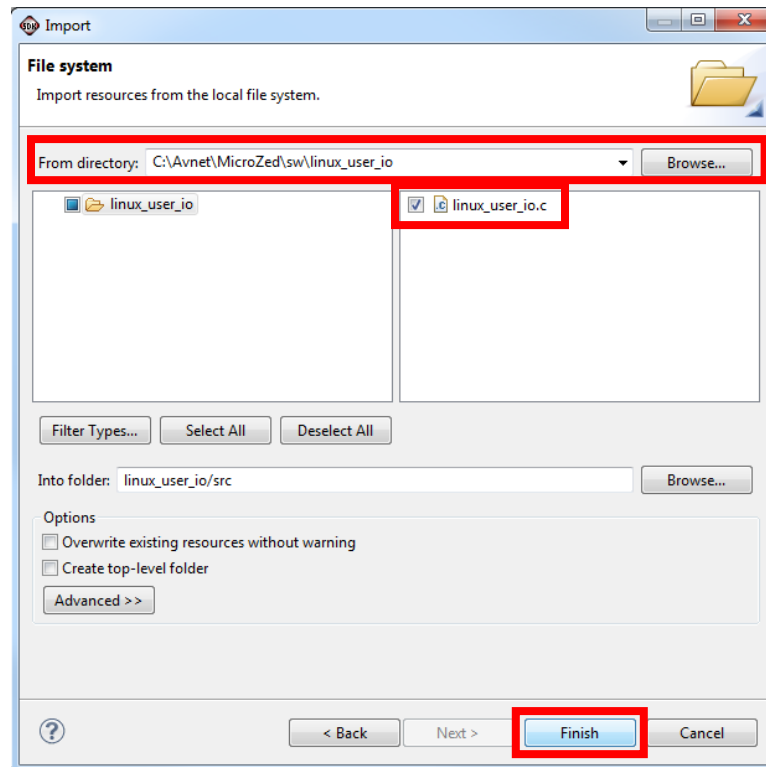
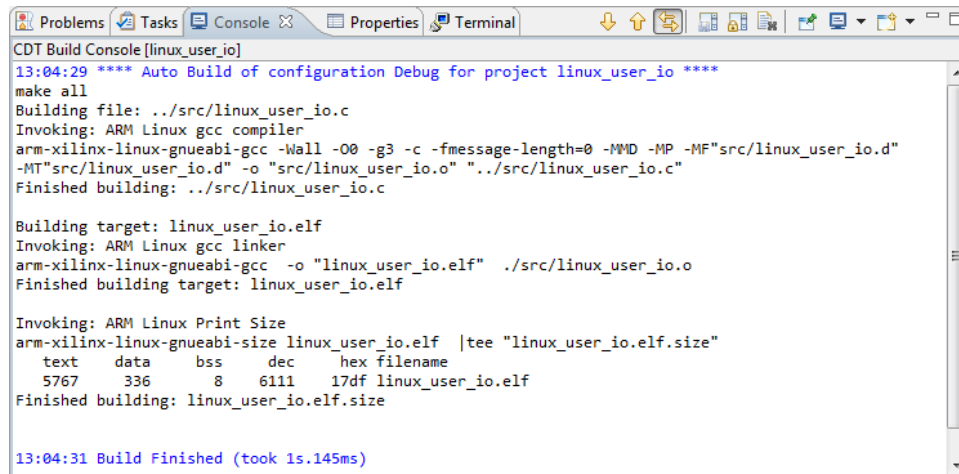


Figure 37 – Selecting Application Source Files

9. The SDK Console panel shows the results of the build. Make sure that the application is built without errors.



```
CDT Build Console [linux_user_io]
13:04:29 **** Auto Build of configuration Debug for project linux_user_io ****
make all
Building file: ../src/linux_user_io.c
Invoking: ARM Linux gcc compiler
arm-xilinx-linux-gnueabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MMD -MP -MF"src/linux_user_io.d"
-MT"src/linux_user_io.d" -o "src/linux_user_io.o" "../src/linux_user_io.c"
Finished building: ../src/linux_user_io.c

Building target: linux_user_io.elf
Invoking: ARM Linux gcc linker
arm-xilinx-linux-gnueabi-gcc -o "linux_user_io.elf" ../src/linux_user_io.o
Finished building target: linux_user_io.elf

Invoking: ARM Linux Print Size
arm-xilinx-linux-gnueabi-size linux_user_io.elf |tee "linux_user_io.elf.size"
  text    data    bss     dec    hex filename
  5767    336      8    6111    17df linux_user_io.elf
Finished building: linux_user_io.elf.size

13:04:31 Build Finished (took 1s.145ms)
```

Figure 38 – Application Build Console Window

10. After SDK finishes compiling the new application code, the ELF is available in the following location:

C:\Avnet\MicroZed\SDK_Workspace\linux_user_io\Debug\linux_user_io.elf

Copy this executable file to the SD card staging folder located here:

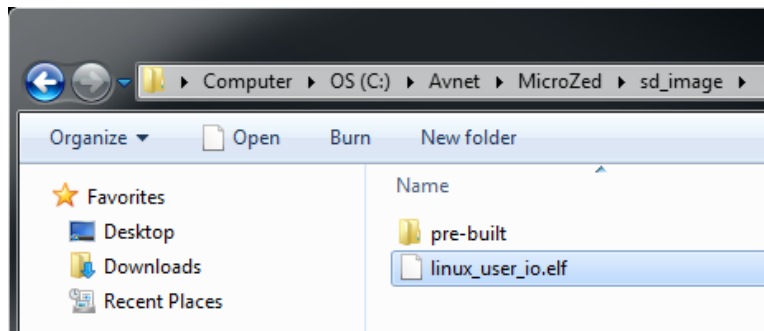


Figure 39 – Executable Copied to SD Card Staging Folder

Experiment 4: Create the Zynq FSBL

Xilinx SDK includes a template to automatically create the Zynq First Stage Boot Loader (FSBL) for us. By interpreting the hardware platform definition, SDK will generate a fully functional FSBL for us. This is one of the many useful things that SDK automates to make your job as a software developer a bit easier.

