UltraZed-7EV PetaLinux Tutorial

Kacen Moody
BYU Electrical and Computer Engineering

4/18/20

Purpose

The goal of this tutorial is to build a PetaLinux 2019.2 image for the UltraZed-EV board (specifically, the UltraScale+ MPSoC using the Xilinx XCZU7EV-FBVB900 FPGA), including some advanced image configuration features. Expected outcomes are the following:

- Create a working PetaLinux image and its corresponding boot image, to be booted from an SD card, and accessed via UART (through a micro USB port) and SSH
- Learn to navigate the PetaLinux tools, including how to base a project on an FPGA design and configure various features

Requirements

- Installed PetaLinux 2019.2 tools (page 9, UG1144)
- Installed Vivado 2019.2 (or prior PetaLinux 2019.2 compatible version) with the UltraZed-7EV board files
- Correctly formatted SD card (page 65, UG1144)
- UltraZed-EV board with a carrier card that include an SD reader, Ethernet port, and micro USB port
- Basic understanding of Linux command line

1 Building Simple PetaLinux Images

1.1 Set Up the Environment

First, **the PetaLinux environment must be set up**. This is done by running a script in the command line and must be done every time a new terminal window is opened to use the tools. For a bash shell, the command is as follows: (page 13, UG1144):

```
$ source <path-to-installed-PetaLinux>/settings.sh
```

1.2 Create a Project

Next, we can **create and initialize a project using the petalinux-create command** (page 20, UG1144). The tools will create a root directory and populate it with various configuration and build files, so this command should be executed in the directory where you want the project's root folder to reside.

```
$ petalinux-create --type project --template zynqMP --name my_petalinux
```

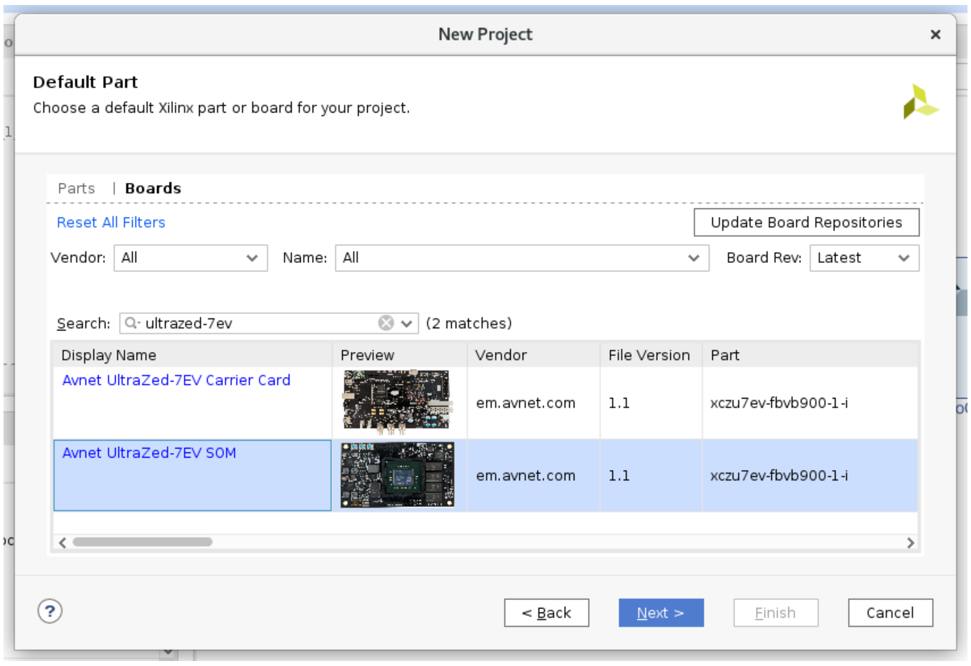
Template options are zynqMP, zynq, or microblaze, and as the UltraZed-EV is an MPSoC, the zynqMP is appropriate in this case.

Alternatively, a project can be created using a board support package (BSP) produced by Avnet, but that will not be discussed here. Refer to UG1144, page 16.

1.3 Importing Hardware and Configuring the Image

With the project initialized, go into the project's root directory in the terminal. The next step is to configure the image in accordance with the hardware present on the UltraZed-EV. Implied here is that while the template we used in the previous step will set up the framework for the general zynqMP architecture, the PetaLinux tools must have an idea about how the processor system is to be configured on whichever MPSoC the project is targeting. To do this we must **build a Vivado project and export either the HDF or XSA file**, which will then be imported into the PetaLinux tools. Below, we will describe the steps for building a very slimmed down project containing just the ZynqMP processor. If you have already built a Vivado project, skip ahead to number 8.

1. Start Vivado 2019.2
2. Under "Quick Start" **click on "Create Project" and set up the project using the defaults** without any additional source files (it doesn't matter what the name or location of the project are). When you arrive at the "Default Part" page, **select either the "UltraZed-7EV SOM" or "UltraZed-7EV Carrier Card" board** as shown in Figure 1, choosing whichever option corresponds with what you are targeting (visit the "Using a Custom Carrier Card" section for more information on which board to use). This way, the UltraZed board files can populate the project constraints using existing board peripherals if necessary. If these boards are not present in the list, you will likely need to install the UltraZed board files into Vivado before proceeding.



Default Part

Choose a default Xilinx part or board for your project.

Vendor: All Name: All Board Rev: Latest

Search: ultrazed-7ev (2 matches)

Display Name	Preview	Vendor	File Version	Part
Avnet UltraZed-7EV Carrier Card		em.avnet.com	1.1	xczu7ev-fbvb900-1-i
Avnet UltraZed-7EV SOM		em.avnet.com	1.1	xczu7ev-fbvb900-1-i

Figure 1

3. After finishing the setup and waiting for Vivado to initialize the project, **click on "Create Block Design"** under "IP Integrator" on the left.
4. In the center of the new window that opens, **click on the plus sign to add an IP, and select "Zynq Ultrascale+ MPSoC."**
5. When the IP has been added, **click "Run Block Automation"** in the green strip above it and then click "OK." This will reduce the number of IO pins visible on the block but will populate it with typical features as shown in Figure 2.

   Notice that UART and SD are both enabled. On the block in the block diagram, there will only be the pl_resetn0 and pl_clk0 ports remaining as shown in Figure 3, which we will leave unconnected.

