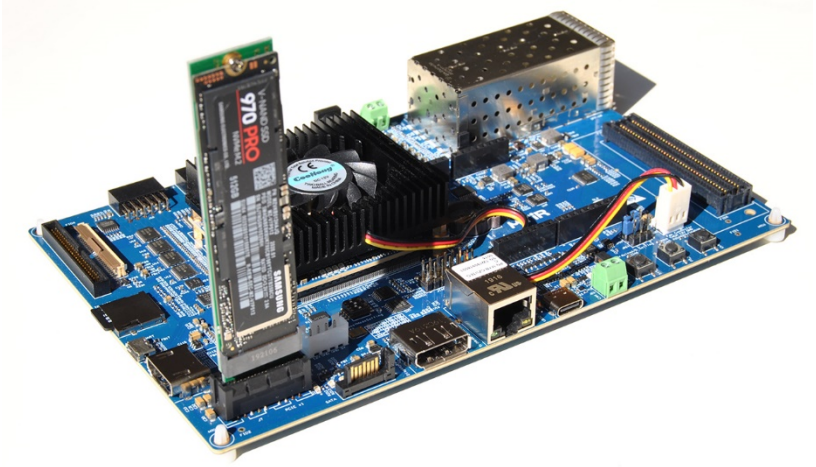
NVMe SSD for Ultramyir

The UltraMyir board by MYIR has a PCIe slot that can be used to connect an NVMe SSD drive. Here I will show how to configure the FPGA PS to use a Samsung 970 Pro as an SSD end point.

We need an adapter board to interface between the PCIe and the M.2 connector the SSD has. They are readily available, I got the one below on eBay for less than £4.

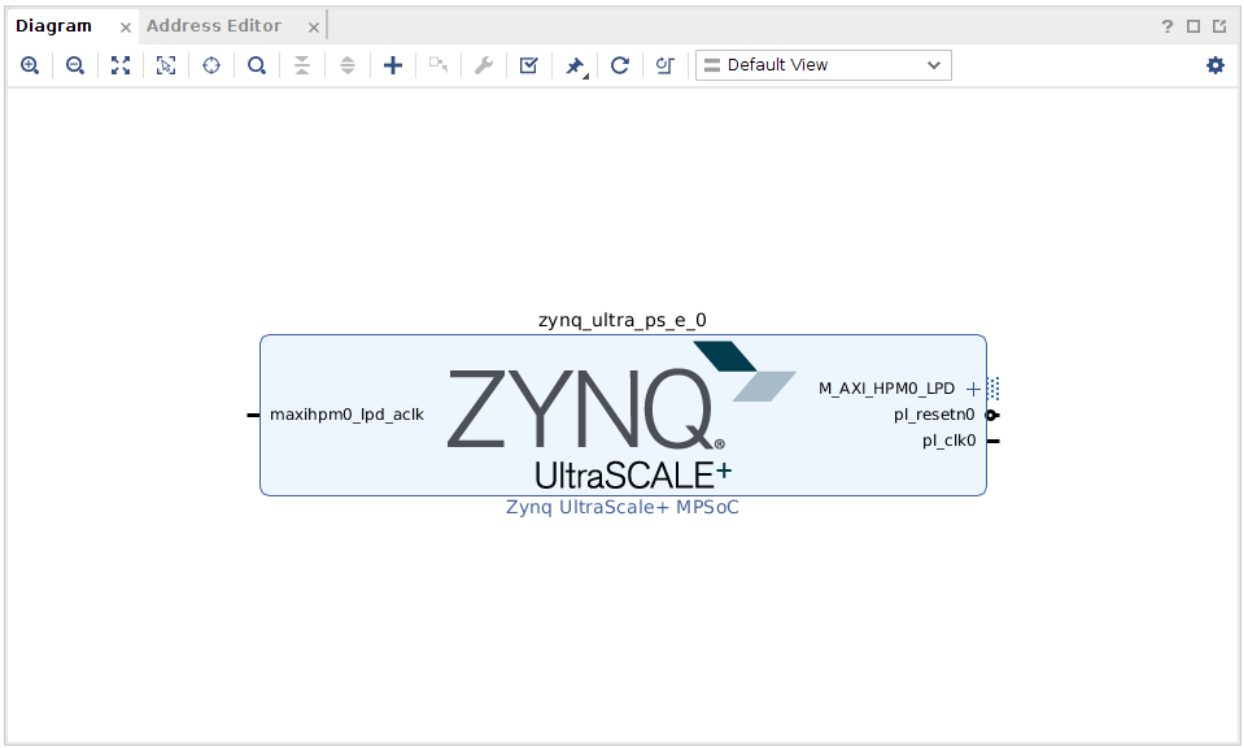
The SSD is easily assembled into the adapter and secured with the provided screw. Connector polarity prevents inserting in the wrong way:

Then it fits into the UltraMyir PCIe slot:

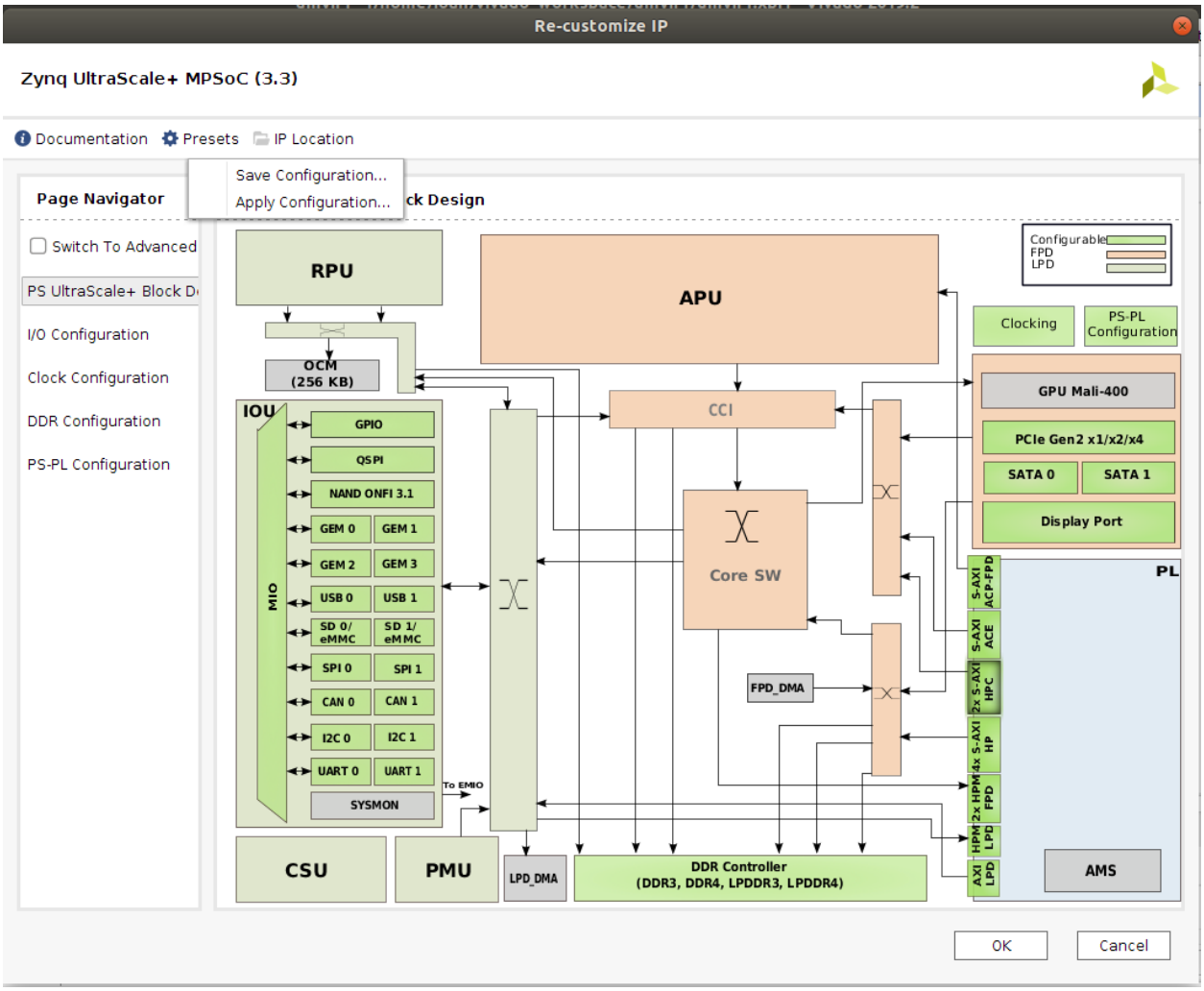


Hardware design

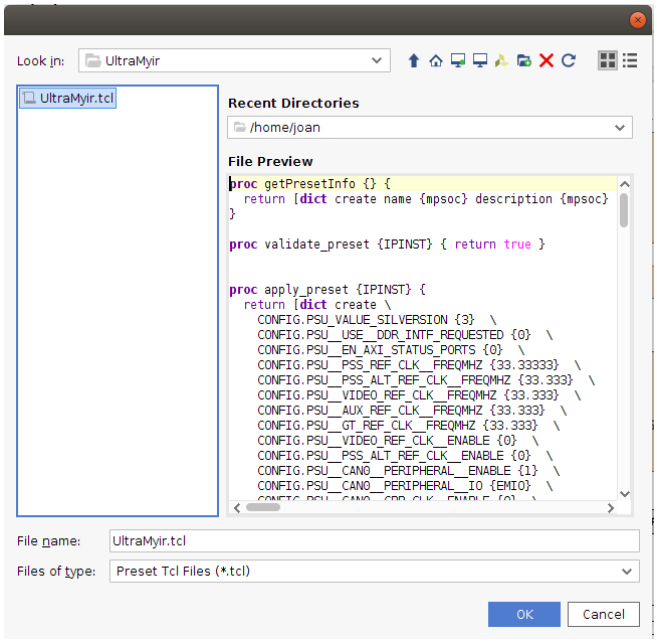
Open Vivado and create an RTL project. Configure it for the UltraMyir board or select its device (xczu3eg-sfvc784-1-e).
Create a block design and add to it the Zynq Ultrascale+ MPSoC IP:



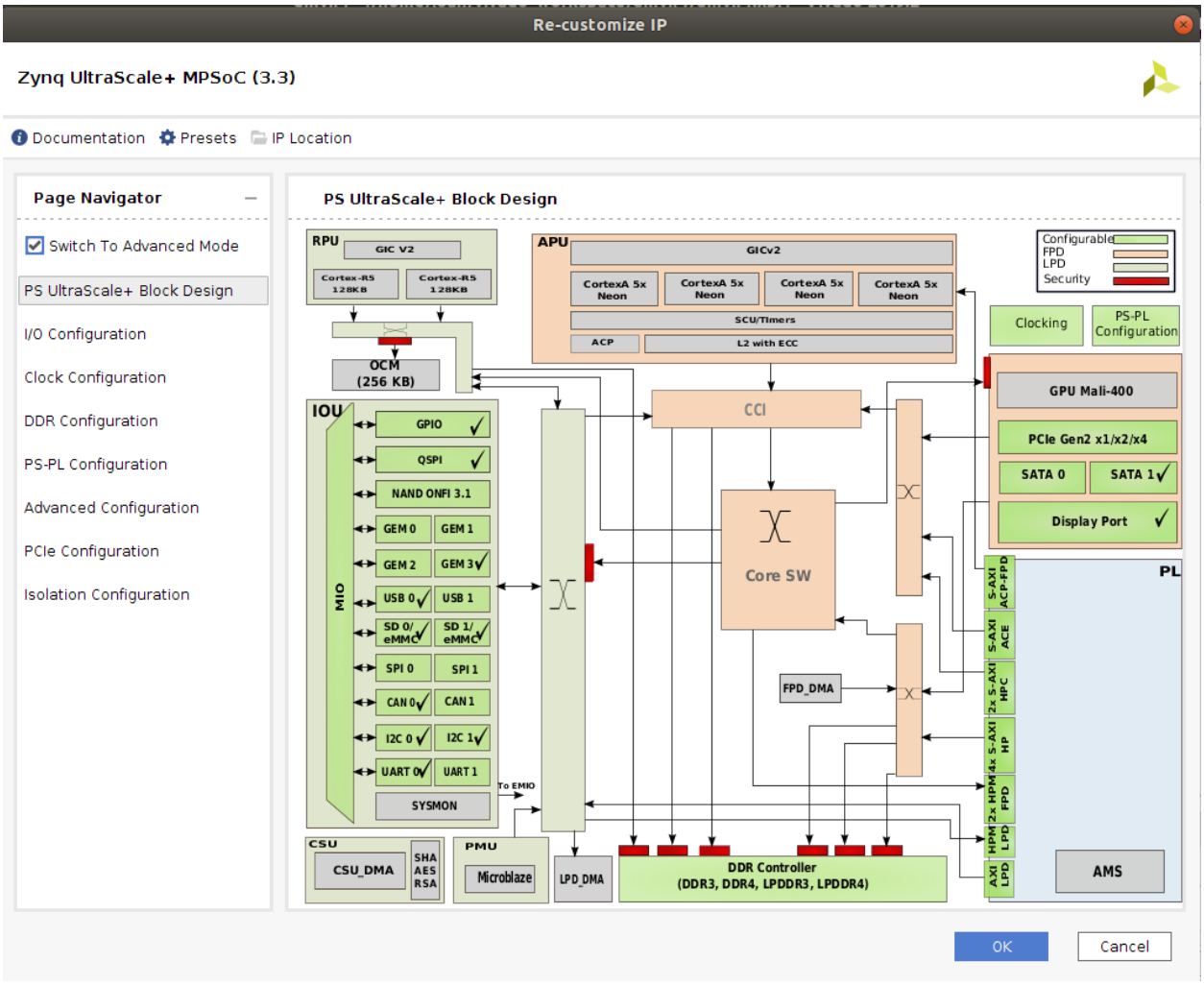
Double click on it to configure it. Click on the Presets button and select “Apply Configuration…”



