UIO Lab

Pre-Requisite

It will be assumed that you already know Vivado and IPI. For that reason, you will not be asked to create the IPI design on your own. Instead, a fully built Vivado project will be provided. You can open and look at the design to reference it, but it will not be required to build.

Overview

The primary goal of this lab is to enable you to do the following with the UIO framework:

1. Peek/Read at registers in the PL
2. Poke/Write at registers in the PL
3. Respond to interrupts from the PL

You will also have the opportunity to use devmem to peek and poke the registers in the PL. After the u-boot training, it is suggested to read and write PL registers at u-boot as well.

Lab Instructions

1. Hardware Setup

The hardware has two IPs in the PL for the PS to communicate to. One is an HLS block that counts for a programmable set of beats and then sends an interrupt afterward. This IP will be used to show how interrupts can be used within UIO.

The second IP in the IPI project is a BRAM. Below is a screenshot of the design:

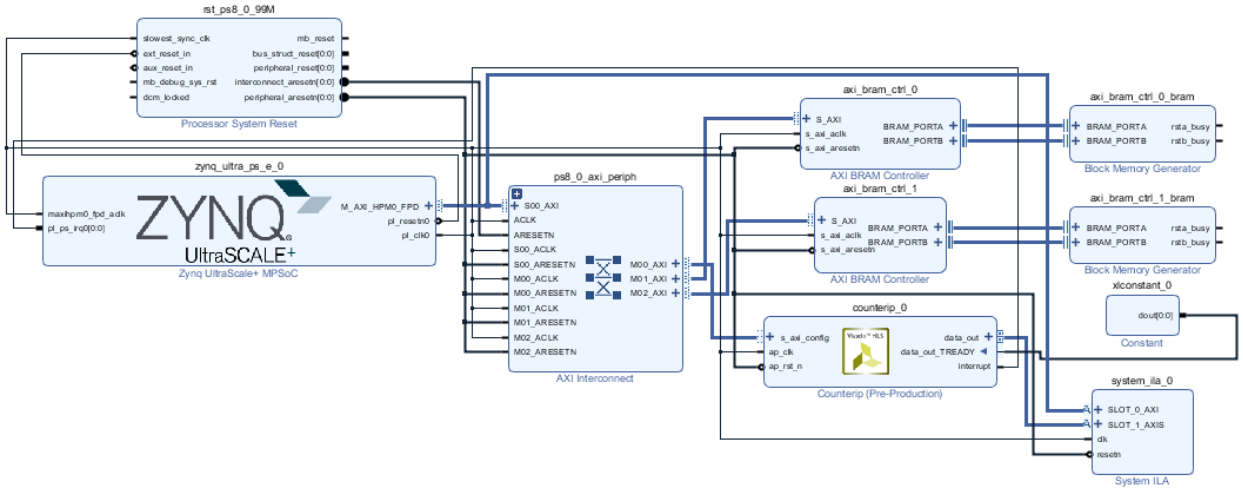


Figure 1

Exceptions

If for any reason the project does not work for you, please feel free to create this subsystem on your own. For example, you may not have a ZCU102 board. This should be a simple enough design to re-create. User the bd tcl or copy the same system manually in IPI.

2. Exporting from Vivado to PetaLinux

The Vivado Project needs to export the Hardware information to PetaLinux. There is an archived Vivado project in the "vivado" directory of the lab file. Unzip the zip file in this directory and open the Vivado project.

>>***unzip ./uiolab_2018.2.xpr.zip***

Extract the HDF with the Export Hardware function in Vivado as shown below in Figure 2. It's possible to export the hardware either before or after the bitstream is created. If the HDF is created after the bitstream is created, you can include the bitstream into the HDF so PetaLinux also manages the bitstream. However, I don't see any benefit in doing this so I recommend to keep the bitstream outside of the HDF.

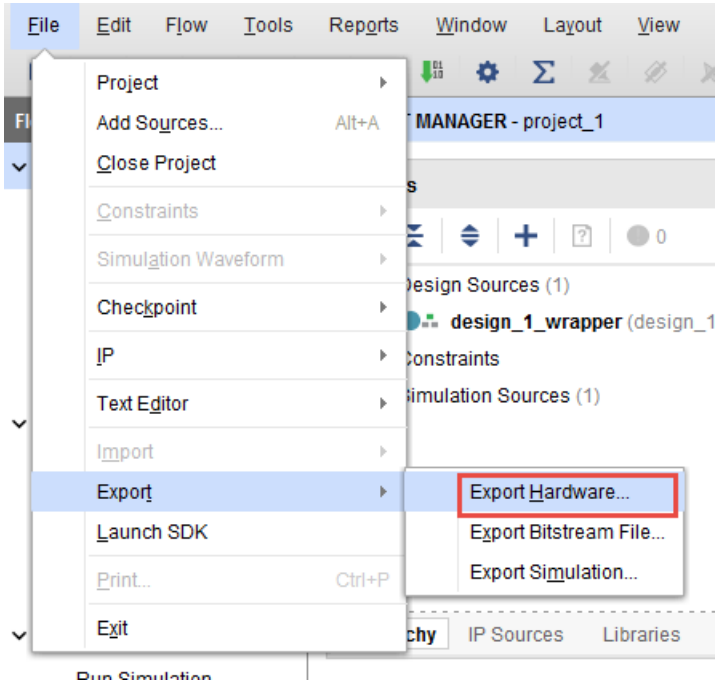


Figure 2

3. Importing the HDF

Now PetaLinux needs to know the information in the HDF. Some information in the HDF that is relevant to PetaLinux is:

- Peripherals used
  - Both PL and PS
- Peripheral addresses

- Device selected – hence processor

Pro Tip - The HDF includes the Tcl to recreate the IPI project. Just unzip it and you'll see it there. You'll find other handy files there too.