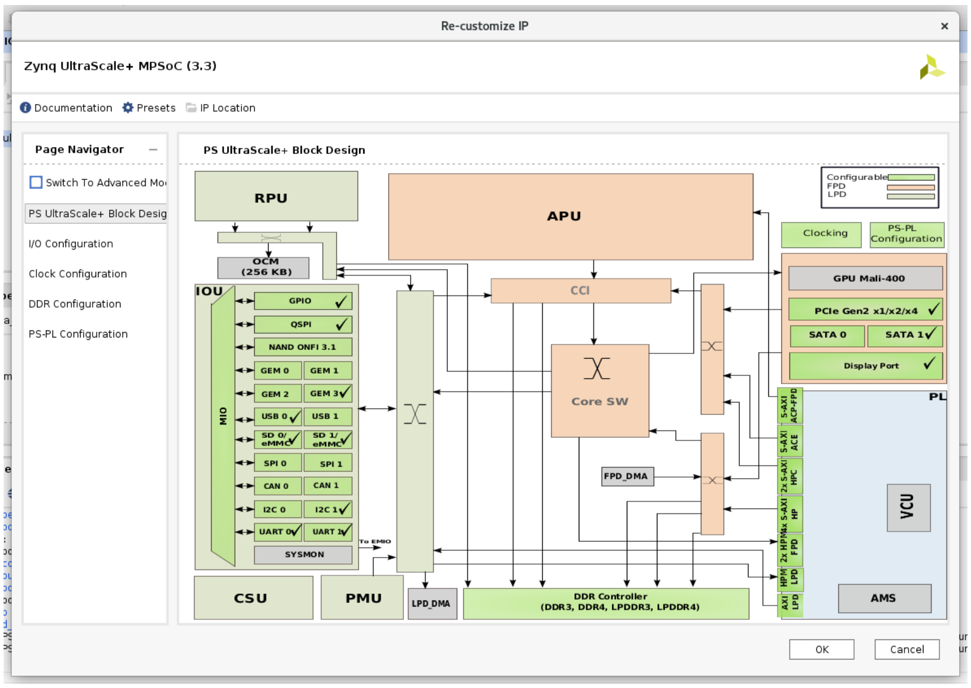
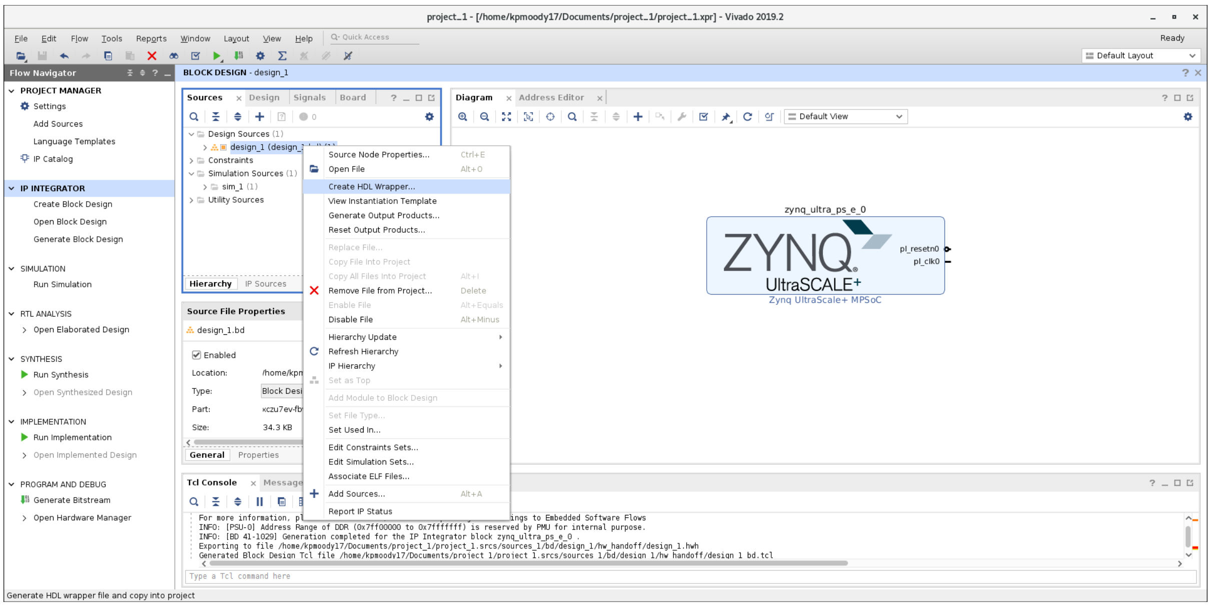


Figure 2

6. Next, in order to build the project, Vivado needs the block diagram to have a top level wrapper module, so in the upper left pane, **click on "Sources" and then right click on "design_1"** next to the little orange symbols. From the menu, **click "Create HDL Wrapper..."** and then click "OK." This is shown in Figure 3.

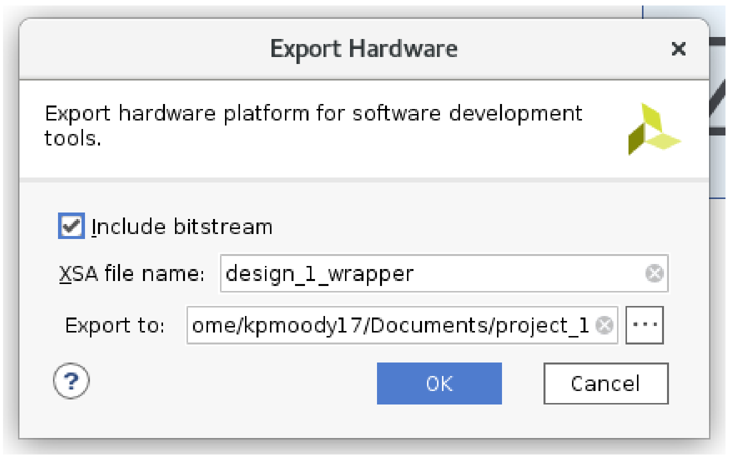


```
For more information, pl                                          ings to Embedded Software Flows
INFO: [PSU-0] Address Range of DDR (0x7ff00000 to 0x7fffffff) is reserved by PMU for internal purpose.
INFO: [BD 41-1029] Generation completed for the IP Integrator block zynq_ultra_ps_e_0 .
Exporting to file /home/kpmoody17/Documents/project_1/project_1.srcs/sources_1/bd/design_1/hw_handoff/design_1.hwh
Generated Block Design Tcl file /home/kpmoody17/Documents/project_1/project_1.srcs/sources_1/bd/design_1/hw_handoff/design_1_bd.tcl
```

Figure 3

7. This is all for the processor so **click "Generate Bitstream"** below "Program and Debug" in the left pane. Click "Yes" if asked to save the project and then choose to launch runs first as well. It may take a while for the project to compile.