Similar to the way the application project in the previous exercise was created, SDK this time will be used to generate the First Stage Boot Loader application using a Standalone application template.

1. In SDK, begin creating the Zynq FSBL project by selecting the **File** → **New** → **Application Project** menu option.

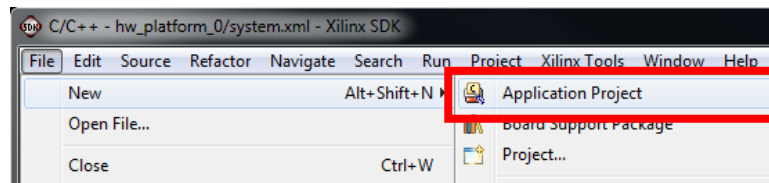


Figure 40 – Creating a New Standalone Application Project

2. Name the new application project **zynq_fsbl_0** then click the **Next >** button.

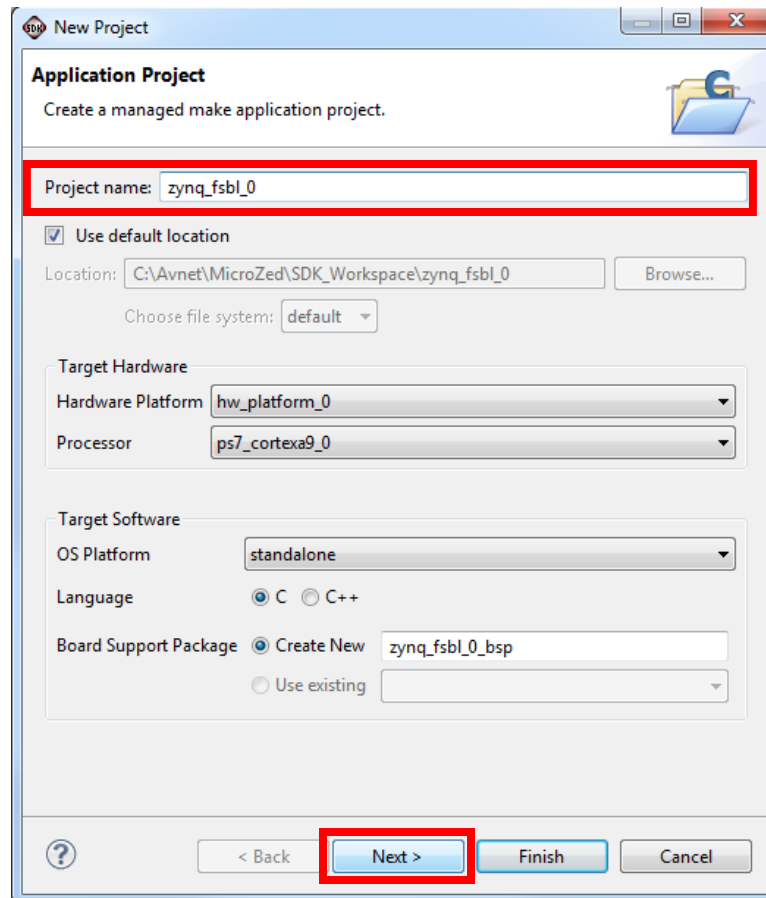


Figure 41 – New Application Wizard

3. Select **Zynq FSBL** from the *Available Templates* panel. Click the **Finish** button.

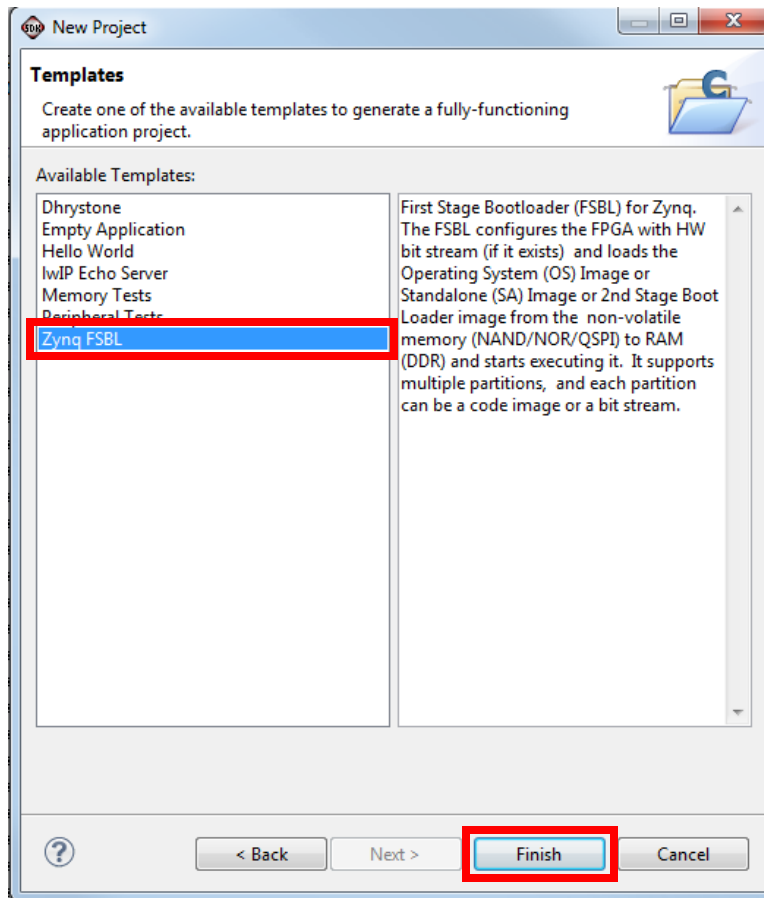


Figure 42 – Use the Zynq FSBL Template

4. Notice that the **zynq_fsbl_0** application is now visible in the *Project Explorer* panel along with the BSP automatically created for the FSBL. By default, SDK will build the FSBL application code automatically after it is added and it should build with no errors or warnings reported in the console.