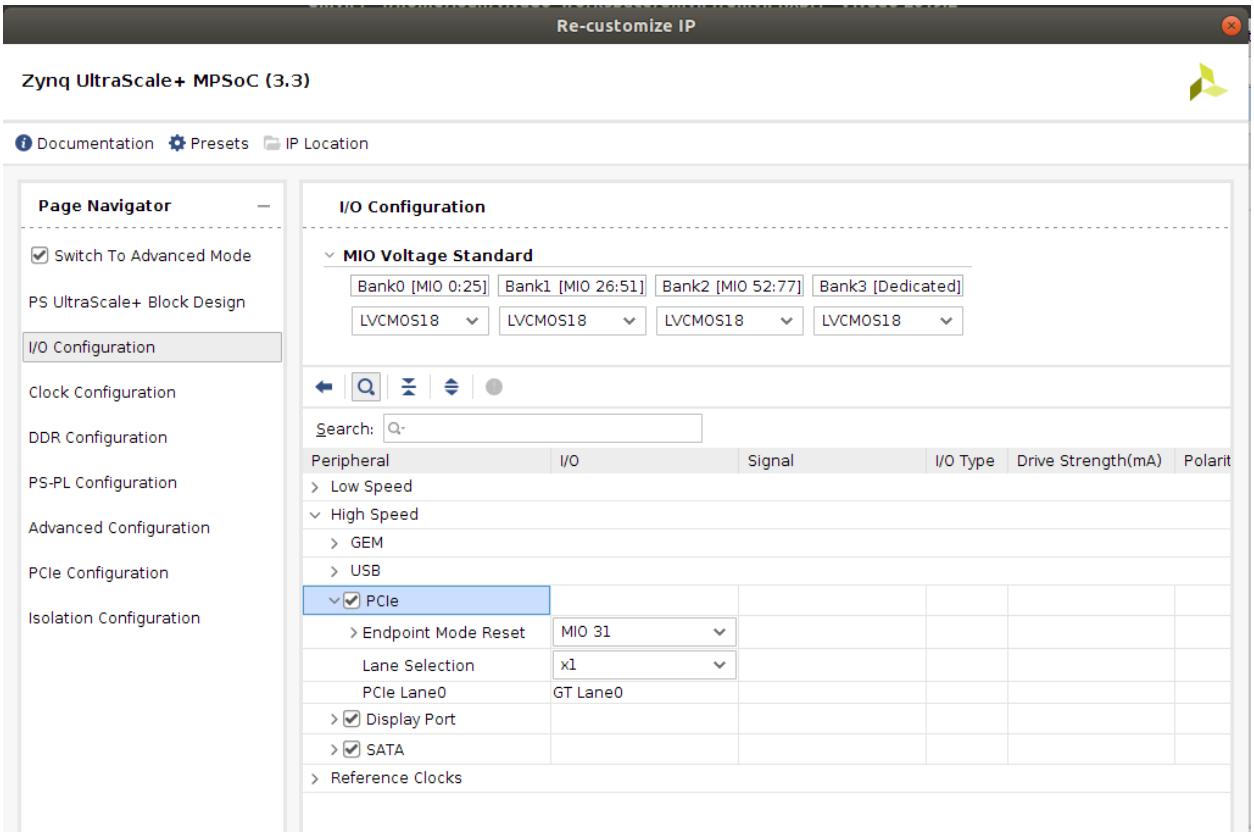
Browse for the tcl configuration file (Ultramyir_PS_config.tcl) and click OK.



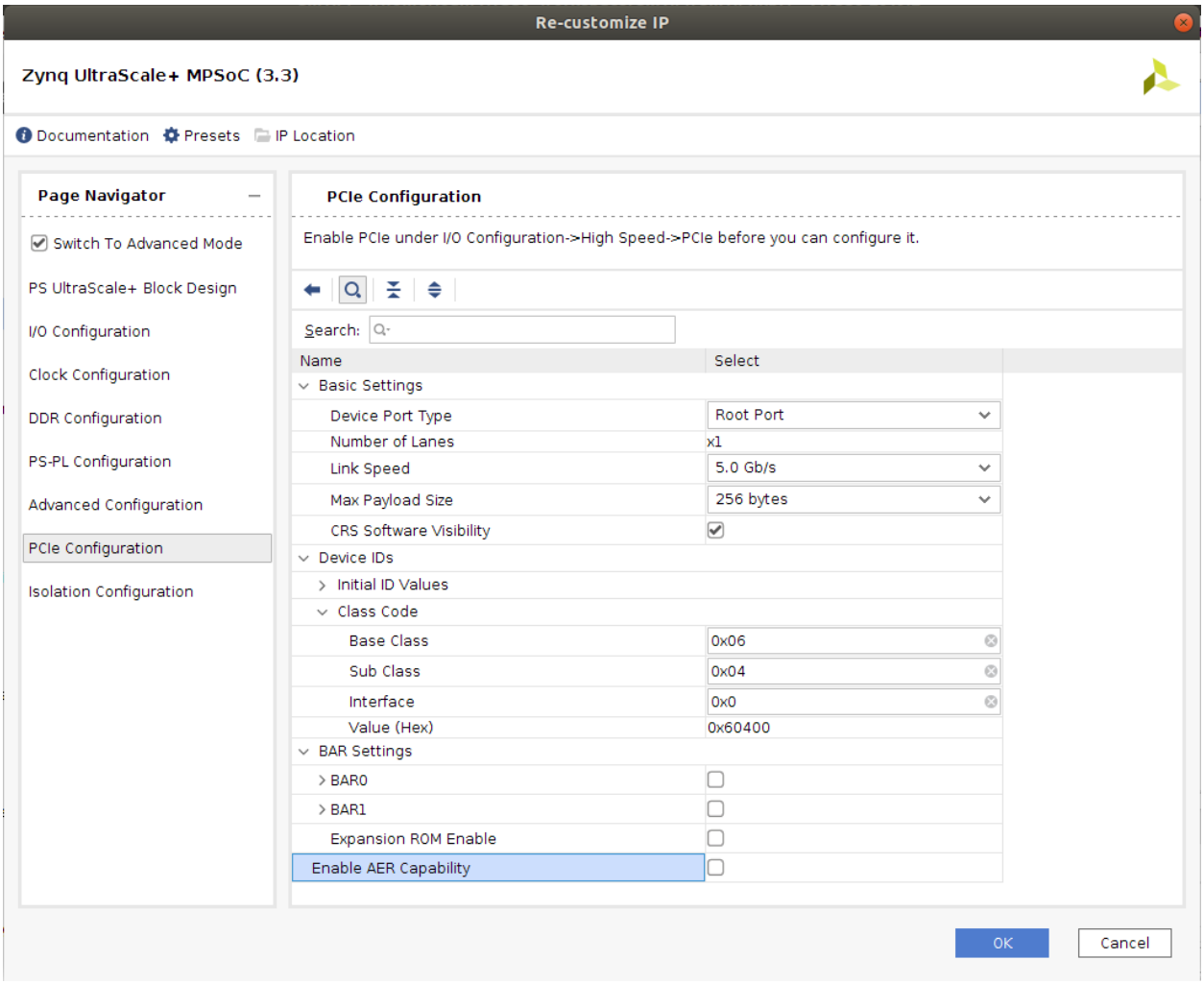
The basic configuration does not include PCIe so we have to do it by hand, first click on the ‘Switch to advanced’ box:



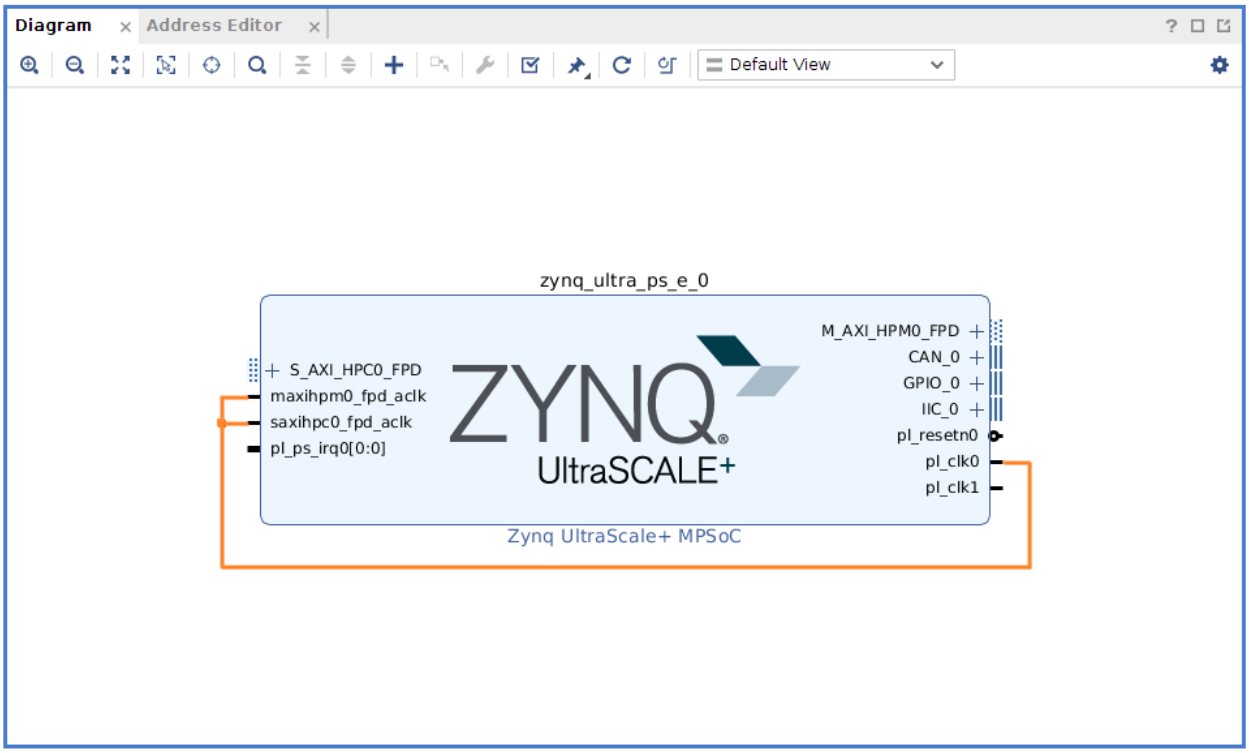
In the I/O configuration tab, activate PCIe and change the Reset pin to MIO31:



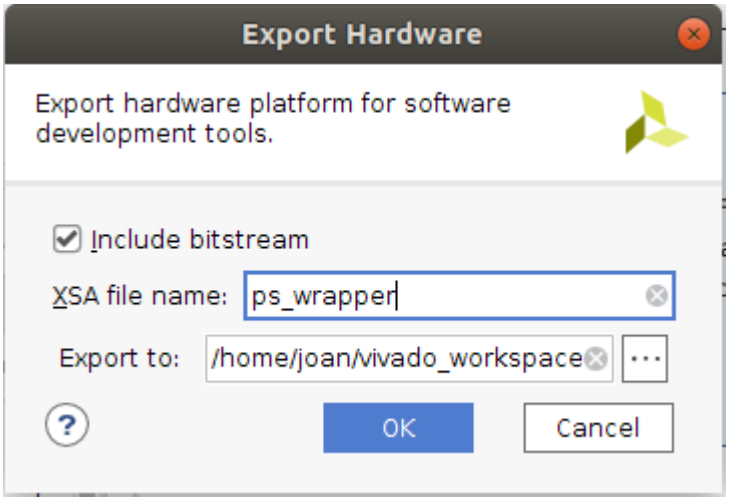
In the PCIe configuration tab, change the Port Type to Root, select one lane, check the CRS Software Visibility and set the Base Class and Sub Class to 0x06 and 0x04 respectively. Then click OK.



Finally, connect clk0 to the AXI clock inputs as below (AXI ports are not used and could as well being turned off).



Validate the design and create an HDL wrapper by right clicking on the block diagram in the Design Sources panel. Once this is done, generate the bitstream and export the hardware definition file (xsa) with File > Export > Export Hardware. Check the box 'Include bitstream':



Once the hardware definition and bitstream has been created we can close Vivado.