8. After the bitstream has been compiled, click on "File" in the upper left of the GUI and select "Export Hardware..." from the "Export" option toward the bottom of the drop down menu. In the window that comes up, check the "Include bitstream" box as shown in Figure 4 and click "OK." This step will **create the HDF or XSA file** that we want. (Vivado 2019.2 will create an XSA by default, while earlier versions of Vivado create an HDF.) Take note of where the output will be saved.



Export hardware platform for software development tools.

☑ Include bitstream

XSA file name: design_1_wrapper

Export to: ome/kpmoody17/Documents/project_1

Figure 4

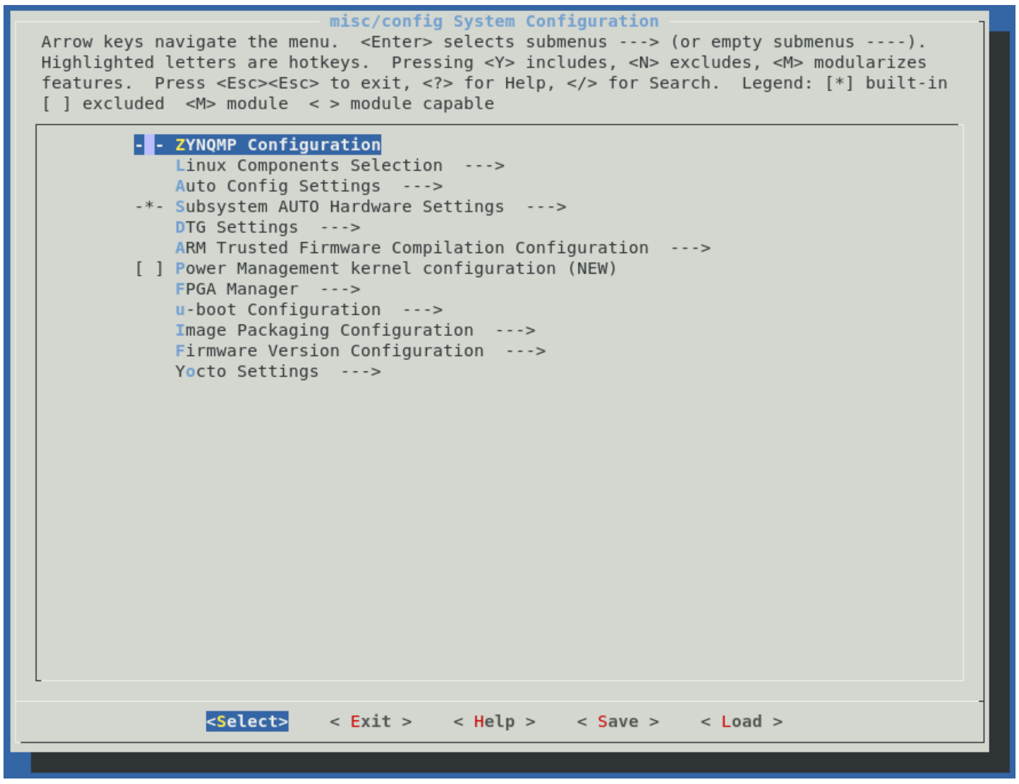
9. In the file system, navigate to the save location of the exported hardware. By default, the file will be called design_1_wrapper.xsa (or .hdf). **Copy this file into the root directory of the PetaLinux project we have already created.**

10. We can now **import the exported hardware file into the project** using the petalinux-config command in the terminal as follows (page 23, UG1144):

```
$ petalinux-config --get-hw-description=.
```

This command will start the configuration process by extracting information from the hardware file.

Notice that the command searches within a directory (specified in this case by the period after the equals sign) and does not look for a specific file name. Don't include the name of the hardware file in this command, just give the path of the directory where the hardware file is located. After importing the file, the command line GUI shown in Figure 5 will open.



```
                    misc/config System Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

          -*- ZYNQMP Configuration
              Linux Components Selection  --->
              Auto Config Settings  --->
          -*- Subsystem AUTO Hardware Settings  --->
              DTG Settings  --->
              ARM Trusted Firmware Compilation Configuration  --->
          [ ] Power Management kernel configuration (NEW)
              FPGA Manager  --->
              u-boot Configuration  --->
              Image Packaging Configuration  --->
              Firmware Version Configuration  --->
              Yocto Settings  --->

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

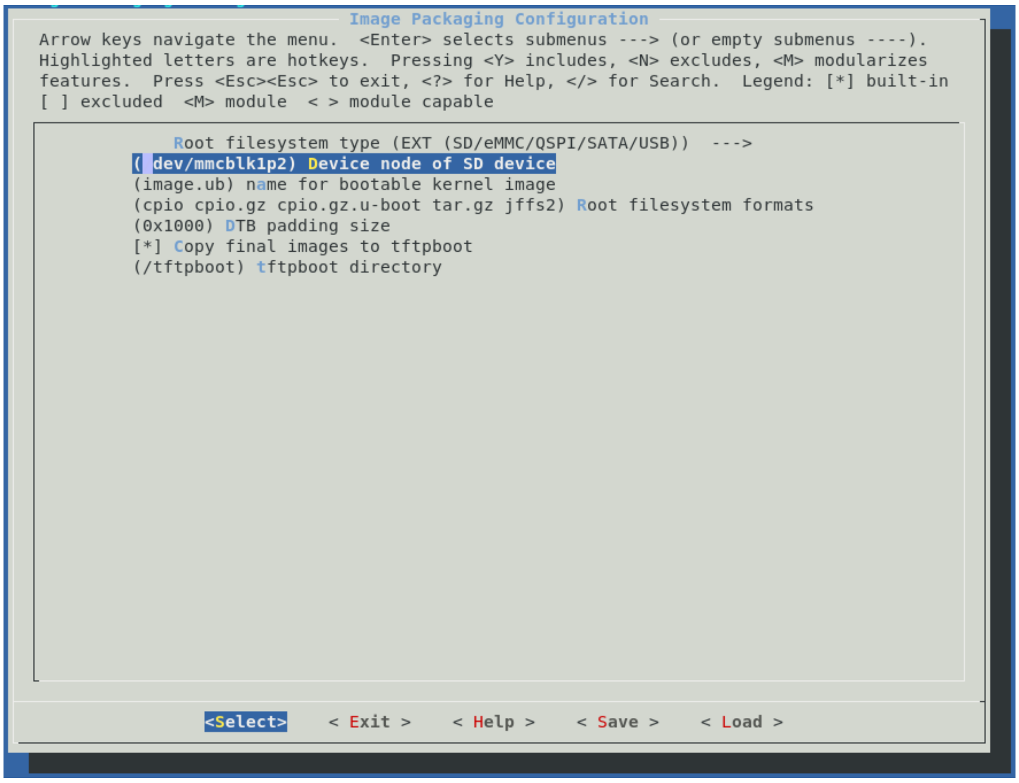
Figure 5

11. This GUI will allow us to control various build options, but we want the tools to handle everything for us so the only thing we will do is change the boot location to be the SD card. To do this, select the "Image Packaging Configuration" option and then select "Root filesystem type." This will bring up a smaller window with the options, from which we will **select the "EXT" option before "other" which includes "SD"** (among other things) as shown after selection on the first line in Figure 6.

12. After the type has been selected, a new menu item called "Device Node of SD device" will have appeared below it. **Select this option and change mmcblk0p2 to mmcblk1p2** as shown on the second line in Figure 6.

    Additional Information: The change in the previous step will tell the boot sequence to search for the ext file system type on the SD card (in our case) when setting up the filesystem.

    This step tells the boot sequence to search on the second partition of the SD card for the filesystem (hence the "p2"), and we change the 0 to a 1 because on boot up, every memory driver is given an ID based how the hardware is set up. According to the Avnet documentation for the UltraZed-EV board, the eMMC flash device is given the device name SD0 which corresponds to mmcblk0, while the SD card is called SD1 and is called mmcblk1. Refer to page 14 of the UltraZed documentation (this link may require login and download of the pdf) for more information.