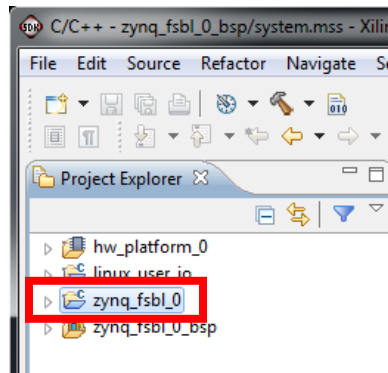


Figure 43 – Project Explorer View with zynq_fsbl_0 Application Added

Experiment 5: Prepare the Boot Image

MicroZed has two non-volatile bootable sources, QSPI and microSD Card. This experiment will demonstrate steps in creating the SD card boot image for MicroZed.

1. In SDK, select the **linux_user_io** application project from the Project Explorer panel. This will result in a blank boot image template being setup within the **Create Zynq Boot Image** window in the next step.

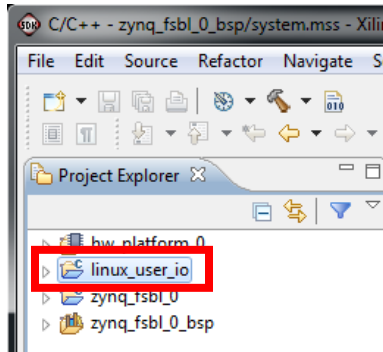


Figure 44 – Select linux_user_io Application

2. Select **Xilinx Tools** → **Create Zynq Boot Image** from the SDK menu to open the **Create Zynq Boot Image** window.

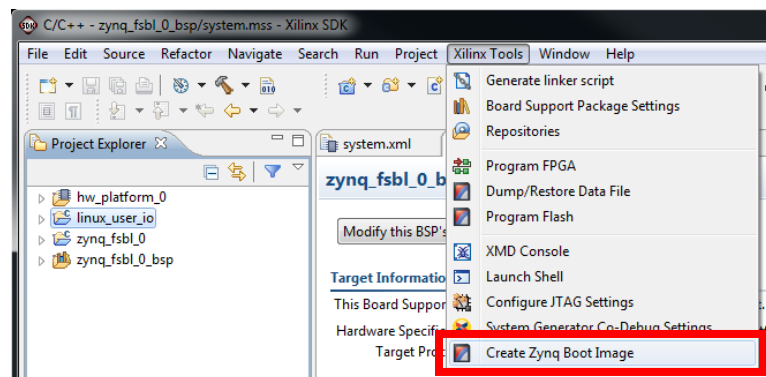


Figure 45 – Opening the Create Zynq Boot Image Window

3. Set the BIF (Boot Image Format) file path to the following:

C:\Avnet\MicroZed\sd_image\bootimage.bif

Click the Add button to add each of the following files (in the order shown here) to the Zynq boot image partition list:

C:\Avnet\MicroZed\SDK_Workspace\zynq_fsbl_0\Debug\zynq_fsbl_0.elf

C:\Avnet\MicroZed\SDK_Workspace\hw_platform_0\design_1_wrapper.bit

C:\Avnet\MicroZed\sd_image\pre-built\u-boot.elf

Set the output file path to the following:

C:\Avnet\MicroZed\sd_image\bootimage.bin

Review the settings for completeness and then click on the **Create Image** button.

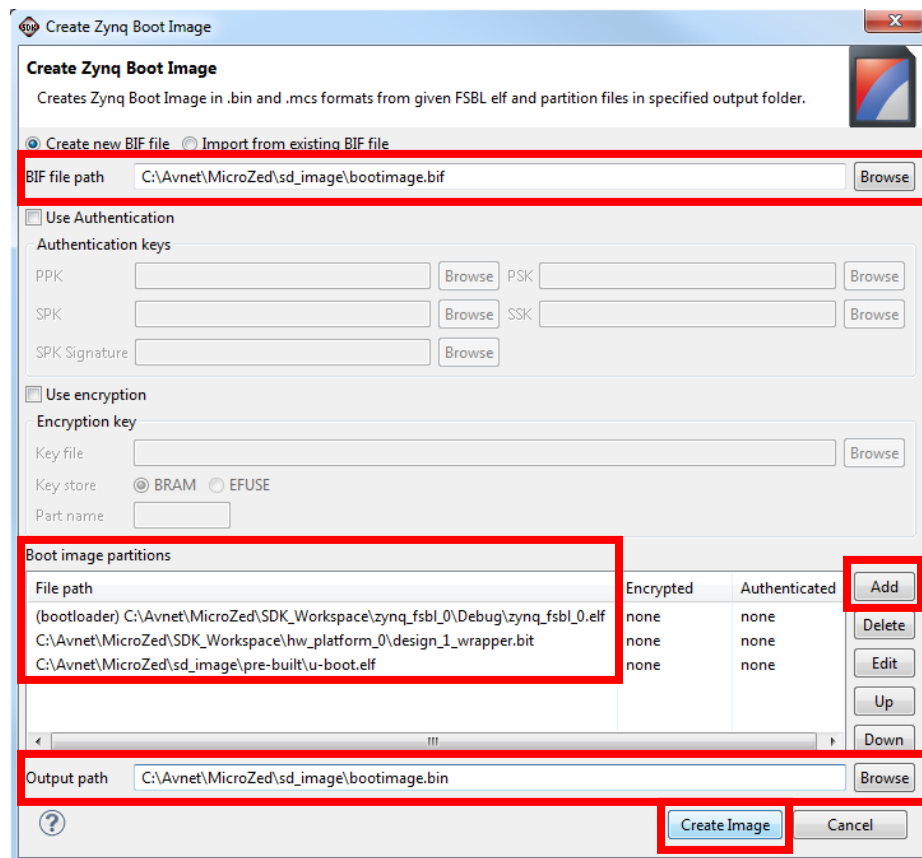


Figure 46 – Create Zynq Boot Image Dialog

Note: The U-boot binary u-boot.elf is the same binary supplied by Xilinx for ZedBoard for the 2013.3 Linux Release and can be found on the Xilinx Wiki:

<http://www.wiki.xilinx.com/Zynq+14.7-2013.3+Release>

4. Using Windows Explorer, navigate to the microSD staging folder:

C:\Avnet\MicroZed\sd_image

Locate the **bootimage.bin** file created in the previous steps and rename the file to **boot.bin** which is the filename that the Zynq BootROM will search for on the microSD card during SD boot mode.