The first step to importing the HDF is to create your PetaLinux project. Within the lab files, there should be a "peta" directory and you'll want to cd into it. Then create a petalinux project by typing the following:

>> ***petalinux-create -n myproj_name -t project --template zynqMP***

After this command, a directory named "myproj_name" will be created and you need cd into it.

>> ***cd myproj_name***

Explore this directory and familiarize yourself with the PetaLinux project.

After you're familiar with the project directory, ensure that you're in the myproj_name directory and run the following command:

>> ***petalinux-config --get-hw-description=../../vivado/project_1/project_1.sdk/*** (make sure it's the directory to where you placed your HDF)

When you enter this command, you'll see an old-school looking menuconfig that allows you to choose different configurations for various aspects of the PetaLinux project. Below is what it should look like:

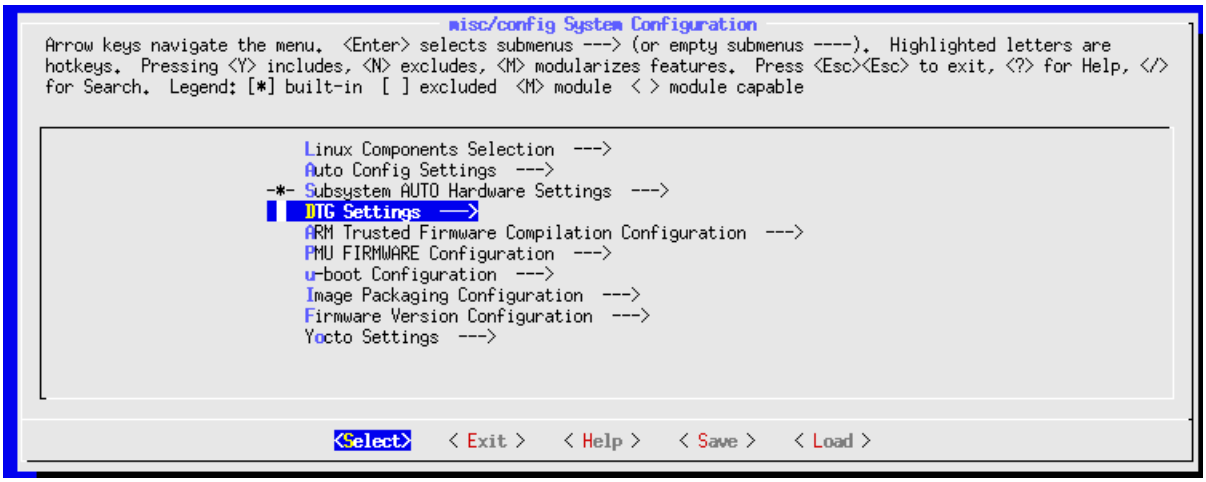


Figure 3

Go through the various options to familiarize yourself. For example, find out how you can change your temp directory (but don't change it). Below are some tips:

Tip 1 – To search use the backslash key (/). Then type the name of what you're looking for.

Tip 2 – Double tap the Esc key to go backward

Tip 3 – Use the space bar to enable/disable items

You have now completed the act of importing the HDF, however, don't close the menuconfig.

4. Configuring the PetaLinux Project

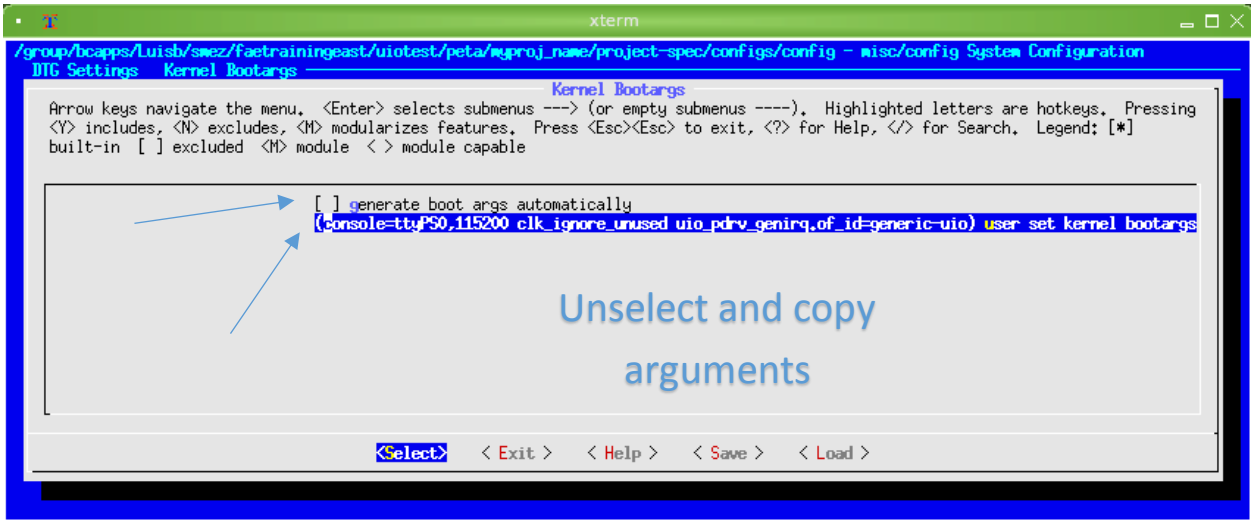
In case you did close the menuconfig, you can bring it back up by simply typing “petalinux-config” without any options.