```
                    Image Packaging Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
 [ ] excluded  <M> module  < > module capable

            Root filesystem type (EXT (SD/eMMC/QSPI/SATA/USB))  --->
        ( dev/mmcblk1p2) Device node of SD device
        (image.ub) name for bootable kernel image
        (cpio cpio.gz cpio.gz.u-boot tar.gz jffs2) Root filesystem formats
        (0x1000) DTB padding size
        [*] Copy final images to tftpboot
        (/tftpboot) tftpboot directory

           <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 6

13. After these two items have been changed, **save the configuration and exit the GUI** (leaving the save location as it is). It may take a little while for the configuration to finish, especially the first time.

14. The last necessary step of the configuration process is to modify the device tree entry for the SD card block to disable write protect. To do this, go to project-spec/meta-user/recipes-bsp/device-tree/files/ starting from the project root directory. In this directory, **open the system-user.dtsi**, which at this point should be empty, and **add the following lines so that the contents of the file look as follows**:

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

/* SD1 with level shifter */
&sdhci1 {
        status = "okay";
        max-frequency = <50000000>;
        no-1-8-v; /* for 1.0 silicon */
        disable-wp;
};
```

Additional Information: This system-user.dtsi file will be parsed by the PetaLinux tools and used to add or overwrite features of modules which reside in other device tree files automatically generated by the tools.

The additional device tree block that we have added is taken from the system-user.dtsi file in the out-of-box BSP produced by Avnet. There are a few things going on here. The first is that we enable the SD driver by setting the status to "okay." Then we disable write protect with "disable-wp," which is necessary because the default configuration prevents drivers from changing memory they can access as we need to be able to do on the SD card. The other two lines essentially just configure the maximum frequency (which per Avnet docs is 52MHz) and configure the driver to have the level shifter which is required by the SD card reader. Without the addition of this node, the boot process will begin but the kernel won't be able to take control of the root file system, causing it to hang.

1.4 Building the PetaLinux Image

Now that things have been configured, we can **build the PetaLinux kernel image**. This is done using the petalinux-build command with no additional tags as shown here (page 25, UG1144):

```
$ petalinux-build
```

Depending on the available hardware of the host OS, this build can take around 20 minutes or over an hour. The result will be an image.ub file that will contain the PetaLinux kernel with all of the hardware and software configurations we have given it.

1.5 Packaging the Boot Image

Next, we nee to **compile the boot image** using the petalinux-package command as shown (page 27, UG1144):

```
$ petalinux-package --boot --fpga --u-boot
```

Additional Information: The tags in this command tell the tools what is being packaged and what to include. The boot tag in this case will request that the BOOT.BIN file be produced, while the other two tags tell the tools to include the bootloader (u-boot) and the FPGA bitstream (fpga) within the BOOT.BIN. For the latter two tags, the locations of the files can be specified if they have been created separately, but if not the default location used is wherever the PetaLinux tools created and placed those files originally. We point this out because we can only use the FPGA tag without a file location afterward due to the fact that we included the bitstream in the exported file from Vivado. If the bitstream were not included, using this tag as we have would produce an error. More information about this command (and others) can be found in UG1157 (the link references page 20 where petalinux-package begins).

1.6 Copying Files to the SD Card

Now that everything has been generated, we can **copy the image.ub and BOOT.BIN files over to the first partition of the SD card**. These images are found, again starting from the project root directory, in the images/linux/ directory. There are many generated image files but we only want these two.

In the second partition, which should have been formatted as ext4, we need to set up the filesystem. To do this, **copy the rootfs.tar.gz zip folder to the second partition and extract it using the following terminal command** (while the terminal path is in the rootfs partition). You will need to put in your user password for the command to execute because it must be run with "sudo" privileges. After the filesystem has been extracted, the rootfs.tar.gz archive can be removed.