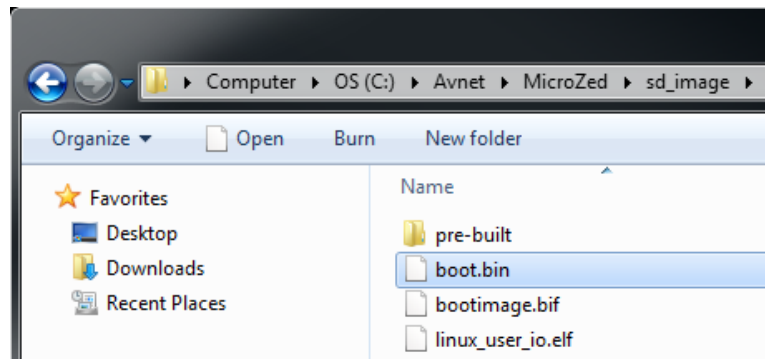


Figure 47 – Renaming bootimage.bin to boot.bin

Experiment 6: Setup MicroZed MicroSD Card

This section of the tutorial demonstrates how to setup a MicroZed microSD card to boot into an open source Linux platform.

Note: If the Zynq boot image `boot.bin` was not built successfully in the previous section, a prebuilt boot image can be copied from the prebuilt image files folder `C:\Anvet\MicroZed\sd_image\prebuilt\` over to the microSD staging folder `C:\Avnet\MicroZed\sd_image\` before completing the remainder of this section of the tutorial.

1. Insert the microSD card into the PC or SD card reader and wait for it to enumerate as a Windows drive. If prompted by Windows when inserting the SD card, select the **Continue without scanning** option.

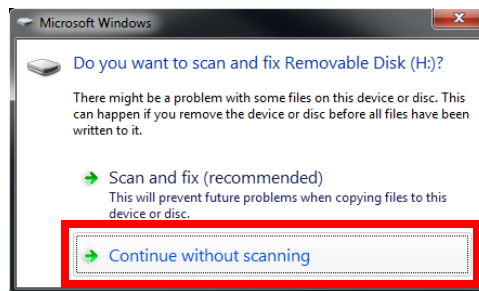


Figure 48 – Windows Prompt for Scanning and Fixing an SD Card

2. The Zynq BootROM is capable of interpreting the FAT32 file system for SD card boot mode.

If the microSD card used is not already formatted for FAT32 file system, right click on the drive in Windows Explorer and select the **Format** option. Select the options shown in Figure 49 and click the **Start** button.

When the format process is complete, click the **Close** button.

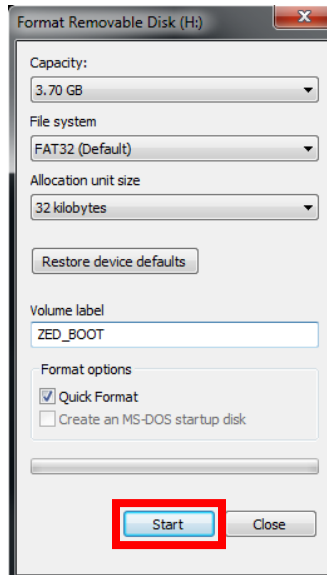


Figure 49 – Formatting the MicroSD Card with FAT32 File System

3. Copy the **boot.bin** and **linux_user_io.elf** files from the **sd_image** staging folder to the top level of the microSD card. Replace any existing versions of these files that may already be on the microSD card.

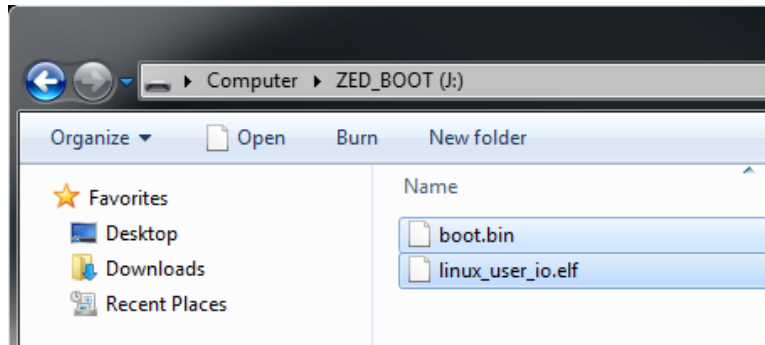


Figure 50 – Boot Image and Linux Application Files Copied to the MicroSD Card

4. Copy the MicroZed Linux system files **devicetree.dtb**, **init.sh**, **uImage**, and **uramdisk.image.gz** files from the **sd_image\pre-built** folder to the top level of the microSD card. Replace any existing versions of these files that may already be on the microSD card.