Change the bootargs to the following so that the uio drivers will bind to the device tree entry. Without doing this, the UIO drivers will not work. The bootargs are located here:

DTG Settings→Kernel Bootargs→

Uncheck the option to generate the boot args automatically and set the options as follows:

console=ttyPS0,115200 clk_ignore_unused uio_pdrv_genirq.of_id=generic-uio



Save the configuration by backing out. Keep double tapping Esc until you’re asked to save.

5. Configuring the Linux Kernel

It is now time to look at the Linux Kernel to see if there’s anything we should change. The only item we need to change is to enable the UIO drivers. Run the following PetaLinux command:

>> ***petalinux-config -c kernel***

Look here for the drivers:

Device Drivers→Userspace I/O drivers→

There you’ll find the following two drivers enabled as loadable modules. You *can* keep them as loadable module, but you can change them to be inserted by default into the kernel instead of loading them manually. Select the following two drivers and click the spacebar until you see a * on them as shown below:

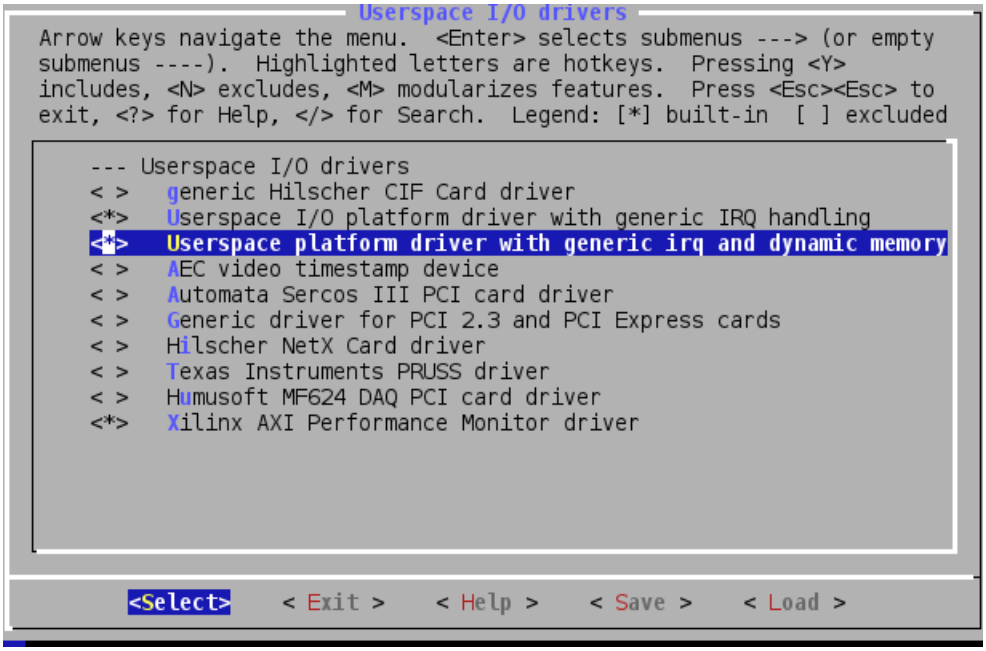


Figure 4

After enabling the UIO drivers into the kernel, now we'll need to disable some power management features to allow us to debug the PL via JTAG. If you don't disable these power management features, Linux will hang while JTAG to HW Server has a connection. Below are the options you'll want to disable:

CPU Power Management→CPU Frequency Scaling

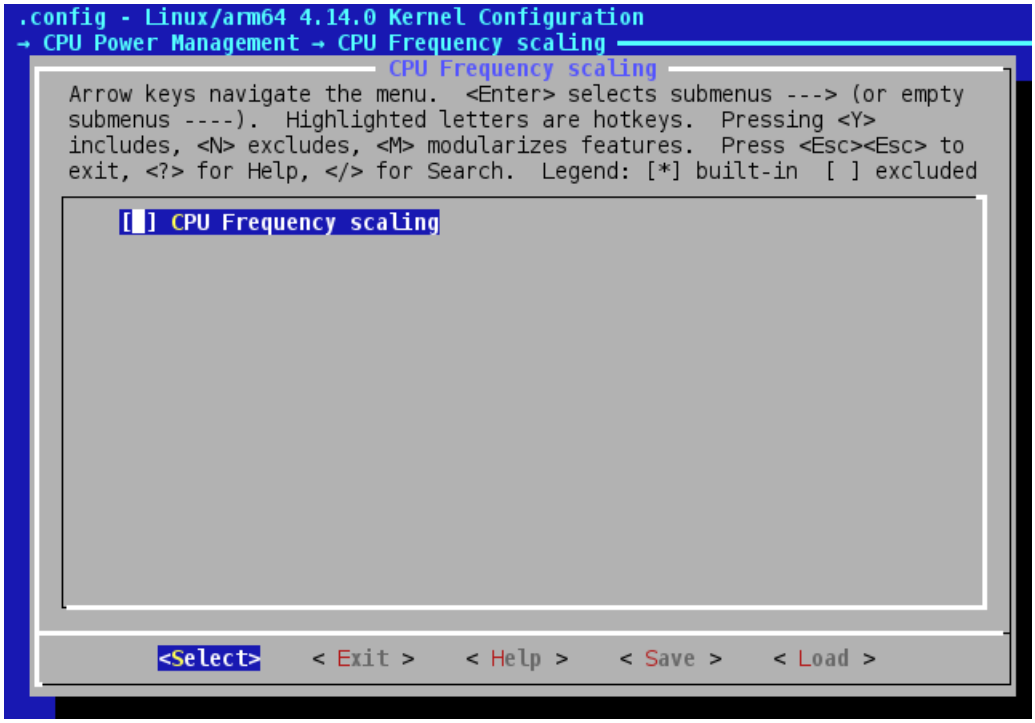


Figure 5

Also disable CPU idle PM Support located here:

CPU Power Management→CPU Idle

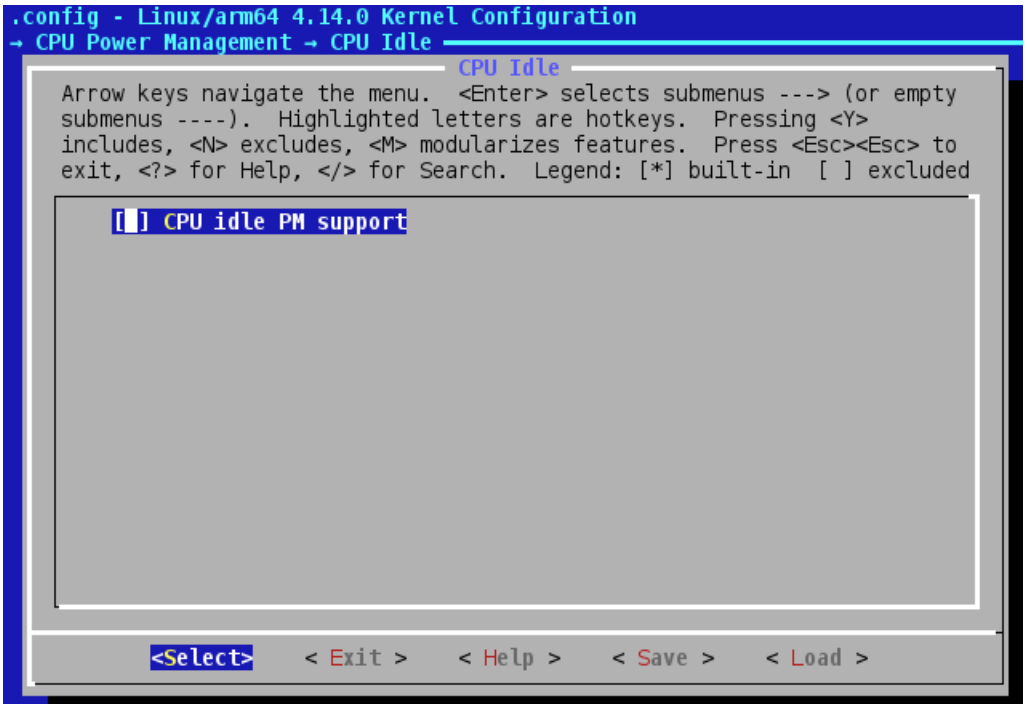


Figure 6

While you're here, look for other drivers to see what's offered, but you're free to back out by double tapping Esc till you're out.

6. Configuring the Root File System

Now we'll set up the Root File System to see if there are any libraries missing, or any other items you wish to have the root file system. It just so happens that we don't need anything, however you can look through the root file system options to see if there's anything to change. Here's the command to run:

>> ***petalinux-config -c rootfs***

This will also pop up a menuconfig you can traverse through. When you're done looking through, go ahead and close it.

7. Editing the device tree in PetaLinux

Within the PetaLinux directory structure, there's a directory where the user can provide device tree modifications. These modification can override previous entries, delete entries, or add completely new ones. An example device tree file is located in the "device_tree" directory of this lab. Place the system-user.dtsi file in the "./project-spec/meta-user/recipes-bsp/device-tree/files" directory. Feel free to look through the device tree file to see what it's doing. Below is a screenshot of the primary device tree updates:

>> ***cp ../../device_tree/system-user.dtsi ./project-spec/meta-user/recipes-bsp/device-tree/files/***

```
/include/ "system-conf.dtsi"
/ {
};

&axi_bram_ctrl_0 {
   compatible = "generic-uio";
};

&axi_bram_ctrl_1 {
   status = "disabled";
};

&counterip_0 {
   compatible = "generic-uio";
};

&gem3 {
  phy-handle = <&phyc>;
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_gem3_default>;
  phyc: phy@c {
    reg = <0xc>;
    ti,rx-internal-delay = <0x8>;
    ti,tx-internal-delay = <0xa>;
    ti,fifo-depth = <0x1>;
    ti,rxctrl-strap-worka;
  };
};
```

Figure 7

The '&' symbol indicates that it is updating a previous entry; it's how entries are referenced. This update will link the bram controller and the HLS IP to the UIO drivers. This will also allow the gigabit Ethernet to work properly by providing Linux the PHY it's using on the board.

8. Build the PetaLinux project

The time has come to build your PetaLinux project. This will build all the software you need for this demo. If this does not build, please seek assistance immediately. Either find the presenter or the processor specialist to help.

>> ***petalinux-build***

This can take a while to build, so occasionally check back with the build so issues can be addressed and time isn't lost. Now is the time to get back to emails, or to skip to the prebuilt folder; if you have a ZCU102.

9. Create the bootable image

In step 8, all the linux components were built. However, that doesn't mean you have the boot files. In order to boot on ZynqMP, you need bootgen to package up the following components:

- FSBL
- PMU Firmware
- ATF
- Bit – optional as the bit can be loaded later
- U-Boot

In order to create the bootable image for an SD card, run the following command:

>> ***petalinux-package --boot --fpga ../../bits/design_1_wrapper.bit --u-boot ./images/linux/u-boot.elf --atf ./images/linux/bl31.elf --pmufw ./images/linux/pmufw.elf --fsbl ./images/linux/zynqmp_fsbl.elf --force***

This will create a BOOT.bin which can be copied onto your SD Card. However, this is only the bootable image which will only boot up u-boot. You also want to add your linux image. For the Linux image, you can copy the image.ub file onto your SD Card. The image.ub file contains the following:

- Linux Kernel you configured
- Root File System you configured
- Device tree Blob (compiled device tree)

All three of these are wrapped in a FIT image so that u-boot can read it and know how to boot it.