```
$ sudo tar -xzf rootfs.tar.gz
```

Additional Information: The .tar.gz archive format preserves access permissions so that when the filesystem is zipped and then extracted elsewhere, all of the files and directories have the same permissions that they did originally. This is required for proper booting because the kernel expects certain directories to be set up beforehand with restricted permissions. The extraction command must be run with root (sudo) privileges for all restricted permissions to be set up correctly.

2 Booting Petalinux

In order to boot PetaLinux and access it through the network using SSH a few different steps must be followed.

2.1 Preparing the Workstation

1. **Insert the SD card** into the reader on the UltraZed's carrier card.
2. **Make sure that the boot configuration switches on the board are set to boot from SD** as described in the Avnet documentation, which will most likely be "OFF-ON-OFF-ON," [1:4].
3. **Connect a USB-to-micro-USB cable between the board and your computer.**
4. **Connect the board's Ethernet port to the network your computer is connected to.**
5. Before turning the board on, **run the screen command in your terminal** as follows:

   ```
   $ sudo screen /dev/ttyUSB1 115200
   ```

   This will allow you to observe the boot messages as the board starts up and see if any errors occur. It will also allow you to log in to the board via UART so that you can find out the current IP address that the board is given on the network.

   Note that the command assumes the USB port on your computer is called "USB1," which may not be the case. If not, you may need to run dmesg in the terminal to see where the board was connected, or try different USB# ports as they appear in the /dev/ directory.

   Additional Information: Notice that command uses a baud rate of 115200. This is due to the default configuration set up by the PetaLinux tools.
6. **Power on the board.** In the terminal you should see boot messages begin almost immediately and describe things like the bootloader, PetaLinux build version, and then list the kernel boot messages as the system is loaded.

2.2 Gaining Access to PetaLinux After Boot

1. When PetaLinux is done booting, you will see "my_petalinux login:" followed by the cursor in the terminal (assuming your project is named my_petalinux). **Enter the username and password**, which are both "root". You should then see "root@my_petalinux:~#".
2. You are now logged into PetaLinux. **To find the IP address, use the ifconfig command**, which will produce several lines of information starting with those shown here:

   ```
   root@my_petalinux:~# ifconfig
   eth0      Link encap:Ethernet  HWaddr 00:0F:36:01:23:01
             inet addr:10.2.116.194  Bcast:10.2.116.255  Mask:255.255.255.0
             inet6 addr: fe81::20d:35fe:fe00:2202/64 Scope:Link
   ```

   The part we care about is "inet addr:10.2.116.194" which will allow us to SSH into the board.
3. In a new terminal, **SSH into the UltraZed** using the following command, substituting whatever IP address you saw in the previous step:

   ```
   $ ssh root@10.2.116.194
   ```

   Again, this should connect to the board and allow you to enter the same credentials as before to login.

3 Advanced PetaLinux Configuration

The PetaLinux tools provide extensive user control over the kernel image, its utilities and features. This section goes over some resources and examples to outline how additional changes to the kernel can be made.

3.1 PetaLinux Device Trees

A device tree is a text file that defines various memory-mapped peripherals and their configurations. The information contained in the device tree will be parsed and compiled into the kernel image, and during the first stages of the boot process, necessary nodes will be initialized for use by the kernel during execution. The following two sites are a good place to start as you begin to learn more about how they work.

1. Device Tree Tips is a page on the Xilinx Wiki which "is intended to be a collection place for tips and tricks related to device trees."

2. Device Trees for Dummies is a slide presentation which goes over device tree background and syntax for typical nodes. This information is for Linux systems at large, so the PetaLinux device trees may contain some differences, but the idea and general structure is the same.

Because it is the memory-mapped peripherals that the kernel needs to be aware of, FPGA hardware designs which contain memory-mapped IP should include those IP in the device tree. Thankfully, when configuring a project based on the hardware file exported from Vivado, the PetaLinux tools will add nodes for each memory-mapped IP to auto-generated device tree files without the user having to do any of this manually. The following steps show how to observe this:

1. In a PetaLinux project like the one made previously that has been configured but not yet compiled, **notice that the components/plnx_workspace/ directory only has the conf/ folder in it**. This plnx_workspace/ directory will eventually contain the build products for device trees and other things.

2. Now, build the image. After it completes, you will **notice the addition of a device-tree/device-tree/ directory branch in the plnx_workspace/ directory, which is populated with various .dtsi files** among other things.

   Additional Information: When the petalinux-build command is run, the default device trees are auto-generated and parsed together with the system-user.dtsi, which, as we have mentioned before, allows the user to add in their own nodes and device configurations. If you want to build the device tree without building the full kernel image, you can run the petalinux-config command as follows after importing the HDF or XSA into PetaLinux.

   ```
   $ petalinux-config -c device-tree
   ```

   This will produce all of the same files observed in the following steps.

3. **In this device-tree/ directory, open the pl.dtsi (if present) and notice that it is mostly empty**, indicating that our FPGA design doesn't have any memory-mapped peripherals. This makes sense because our design only has the zynqMP processor block in it.

4. Now **say that we have an FPGA design in Vivado that has a memory-mapped IP called "axi_lrf_controller_0,"** and which is mapped to the PS memory base address 0xA0000000. After building the bitstream, exporting the hardware from Vivado and importing it into a PetaLinux project as we did previously, **the petalinux-build command will create the pl.dtsi and populate it with something similar to the following** (along with the header and perhaps a node for PL clocks):

   ```
   / {
       amba_pl: amba_pl@0 {
           #address-cells = <2>;
           #size-cells = <2>;
           compatible = "simple-bus";
           ranges ;
           axi_lrf_controller_0: axi_lrf_controller@a0000000 {
               clock-names = "lrf_word_clk", "S_AXI_ACLK";
               clocks = <&misc_clk_0 >, <&zynqmp_clk 71>;
               compatible = "xlnx,axi-lrf-controller-1.0";
   ```