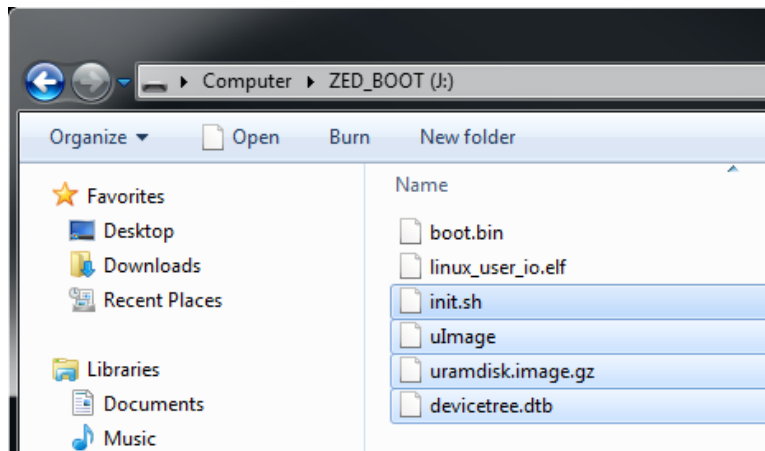


Figure 51 – MicroZed Linux Platform Files Copied to the MicroSD Card

Once these files are copied to the microSD card, eject the microSD card from the PC or SD card reader.

Experiment 7: Run the Design

This section of the tutorial explains how to set up the hardware for operation and how to interact with the design once it is running.

The example design interfaces the Zynq processing system (PS) through routing resources inside the programmable logic (PL). A Linux application controls the blink pattern and rate of the four LEDs on the FMC-CC. This is done with the built-in PS GPIO controller. The LEDs are controlled by input through the two push buttons. The LEDs and push buttons of interest are highlighted in the photo below.

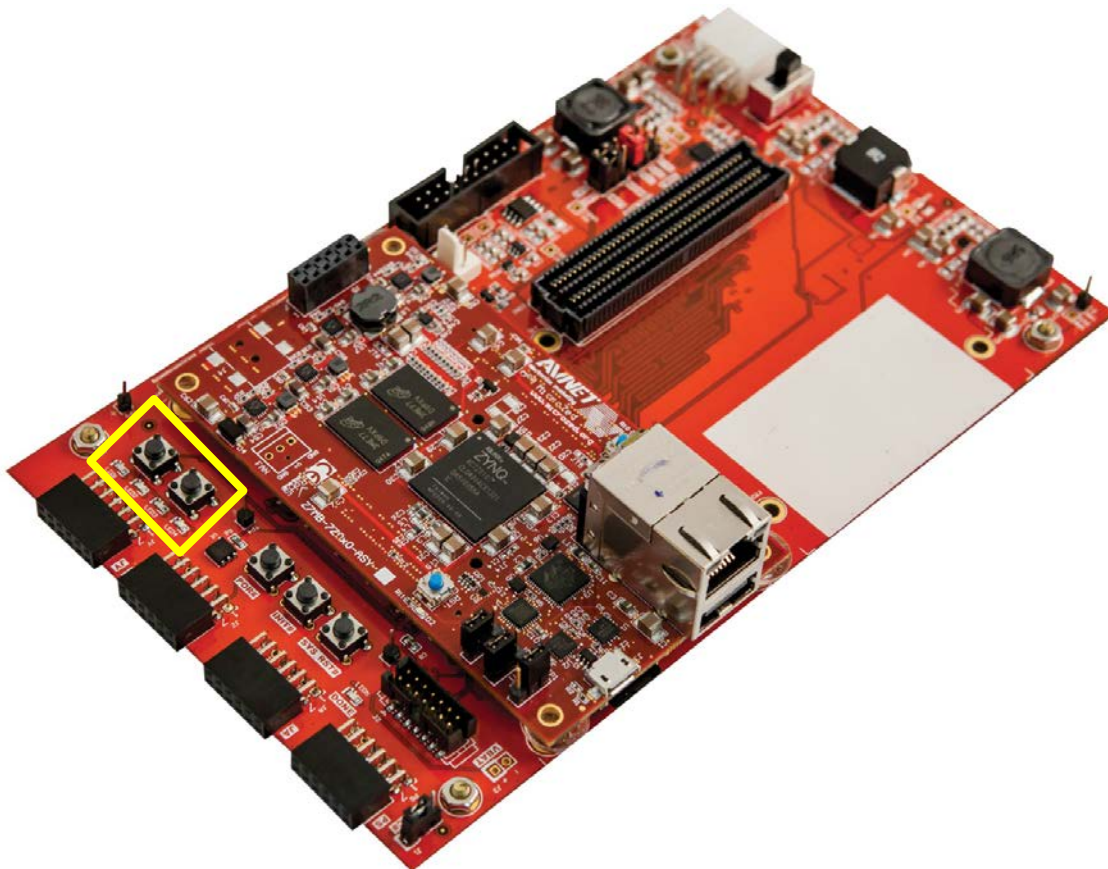


Figure 52 – User LEDs and Push Buttons for Example Design

A block diagram for the design is shown below.