10. Boot Linux

Use your terminal of choice and select the BAUD rate to 115200. I personally use Tera Term, but you can use which ever you'd like.

Next, ensure your mode pins are set to boot from SD. This will depend on your board and the revision.

Power on the board and watch the terminal fly…

What you should see at first is the FSBL running. One nice feature of the FSBL is that it prints out the day and time it was built. After the FSBL, you should see u-boot run. There's a point during u-boot where you can halt u-boot from booting Linux if you feel like playing around in u-boot. However if you want to proceed, just let it timeout. Lastly, you'll see a bunch of messages on Linux booting. For the most part, these print messages are printk's from kernel level code. Lots of these messages are from drivers. You can display these messages in the terminal post-boot with the dmesg command.

After Linux boots, you'll notice that several components are mapped to uio drivers. cd into the /sys/class/uio directory and notice there are multiple uio directories there. Go into each directory and cat the name file to see which component it is. Below is an example:

>> ***cat /sys/class/uio/uio0/name***

This should tell you the component you can talk to in the Linux application you're about to make.

11. Build your Linux application.

Open SDK on your local machine; you'll be debugging later which is why it's best to do it locally. Select a clean directory for a workspace. Next, select on "Create Application Project." For the next window, select the following selections and select Next:

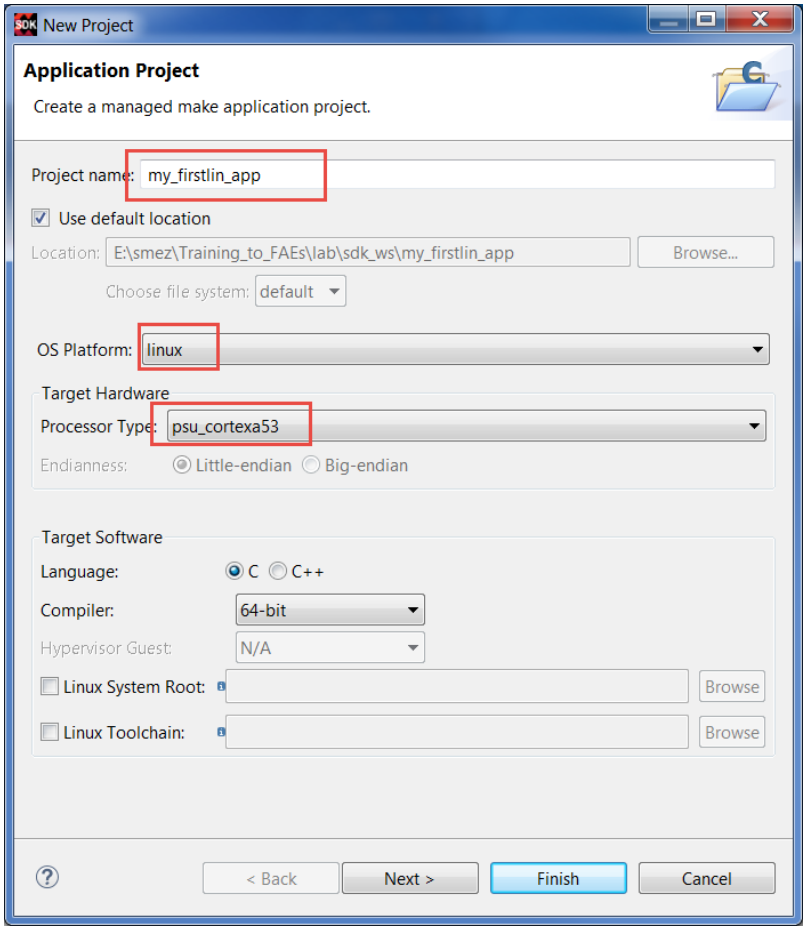


Figure 8

Lastly, select the Hello World Application.

Now copy the elf file in the Debug directory onto the SD card and boot Linux. Since the application is on the SD card, we'll have to mount the SD card to read it. Mount the SD card by creating a new directory and running the following command:

>>*mount /dev/mmcblk0p1 ./your_new_dir*

Now the files will show up in you_new_dir and you can execute the elf to see hello world app run.

Now that you've run hello world, go back to the hello world app and copy the contents in the /Linux_app/UIO_BRAM/hellouio.c file into the helloworld.c file in the SDK project. Now look at line 49 of the application and notice that /dev/uio0 is being opened:

```
47    printf("Hello World!\n");
48
49    uiofd = open("/dev/uio0", O_RDWR);
50    if (uiofd < 0) {
51        perror("open");
```

Figure 9

Figure 10

Ensure that /sys/class/uio/uio0/name is indeed the block ram.

Take this new elf and repeat the process of adding it to the SD card. However, before running it, open a hardware session in Vivado to see the AXI transactions make it into the PL.