```
            reg = <0x0 0xa0000000 0x0 0x1000>;
        };
    };
};
```

5. This node is essentially saying that the PL exists, and then lists the memory mapped IPs it contains and their configurations within it. **Notice that the axi_lrf_controller_0 is now in the device tree**, and that it has been given the same address that it had in Vivado. You can also see how it shows which clocks are connected to it on lines 8 and 9, and that the block will need 0x1000 range of address space on line 11. This size parameter should also be the same as the size in Vivado.

   Additional Information: To find the default SD memory driver in the device tree, open zynqmp.dtsi in the same folder. This will have all driver nodes that are included in the processor by default. Search for "sdhci" and you will see the two memory device drivers with all of their configuration material. You will recall that it is "sdhci1" that we overwrote previously to configure the UltraZed to boot from the SD card.

3.2 FPGA Manager Utility

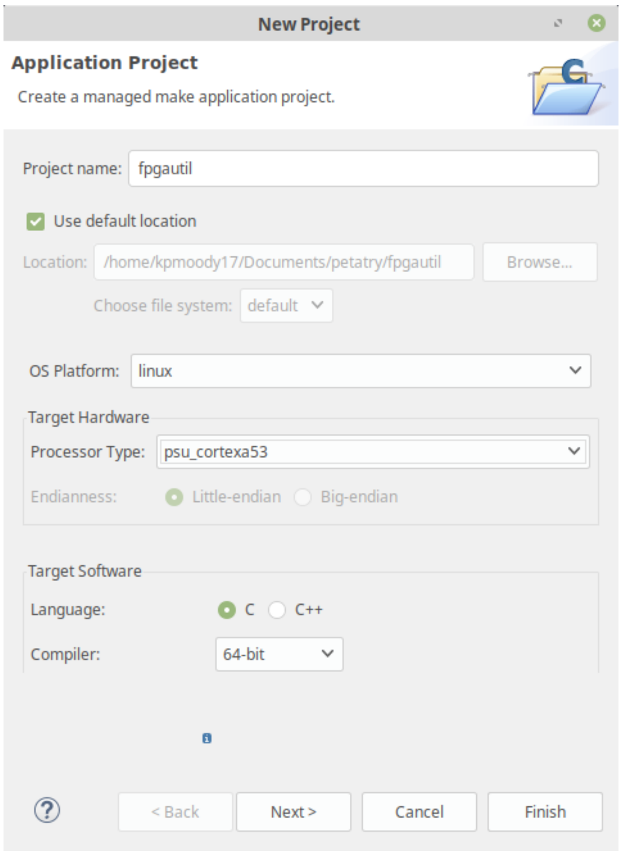
The FPGA Manager is a feature enabled by default in the PetaLinux configuration. One of the purposes of this feature is to allow the user to change the FPGA bitstream while booted into PetaLinux. In order to interface with the Manager properly, Xilinx provides the "fpgautil" utility, which the user can execute to upload a bitstream as described briefly in the following steps:

1. **Visit the** FPGA Programming **section of the** Solution ZynqMP PL Programming **Xilinx Wiki page.**

2. Under the subsection "Exercising FPGA programming using fpgautil", **click on the link "fpgautil.c" to download the utility source code.**

3. Next, we have to compile this source code to use it on the board, which can be done using Vitis or the Xilinx SDK. To do this in the SDK, first **open the IDE and set up an application project to operate on Linux, targeting the psu_cortexa53 processor, and using C**, as shown in Figure 7.

   Additional Information: The Xilinx SDK version used for these instructions was 2019.1, which is the last version before the SDK was absorbed into Vitis for 2019.2. Even though we are using 2019.2 for the rest of the tools, compiling the executable for this utility only needs to know that the platform is Linux and what kind of processor it will be running on. Getting a quick application project up and running from the SDK is several steps simpler than Vitis, which is why it was chosen here.



New Project

Application Project

Create a managed make application project.

Project name: fpgautil

[✓] Use default location

Location: /home/kpmoody17/Documents/petatry/fpgautil

Choose file system: default

OS Platform: linux

Target Hardware

Processor Type: psu_cortexa53

Endianness: [✓ Little-endian] Big-endian

Target Software

Language: [✓ C] C++

Compiler: 64-bit

Figure 7

4. If asked what kind of application project you want to create, the "hello world" template will work fine. When the project is set up, **copy the fpgautil.c file into the src directory and delete the helloworld.c file**.

5. **Save this project and build it.** If auto-build is not turned on, you can click the hammer in the upper left of the GUI or press ctrl+b on the keyboard to build the project.

6. In the console window, you should see "Build Finished" which indicates that the executable is ready in the Debug/ directory. Assuming your application project was called fpgautil as shown in Figure 7, **copy the fpgautil.elf executable from <project-root-directory>/fpgautil/Debug/ to a convenient location in the PetaLinux filesystem.**