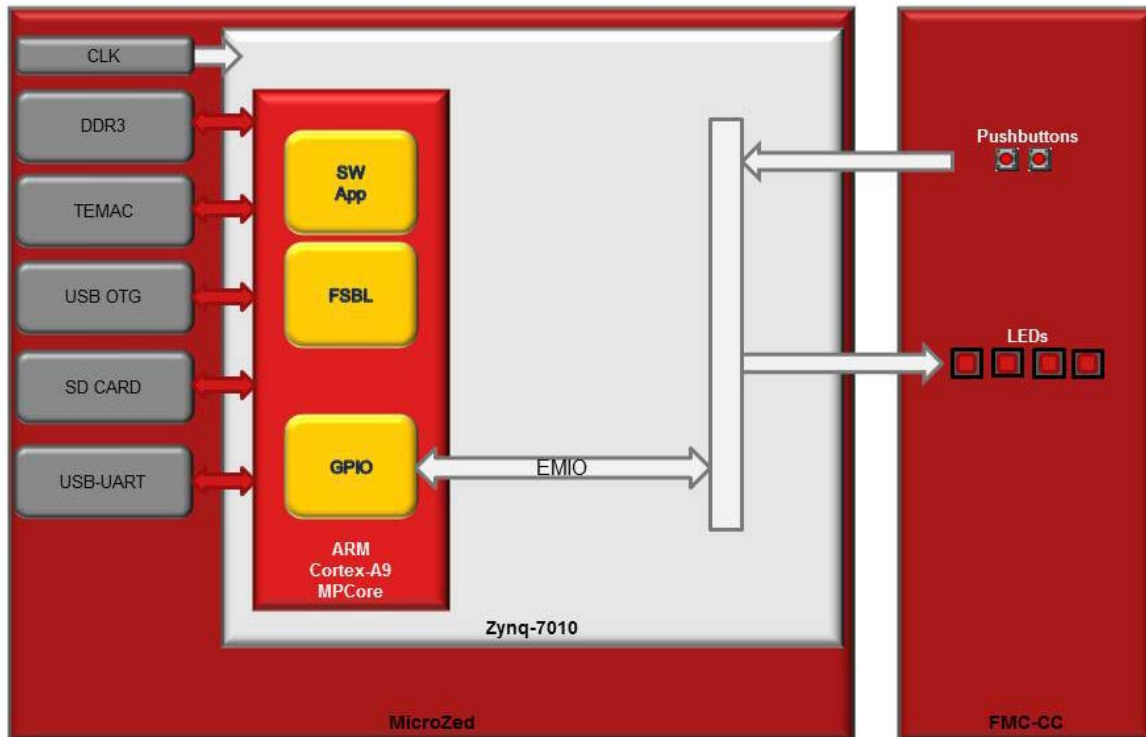


Figure 53 – MicroZed/FMC-CC Hardware Design

1. Insert the microSD card into the microSD card slot (J6) located on the underside of MicroZed module.

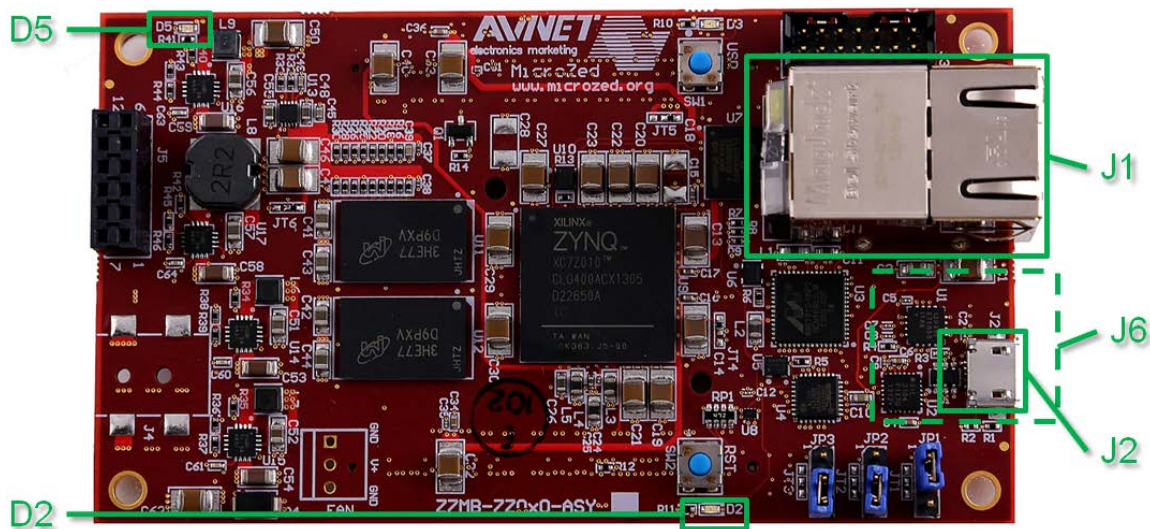


Figure 54 – MicroZed Hardware Reference

2. Set the MicroZed boot mode (JP3-JP1) jumpers to SD card mode as described in the Hardware Users Guide.

SD Card



Figure 55 – SD Card Boot Jumper Settings

3. Make sure the FMC-CC power switch is in the OFF position.
4. Insert the MicroZed module onto the FMC-CC.
5. Set the VADJ (J6) jumpers to **3V3** (1-2).
6. Insert the appropriate country plug into the 12V AC/DC adapter. Plug it into the CON4 2x3 power connector. (NOTE – this 2x3 connector is NOT compatible with ATX power supplies.)
7. We will be using a micro-USB cable to communicate through a terminal, but this will be plugged in a bit later.

8. Turn the power switch on the FMC-CC to the ON position. After 1-2 seconds, you will notice four LEDs that are lit:
 - D5 on MicroZed, indicating Power Good
 - LED6 on FMC-CC, indicating Power Good
 - D2 on MicroZed, Zynq PL configuration DONE
 - LED5 on FMC-CC, Zynq PL configuration DONE