While the hardware manager is up, use devmem to talk to the other peripherals in the PL and trigger on them. Below is the address map:

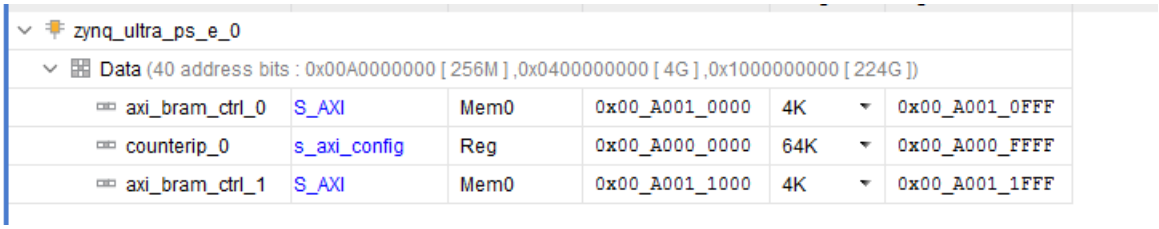


Figure 11

An example of a devmem read is below:

>>***devmem 0xA0010000 32***

An example of a devmem write is below:

>>***devmem 0xA0010000 32 0x12345678***

12. Debug the Linux application

While SDK is still up, open a new project by issuing the following:

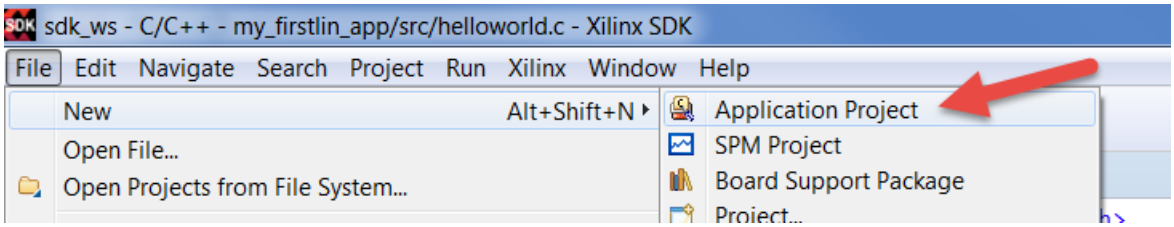


Figure 12

Use the same setup as before except give it a different name and choose the empty application:

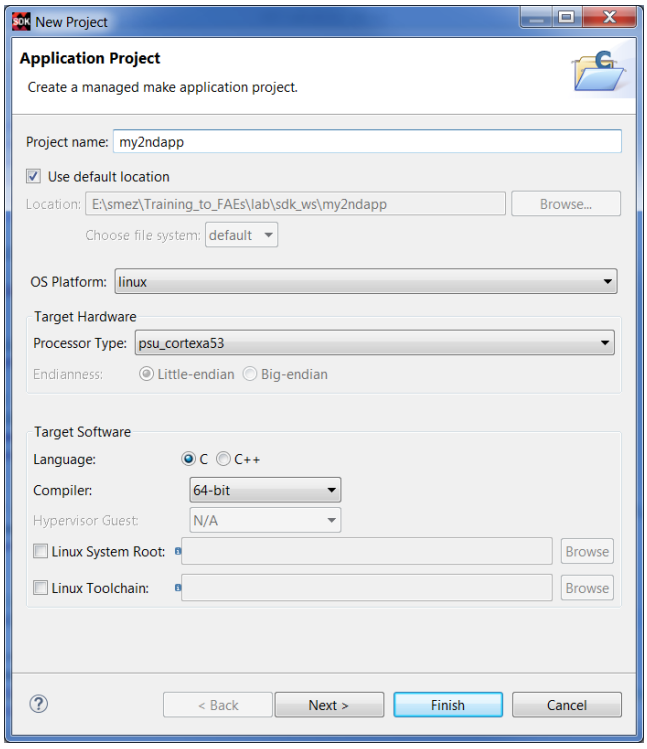


Figure 13

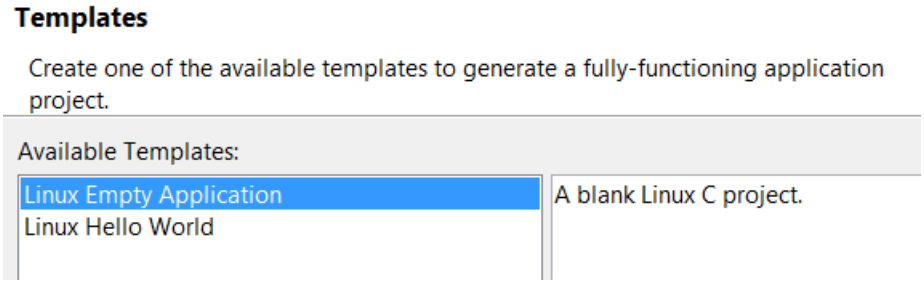


Figure 14

Right-click the src directory for the second application and select to Import as shown below:

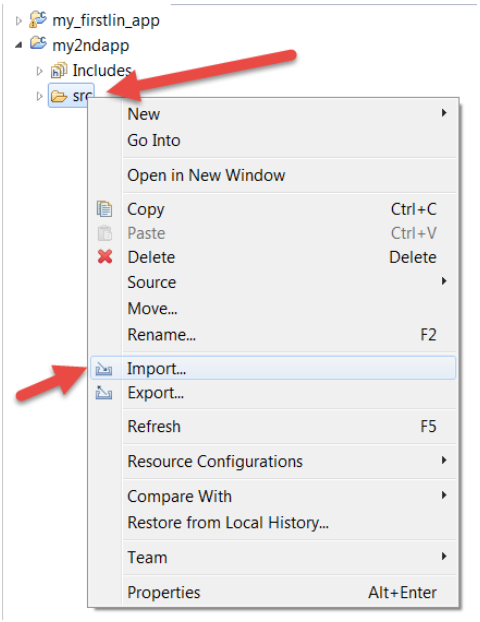


Figure 15

In the next pop up window select File System:

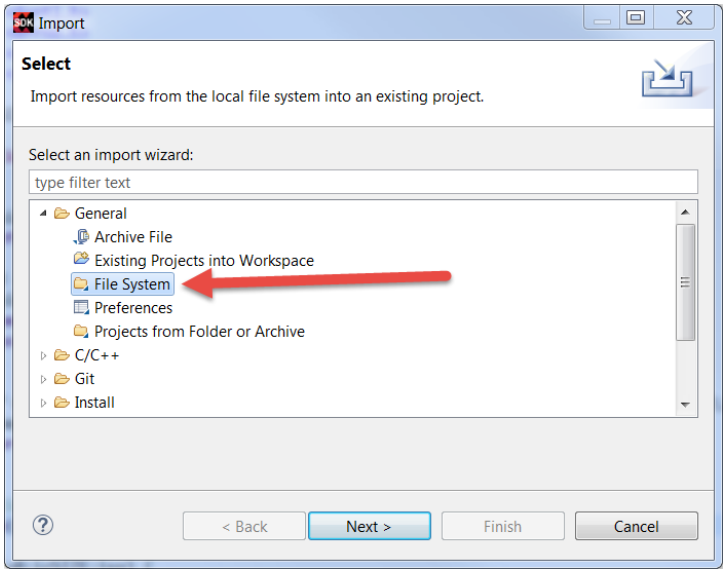


Figure 16

Next, track down the UIO_interrupt_counterIP app and select to add all the sources as shown below:

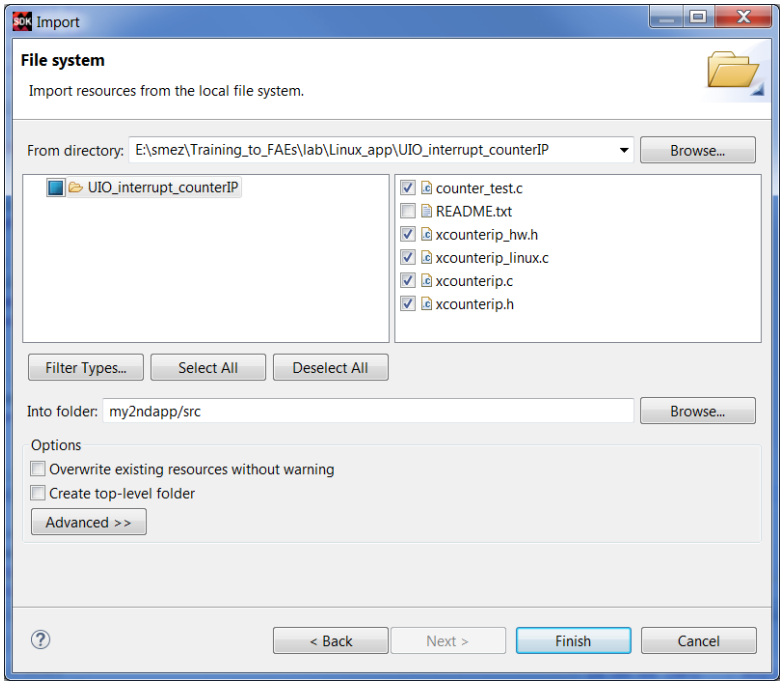


Figure 17

The application to debug is now added in the project. In order to debug the application, you will need to connect the Ethernet cable from the Zynq device to your laptop. This is where you need to ensure your firewall is off. Since your laptop is likely not running a DHCP server, you'll likely need to assign an IP address yourself. If you're in MS Windows, then you'll need to set your IP to be static in your network connections(if you don't know how to do this, please seek assistance). You will also need to set a static address on Zynq. One way of doing this is with the ifconfig command as shown below:

>>***ifconfig eth0 192.168.1.111 netmask 255.255.255.0***

You can verify if this stuck by typing ifconfig without any arguments.

Now open a terminal on you're your laptop and try to ping the address you just gave Zynq as shown below:

>>***ping 192.168.1.111***

Hopefully you get a response…

If you do, now you're ready to debug in SDK. Within SDK, double click the Linux Agent under Linux TCF Agent as shown below:

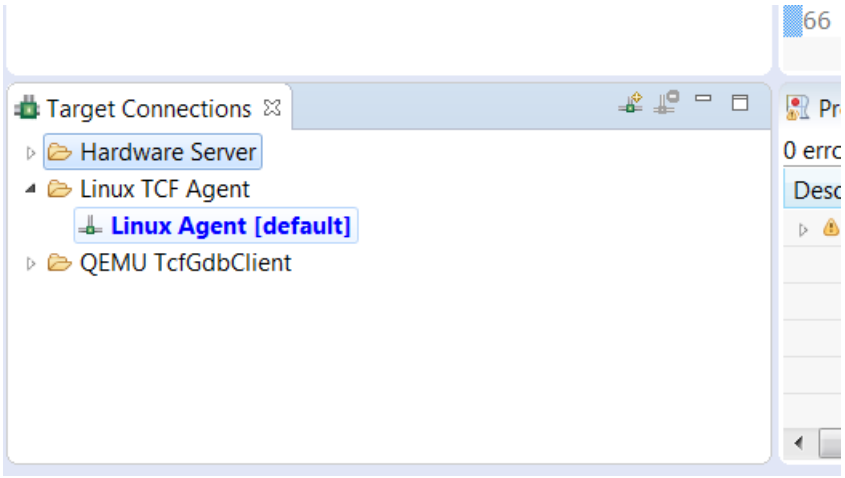


Figure 18

In the next pop up select the IP address of zynq and keep the port the same:

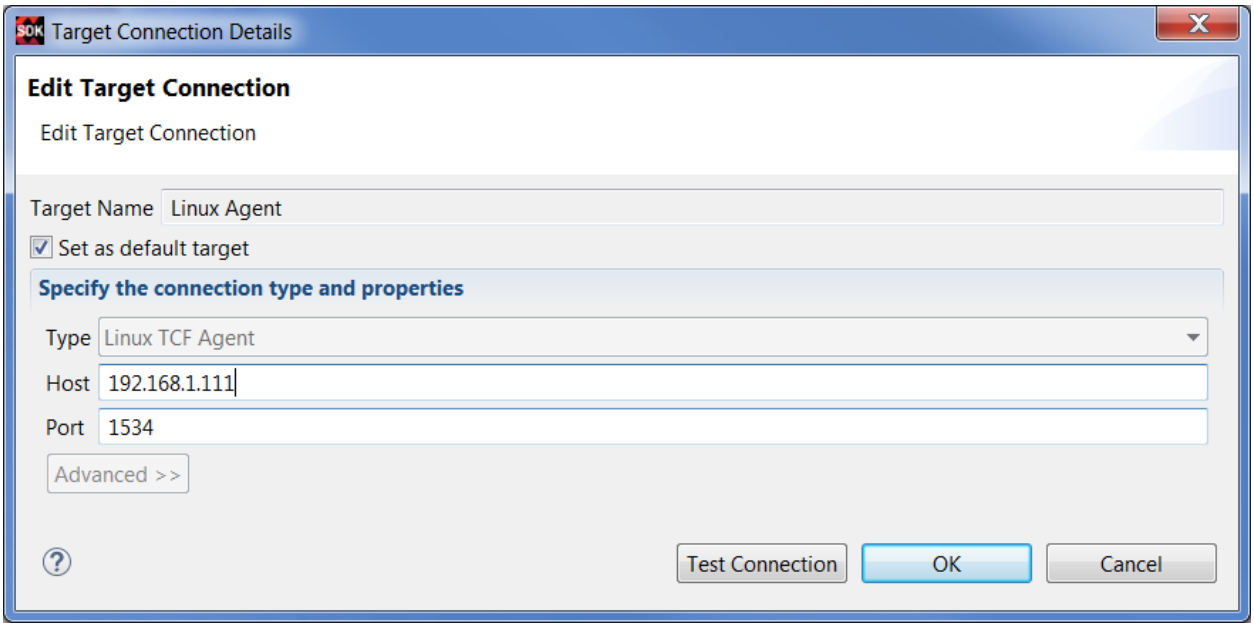


Figure 19

Test the Connection and make sure it's successful.

After the successful test, right click your application and select Debug As->Launch on Hardware

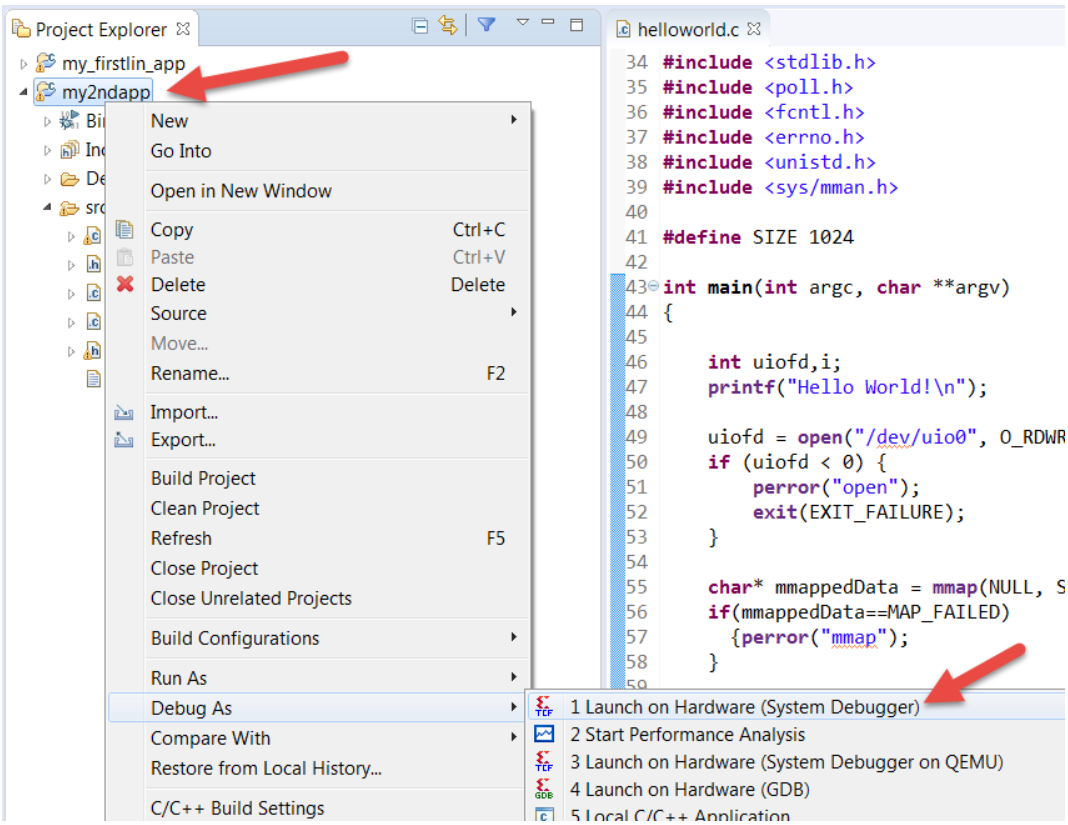


Figure 20

Now you can step through your code and look at all your structures. This can be very helpful for a newbie Linux person…

Also note that you can access the application within Zynq's Linux in the /mnt/ directory. The elf should be there too.