   Additional Information: This can be done from the terminal using the scp command as shown here (assuming the board is booted and connected to the network as described previously, and also assuming that you have gone to the SDK project's root directory in the terminal first).

   ```
   $ scp fpgautil/Debug/fpgautil.elf root@10.2.116.194:~/
   ```

7. **Copy a bitstream to the same location as the fpgautil executable.** (Or elsewhere on the board, as long as you point to it correctly in the next command.)

8. **Program the bitstream onto the board using the executable as shown here:**

   ```
   $ ./fpgautil.elf -b design_1_wrapper.bit
   ```

   The "-b" in this command indicates that the file following it is the bitstream. If successful, the output should look something like the following:

   ```
   Time taken to load BIN is 190.000000 Milli Seconds
   BIN FILE loaded through zynqMP FPGA manager successfully
   ```

Additional Information: For more information on additional fpgautil options, visit the Xilinx Wiki page mentioned in the first step.

3.3 Unlocking Protected Device Memory (enabling /dev/mem)

From version to version, Xilinx will occasionally change the defaults for the PetaLinux configuration. One such example, when migrating from 2019.1 to 2019.2, is that Xilinx enabled a memory protection module which allows only the kernel to access device memory which is not owned by a device in the device tree. Prior to this version, the user could write programs, open device memory—called /dev/mem—and use Linux commands like "mmap" to get a pointer to any memory address and read from and write to that memory. Because some applications (like DMA) benefit from this flexibility, these next steps show how to turn off the memory protection module. This will also give a glimpse into another corner of the PetaLinux tools.

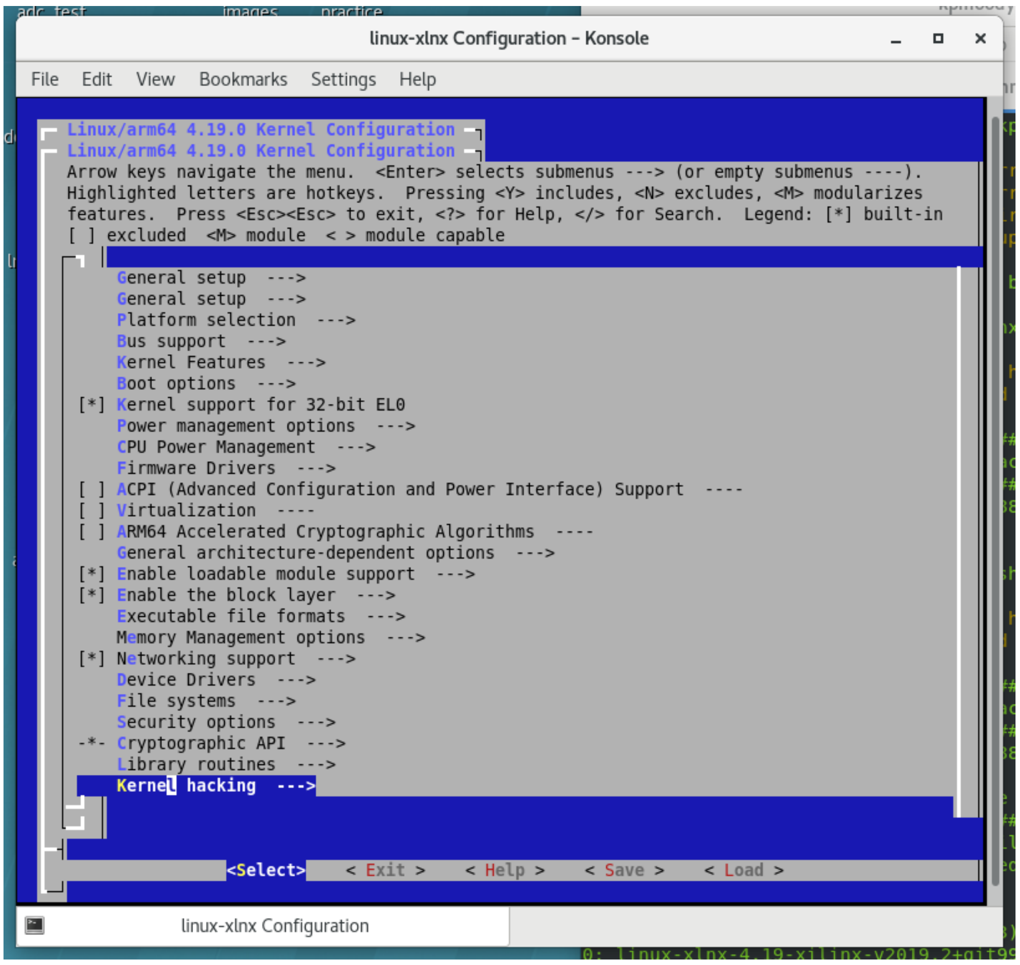
Caveat: Because these steps will remove the restrictions on memory access, there is greater inherent risk of inadvertently messing up memory contents. This is due to the fact that removing the protection module will allow the root user (which is the only default user in PetaLinux) to access kernel memory and restricted memory, such that they can read and write anywhere at will. Furthermore, this will give malicious users the opportunity to cause intentional damage if they manage to log in as root. So, it is recommended, where possible, to create a reserved memory node in the device tree that you can access freely in spite of the memory protection module, rather than disabling that module altogether. However, because this feature was disabled by default in the past, and for the sake of exploring the tools, the steps are shown here.

1. These instructions assume that a PetaLinux project has already been created and that the hardware file has been imported. If the images have been created as well, that's ok, but they will need to be rebuilt again after following the rest of the steps.

2. From in the root directory of the PetaLinux project, in a terminal which already has the PetaLinux tools environment set up, **run the petalinux-config command with the kernel component as shown** to open the kernel configuration GUI shown in Figure 8. Unfortunately this GUI has some formatting issues, but everything is still visible.

   ```
   $ petalinux-config -c kernel
   ```

3. **Click on "Kernel hacking" option**, which is highlighted at the bottom of Figure 8.


linux-xlnx Configuration – Konsole
File Edit View Bookmarks Settings Help
linux-xlnx Configuration