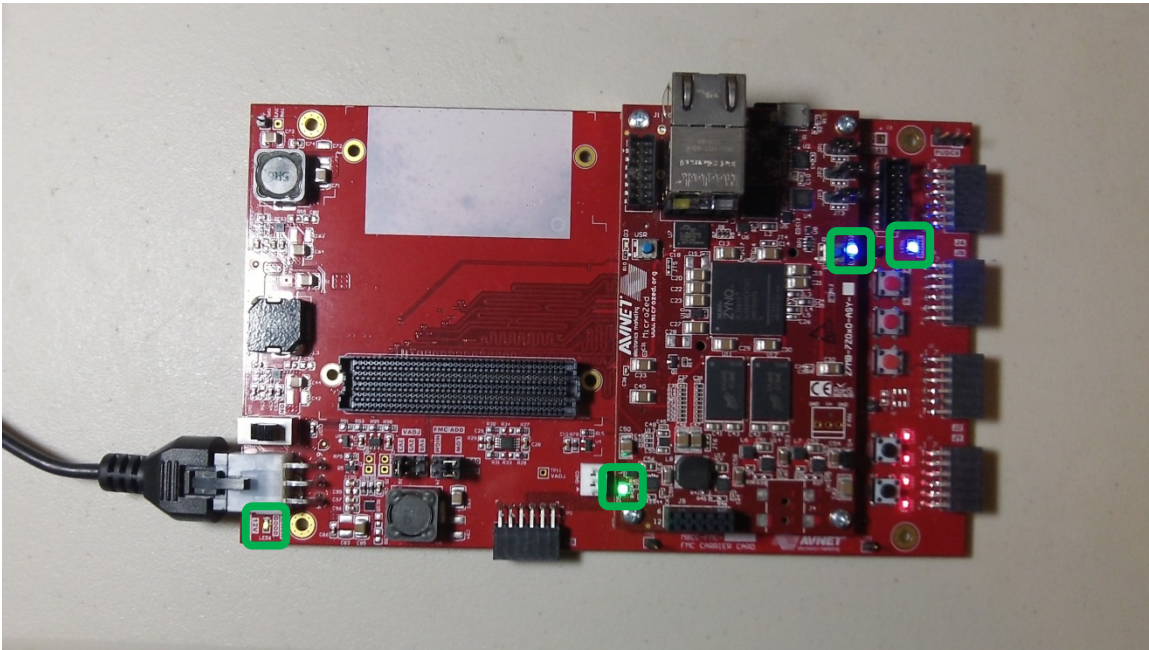


Figure 56 – MicroZed / FMC-CC Powered On with LEDs

9. Now plug in the microUSB cable between the PC and the J2 USB-UART connector on MicroZed.

10. On the PC, open a serial terminal program. Tera Term is used to show the example output for this lab document. Follow the instructions in the CP210x Setup Guide to set the terminal as shown in Figure 57, using the appropriate COM port that you discover on your own machine.

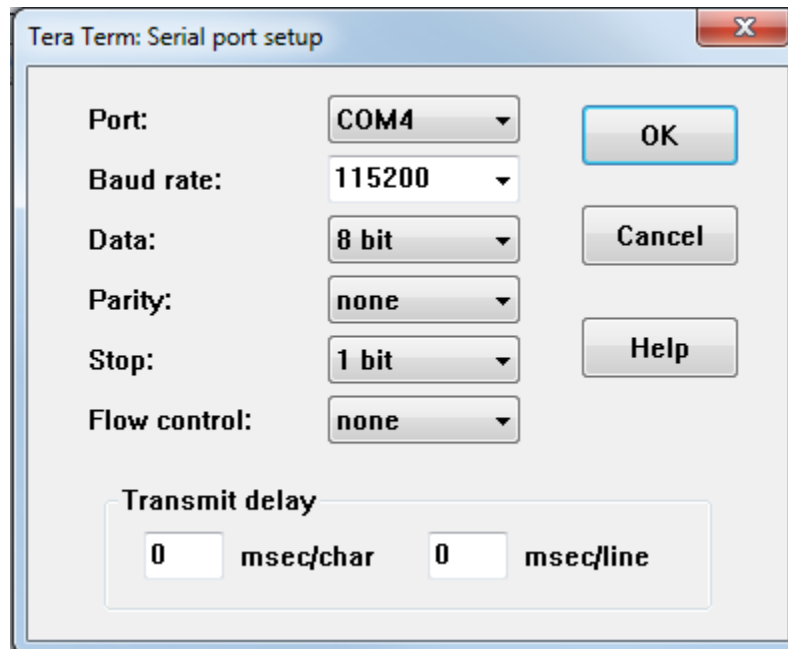
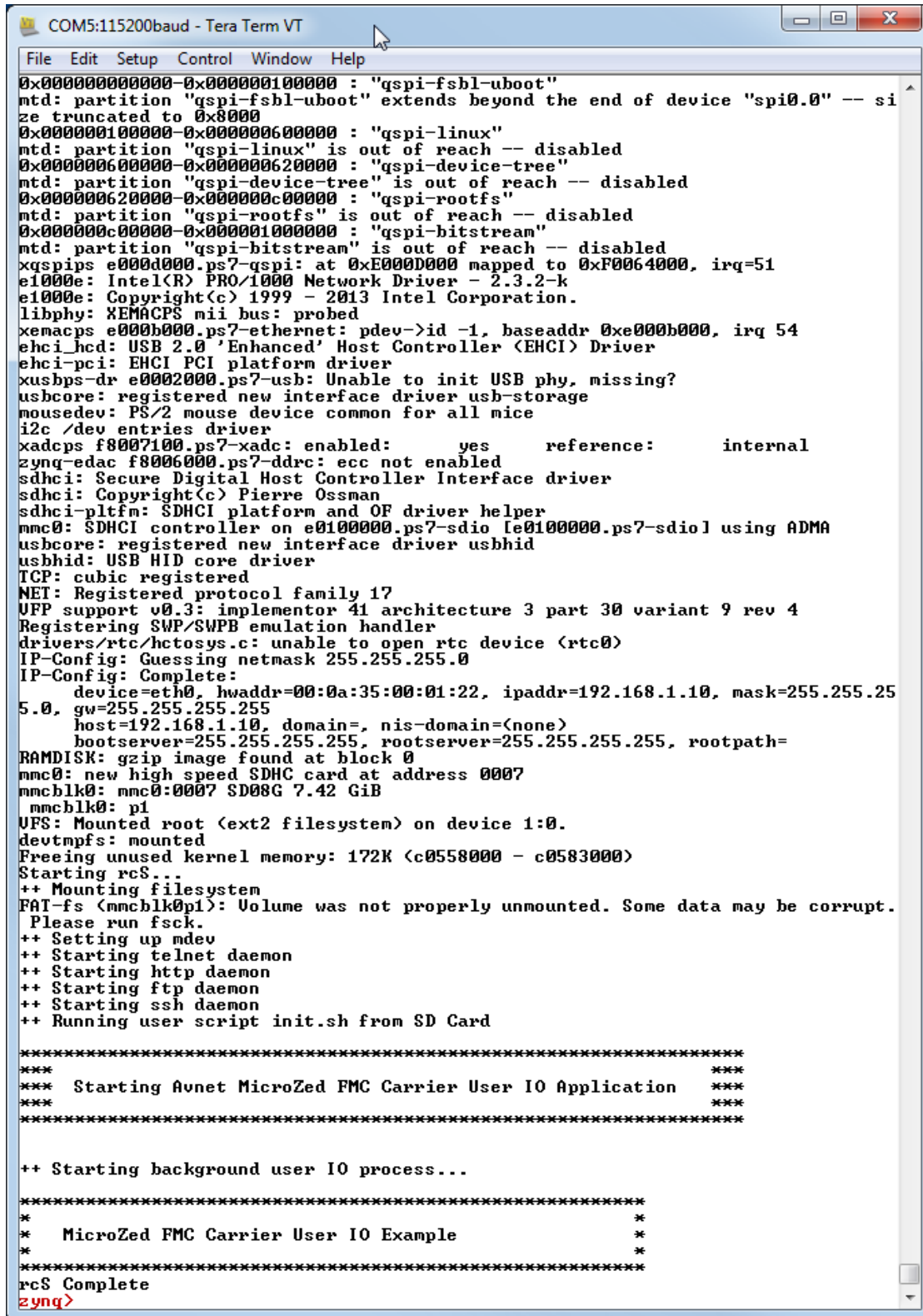


Figure 57 – Connect Tera Term to the proper COMx port

11. Perform a System Processor Reset by pushing the SYS_RST# button (SW1) on the FMC-CC. The terminal output shows that the Zynq boots to U-boot and then Linux, launching the MicroZed FMC Carrier User IO Example and concluding with the **zynq>** prompt.



```

COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help
0x00000000-0x00000001 : "qspi-fsbl-uboot"
mtd: partition "qspi-fsbl-uboot" extends beyond the end of device "spi0.0" -- si
ze truncated to 0x8000
0x00000001-0x00000006 : "qspi-linux"
mtd: partition "qspi-linux" is out of reach -- disabled
0x00000006-0x00000062 : "qspi-device-tree"
mtd: partition "qspi-device-tree" is out of reach -- disabled
0x00000062-0x000000c0 : "qspi-rootfs"
mtd: partition "qspi-rootfs" is out of reach -- disabled
0x000000c0-0x00000100 : "qspi-bitstream"
mtd: partition "qspi-bitstream" is out of reach -- disabled
xqspips e000d000.ps7-qspi: at 0xE000D000 mapped to 0xF0064000, irq=51
e1000e: Intel(R) PRO/1000 Network Driver - 2.3.2-k
e1000e: Copyright(c) 1999 - 2013 Intel Corporation.
libphy: XEMACPS mii bus: probed
xemacps e000b000.ps7-ethernet: pdev->id -1, baseaddr 0xe000b000, irq 54
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ehci-pci: EHCI PCI platform driver
xusbps-dr e0002000.ps7-usb: Unable to init USB phy, missing?
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
xadcps f8007100.ps7-xadc: enabled: yes reference: internal
zynq-edac f8006000.ps7-ddrc: ecc not enabled
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
sdhci-pltfm: SDHCI platform and OF driver helper
mmc0: SDHCI controller on e0100000.ps7-sdio [e0100000.ps7-sdio] using ADMA
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
UFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
Registering SWP/SWPB emulation handler
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
    device=eth0, hwaddr=00:0a:35:00:01:22, ipaddr=192.168.1.10, mask=255.255.25
5.0, gw=255.255.255.255
    host=192.168.1.10, domain=, nis-domain=(none)
    bootserver=255.255.255.255, rootserver=255.255.255.255, rootpath=
RAMDISK: gzip image found at block 0
mmc0: new high speed SDHC card at address 0007
mmcblk0: mmc0:0007 SD08G 7.42 GiB
    mmcblk0: p1
UFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing unused kernel memory: 172K (c0558000 - c0583000)
Starting rcS...
++ Mounting filesystem
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt.
Please run fsck.
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting ssh daemon
++ Running user script init.sh from SD Card

*****
*** Starting Avnet MicroZed FMC Carrier User IO Application ***
*****

++ Starting background user IO process...

*****
* MicroZed FMC Carrier User IO Example *
*                                     *
*****
rcS Complete
zynq>

```

Figure 58 – MicroZed and FMC-CC Example Design

You'll notice now that the four User LEDs are flashing back and forth rapidly. Using the Push Buttons, you can cause the processor to change the blink pattern and rate.

12. Press BTN1. The LED flash pattern changes to ALL ON then ALL OFF at a rate of 200 milliseconds. Release BTN1.
13. Press BTN2. The pattern is identical to BTN1 push. Release BTN2.
14. Push both BTN1 and BTN2. The blink pattern changes to ALL ON / ALL OFF, but the rate is 4 times slower at 800 milliseconds. Release the buttons.

A number of activities can also be performed through the terminal using Linux. If you are unfamiliar with Linux, please refer to the *MicroZed Getting Started Guide* for some examples.

This concludes the tutorial.

Note: The U-boot binary u-boot.elf is the same binary supplied by Xilinx for ZedBoard for the 2013.3 Linux Release and can be found on the Xilinx Wiki:

<http://www.wiki.xilinx.com/Zynq+14.7-2013.3+Release>

Note: The Linux kernel binary ulmage is the same binary supplied by Xilinx for ZedBoard for the 2013.3 Linux Release and can be found on the Xilinx Wiki:

<http://www.wiki.xilinx.com/Zynq+14.7-2013.3+Release>

Note: The ramdisk image uramdisk.image.gz is a U-boot wrapped version of the Xilinx provided arm_ramdisk.image.gz found on the Xilinx Wiki:

<http://www.wiki.xilinx.com/Build+and+Modify+a+Rootfs>

Note: The kernel device tree binary used here is the flattened binary representation of the devicetree_microzed_2013_3.dts device tree source file found in the C:\Avnet\MicroZed\sd_image\pre-built\ folder.

Revision History

Date	Version	Revision
27 Mar 2014	2013_3.01	Initial Avnet release for Vivado 2013.3