```
Linux/arm64 4.19.0 Kernel Configuration
Linux/arm64 4.19.0 Kernel Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
[ ] excluded  <M> module  < > module capable

    General setup  --->
    General setup  --->
    Platform selection  --->
    Bus support  --->
    Kernel Features  --->
    Boot options  --->
[*] Kernel support for 32-bit EL0
    Power management options  --->
    CPU Power Management  --->
    Firmware Drivers  --->
[ ] ACPI (Advanced Configuration and Power Interface) Support  ----
[ ] Virtualization  ----
[ ] ARM64 Accelerated Cryptographic Algorithms  ----
    General architecture-dependent options  --->
[*] Enable loadable module support  --->
[*] Enable the block layer  --->
    Executable file formats  --->
    Memory Management options  --->
[*] Networking support  --->
    Device Drivers  --->
    File systems  --->
    Security options  --->
-*- Cryptographic API  --->
    Library routines  --->
    Kernel hacking  --->

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 8

4. In the new menu that appears, scroll down using the arrow keys until you reach the line that says "[*] Filter access to /dev/mem." The asterisk in the brackets indicates that it is enabled, so **click "n" on the keyboard to disable it**.

5. **Save and exit the kernel configuration menu.**

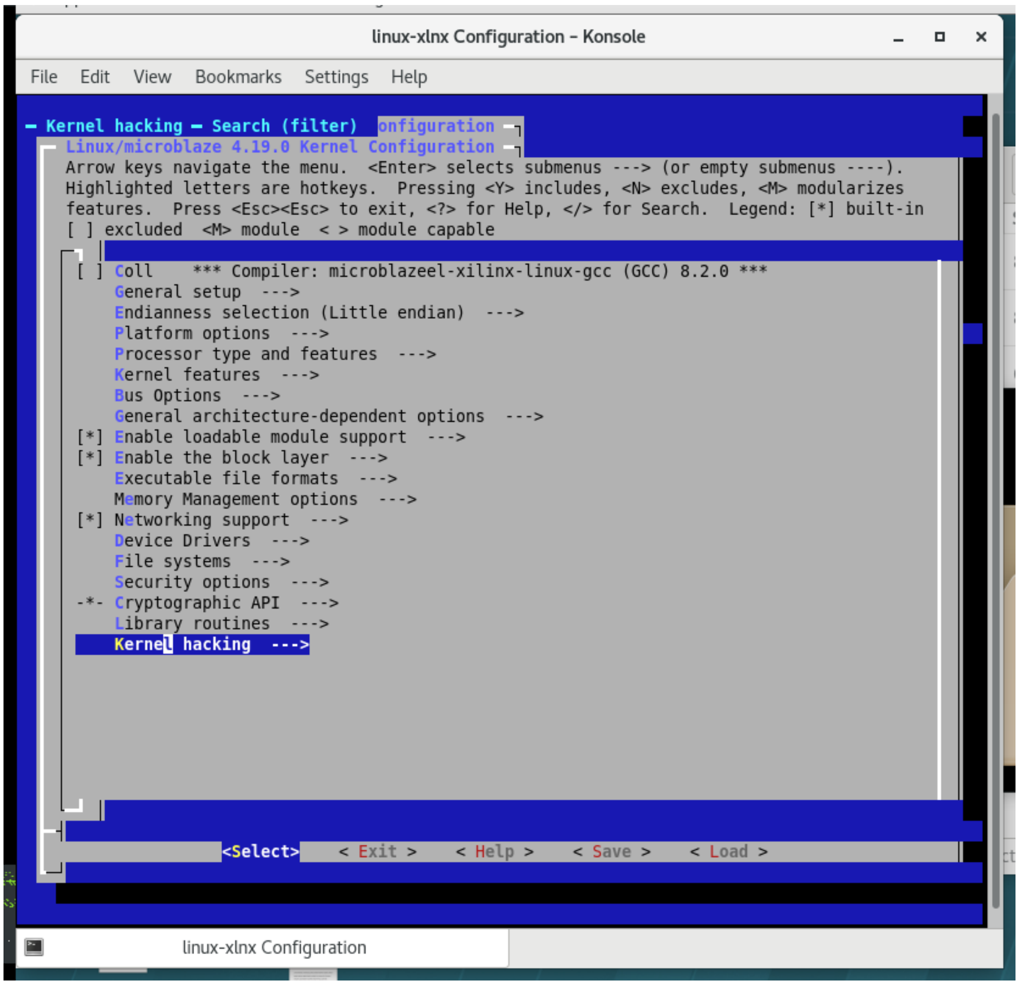
6. **Proceed as usual to build and package the images.** Now, when you boot up PetaLinux, you will be able to map to all device memory and use it in applications.

Additional Information: One key takeaway from these brief steps is that there are *many* configuration options that the user can control. In Figure 8 you can see configuration options for drivers, the filesystem, booting, power management, cryptography, etc. and within each of these, as we saw with the kernel hacking option, there are several modules which can be enabled or disabled.

4 Working With Other FPGA Boards

Because FPGA specifications differ from family to family, and because development board manufacturers can choose how they want to implement a given FPGA chip, building a PetaLinux image on a board other than the UltraZed will naturally involve some differences from the steps described in this tutorial. This is evident in this tutorial by the fact that we had to change the memory block device in order to boot from the SD card because Avnet elected to organize their memory drivers in the way that they did.

Similarly, considering designs like those that use the MicroBlaze processor, the degree of configurability may be quite a bit different from this tutorial because the processor architecture is different, the size is smaller, and general use-case is unique from the ARM processors on Zynq-7000 and ZynqMP chips. The kernel config window for a MicroBlaze PetaLinux project is shown in Figure 9 to show its contrast from Figure 8.


linux-xlnx Configuration – Konsole
File Edit View Bookmarks Settings Help
linux-xlnx Configuration

```
- Kernel hacking - Search (filter) onfiguration
  Linux/microblaze 4.19.0 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

   [ ] Coll    *** Compiler: microblazeel-xilinx-linux-gcc (GCC) 8.2.0 ***
       General setup  --->
       Endianness selection (Little endian)  --->
       Platform options  --->
       Processor type and features  --->
       Kernel features  --->
       Bus Options  --->
       General architecture-dependent options  --->
   [*] Enable loadable module support  --->
   [*] Enable the block layer  --->
       Executable file formats  --->
       Memory Management options  --->
   [*] Networking support  --->
       Device Drivers  --->
       File systems  --->
       Security options  --->
   -*- Cryptographic API  --->
       Library routines  --->
       Kernel hacking  --->

                <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 9

If you would like more information about using MicroBlaze with PetaLinux, one tutorial for PetaLinux on a MicroBlaze can be found here.

Even among boards with the same template (such as the UltraZed and Ultra96, for example, which both fall under "zynqMP"), there can be challenging differences because different boards handle hardware differently. While it is impossible to list everything that may be different between boards and templates here, we at least find it a valuable disclaimer to say that this tutorial will differ from others, and that it is likely that setting up PetaLinux on another board will require some adjustment.

5 Using a Custom Carrier Card

The carrier card used for this tutorial was not the Avnet carrier card, though the layout for it was created by mimicking Avnet's version minus the majority of their additional peripherals. In our design we only really needed the Ethernet port, micro USB, and SD reader, as well as a couple of our own peripherals, so in the long term we needed to migrate away from Avnet's card anyway.

Though intuitive in some ways, it is worth saying that our custom carrier card was designed to mimic the Avnet card because doing so allows us to use the default configurations set up by Vivado, PetaLinux, and Avnet's UltraZed-7EV SOM board. If the Ethernet port, for example, were to be connected to different IO, that may also require changes in the configuration of the Vivado project (particularly if the Ethernet controller is in the PL), as well as the PetaLinux image. This alternative route is out of the scope of this tutorial, but we venture to point out that using the defaults where possible will reduce the amount of additional work required to set up a project, if a custom carrier card is to be used.

Furthermore, we note that when starting a Vivado project and selecting the default part to use, it can be confusing which option of the three to choose—the FPGA part alone, the UltraZed-7EV SOM or the UltraZed-7EV Carrier Card. In our projects, we have found that there is some crossover regarding which option works in which scenario, which is to say that the UltraZed-7EV SOM option will likely work for projects on the carrier card, the UltraZed-7EV Carrier Card option will likely work even if not on a carrier card, the FPGA part will likely work as well on and off the carrier card, etc. While it is hard to know all of what Vivado configures when a certain option is chosen, it is assumed that selecting the carrier card option will import the additional peripheral drivers that are present for use on the board such that the user can more easily interface with them. As long as these features are not needed, it shouldn't matter which option is chosen. For our configurations and using our custom carrier card, we have found it safe to use the UltraZed-7EV SOM option without issue (as of yet).

Acknowledgements

This tutorial is based heavily on Xilinx PetaLinux documentation (UG1144 and UG1157) and Xilinx Wiki posts (xilinx-wiki.atlassian.net/wiki/home). Various other resources were used including online sources, work done by various BYU students, and Xilinx employee contacts. Where helpful, these resources have been included as links in the tutorial.