Petalinux creation

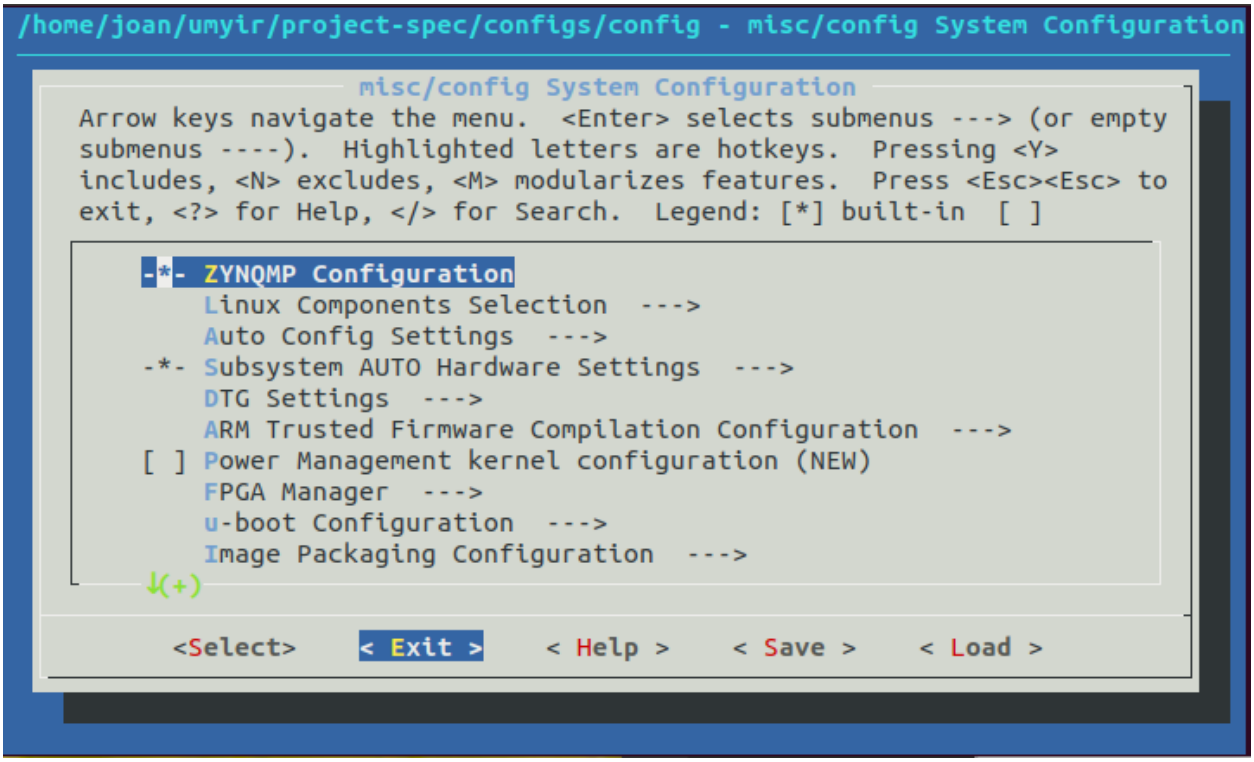
It is assumed here you have a Linux machine (I'm using Ubuntu 18.04 on a Virtual machine) with Petalinux tools installed (I have v2019.2). On a terminal in the root folder create a petalinux project:

```
joan@ubu18xilinx:~$ petalinux-create -t project -n umyir --template zynqMP
INFO: Create project: umyir
INFO: New project successfully created in /home/joan/umyir
joan@ubu18xilinx:~$
```

Change to the created directory and configure the project with the exported hardware. Here I previously copied the xsa file generated with Vivado to the root folder. Note that only one hardware description file must exist in the specified folder.

```
joan@ubu18xilinx:~$ cd umyir
joan@ubu18xilinx:~/umyir$ petalinux-config --get-hw-description ~/
INFO: Getting hardware description...
INFO: Rename ps_wrapper.xsa to system.xsa
[INFO] generating Kconfig for project
```

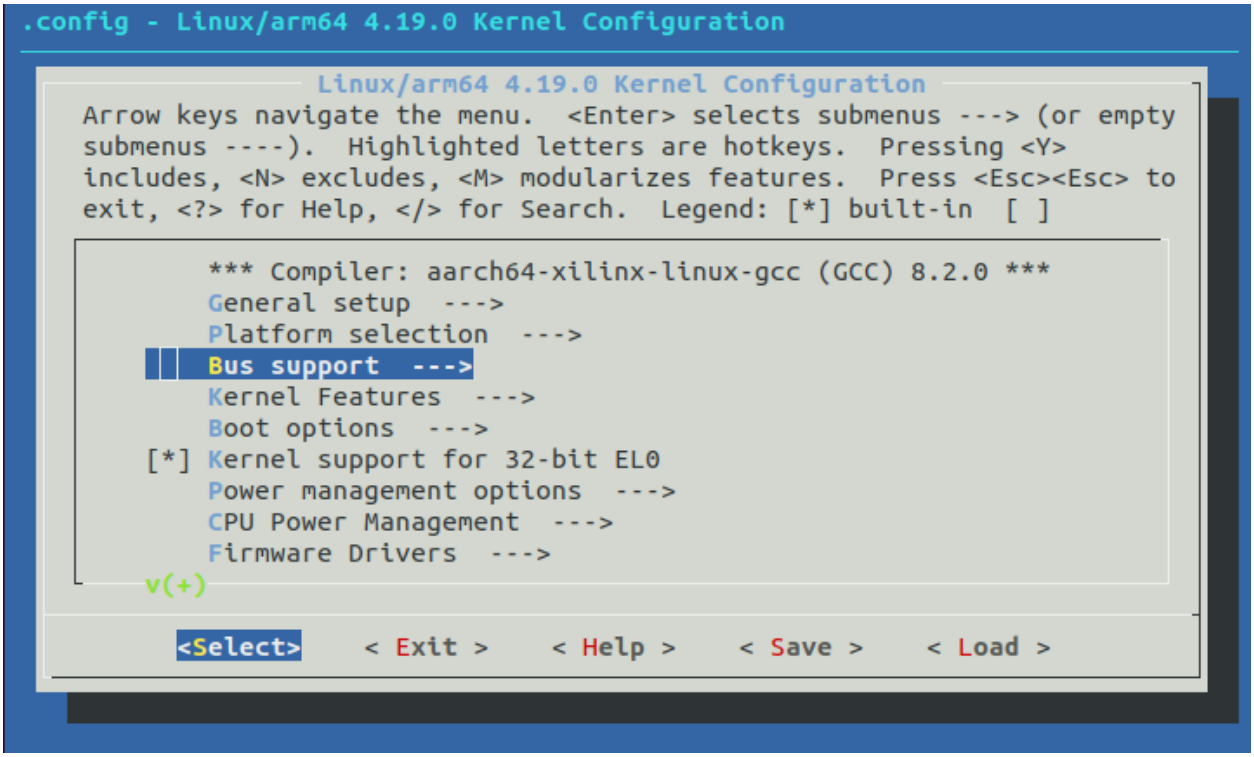
The base configuration does not need any changes so exit and save it:



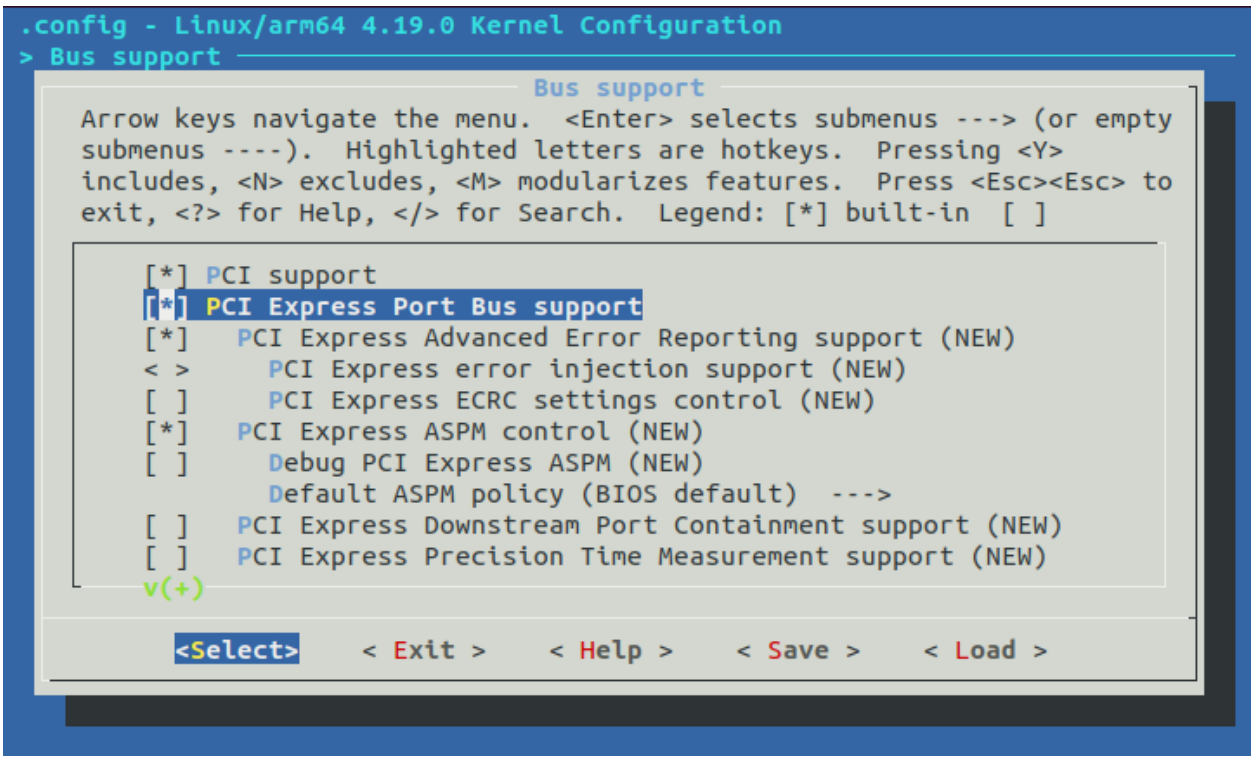
The kernel needs some configuration. run the following command:

```
joan@ubu18xilinx:~/umyir$ petalinux-config -c kernel
[INFO] generating Kconfig for project
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
```

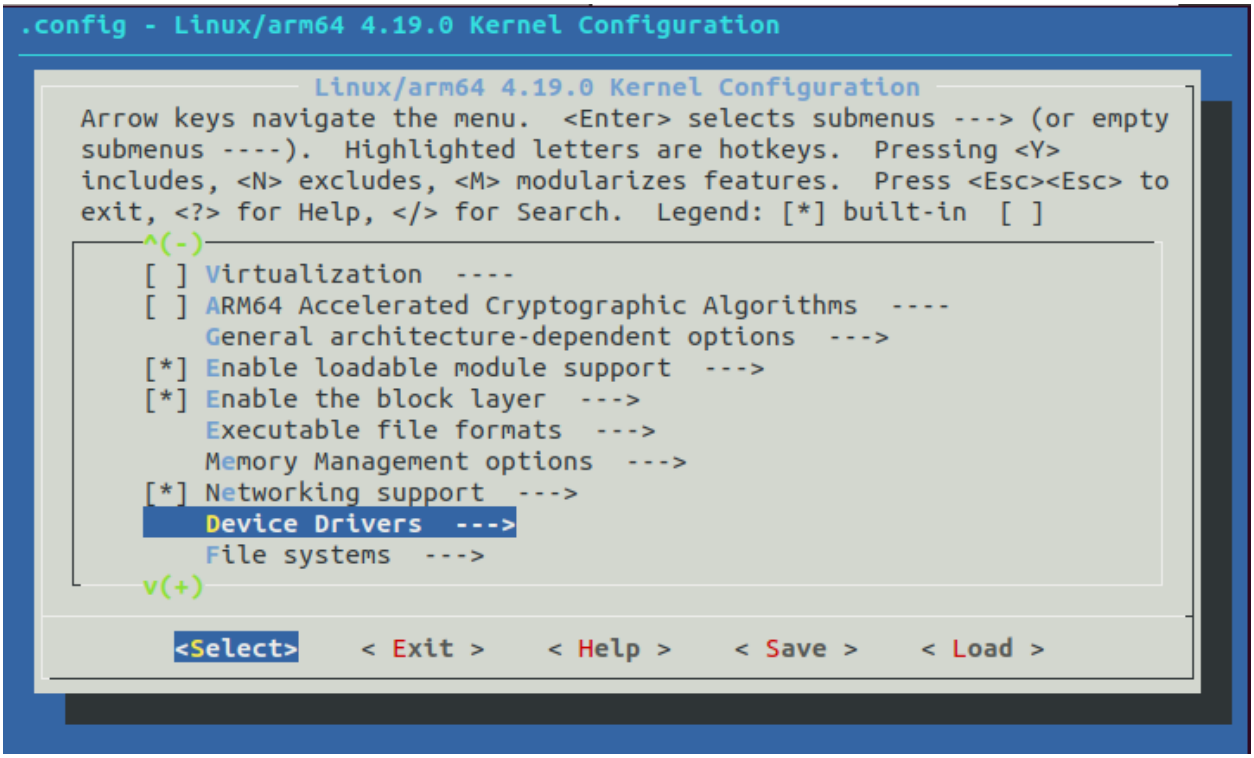
Browse to Bus Support and enter:



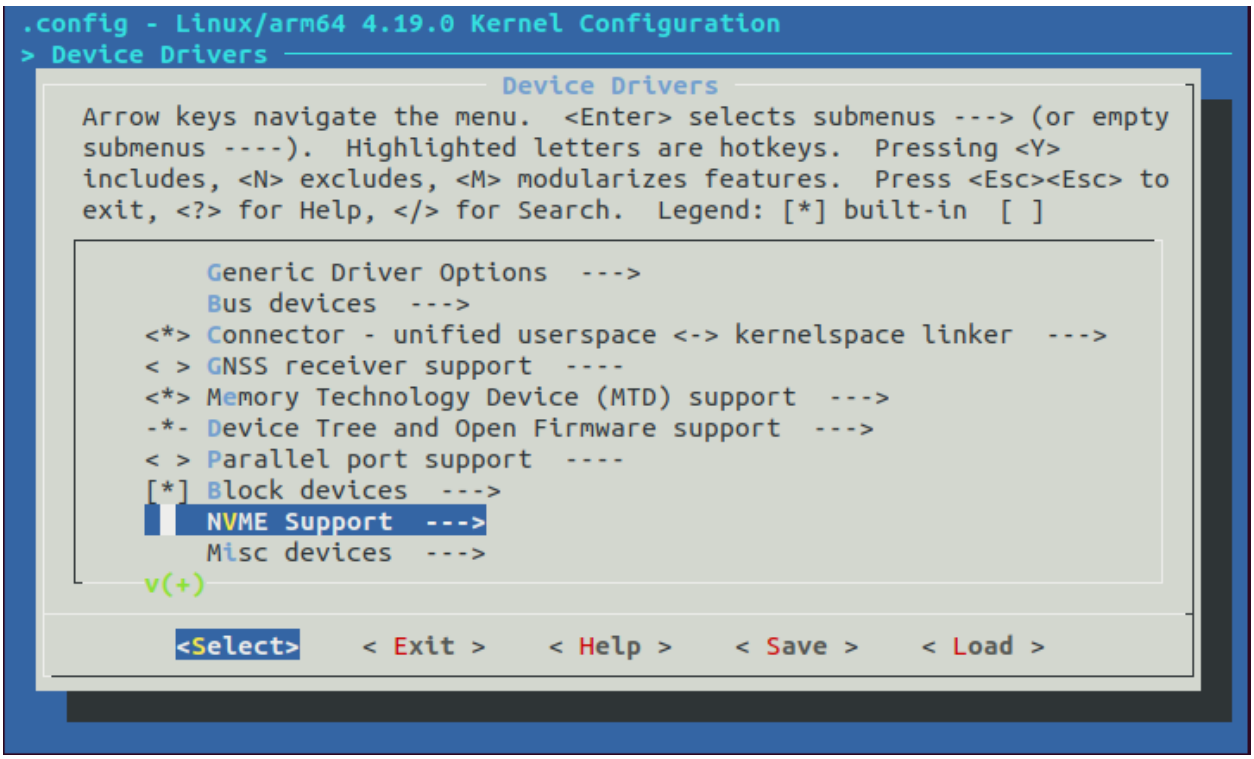
Here activate the PCI Express Port Bus Support option and leave others as they are.



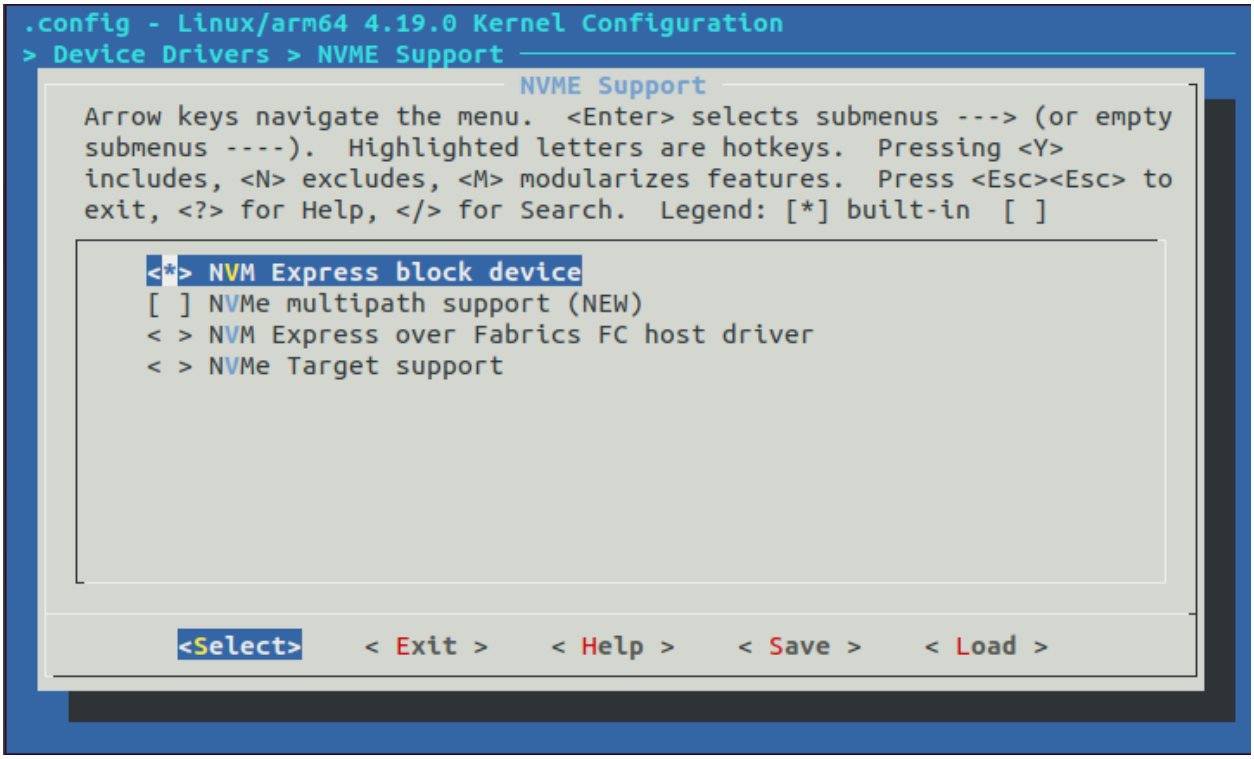
Go one level back, browse to Device Drivers and enter:



Browse for NVMe Support and enter:



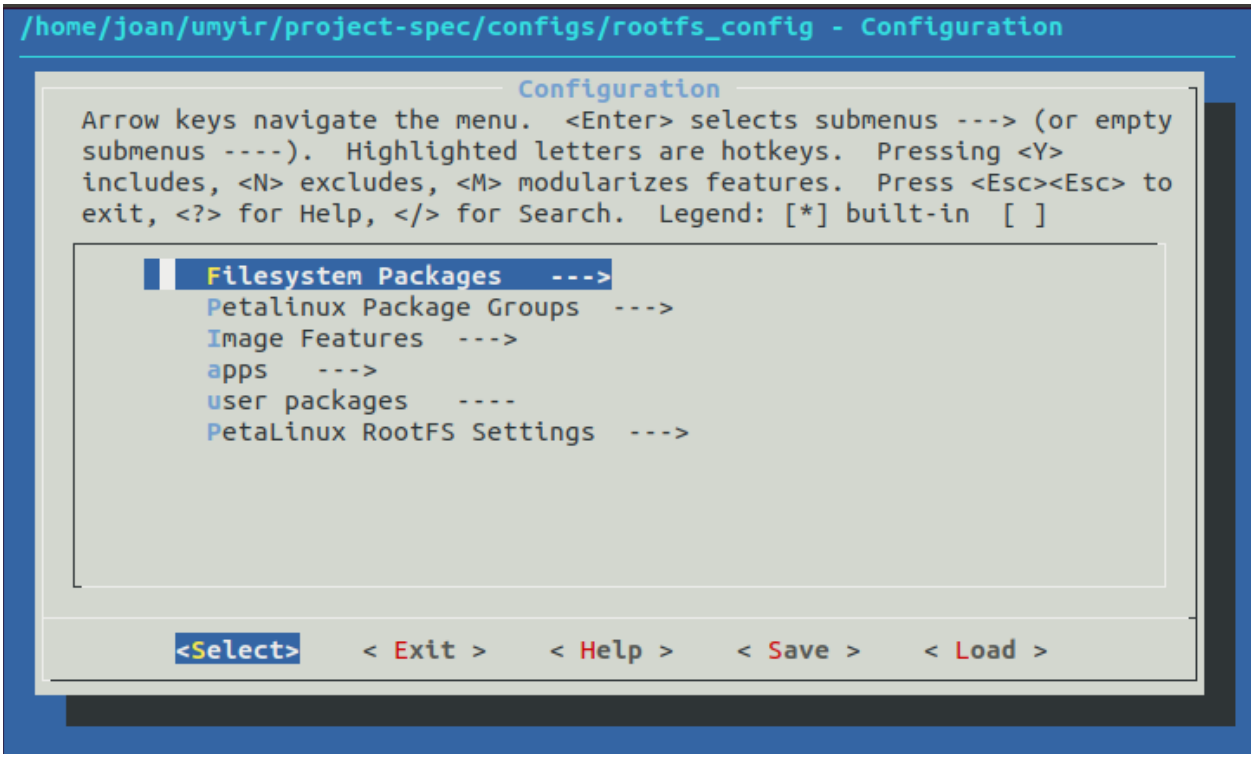
Here activate NVMe Express Block Device:



Save and exit, then configure the root file system with the following command:

```
joan@ubu18xilinx:~/umyir$ petalinux-config -c rootfs
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
```

Enter Filesystem packages



Inside base > util-linux, check util-linux, util-umount, util-mount, util-mkfs, util-fdisk
Inside base > e2fsprogs, check e2fsprogs-mke2fs
Inside console > utils, check libpci, pciutils

Save and exit.

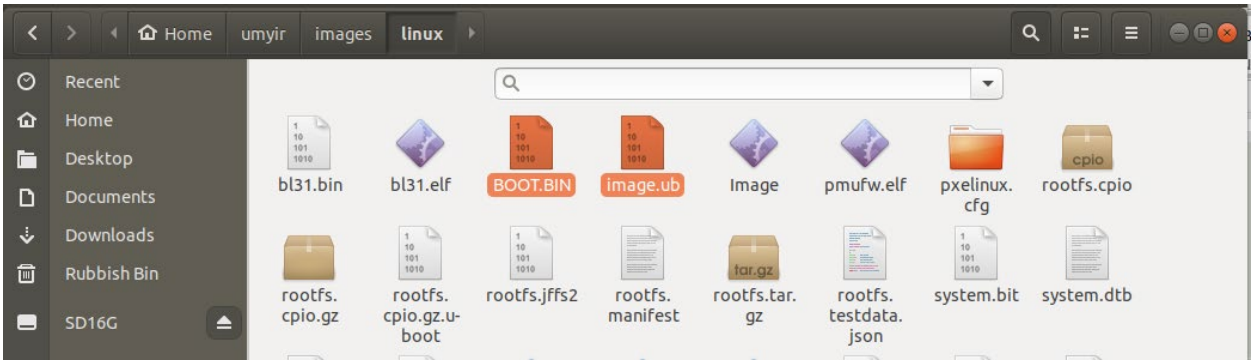
Build with this command:

```
joan@ubu18xilinx:~/umyir$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
[INFO] generating user layers
```

Last, package to boot from SD card:

```
joan@ubu18xilinx:~/umyir$ petalinux-package --boot --force --fsbl ./images/linux
/zynqmp_fsbl.elf --fpga ./images/linux/system.bit --u-boot ./images/linux/u-boot
.elf --pmufw ./images/linux/pmufw.elf
```

The files boot.bin and image.ub should be in the images/linux folder inside the project folder. Insert an SD card and copy these two files to the card (FAT formatted)



Extract the SD card, insert it into the ultramyir board, set the boot switches to boot from SD card, connect a USB cable to use the serial console and power the board:

The boot log should show the PCI:

```
[    3.725145] of-fpga-region fpga-full: FPGA Region probed
[    3.734149] nwl-pcie fd0e0000.pcie: Link is UP
[    3.738624] nwl-pcie fd0e0000.pcie: host bridge /amba/pcie@fd0e0000 ranges:
[    3.745591] nwl-pcie fd0e0000.pcie:    MEM 0xe0000000..0xefffffff -> 0xe0000000
[    3.752812] nwl-pcie fd0e0000.pcie:    MEM 0x600000000..0x7fffffff -> 0x600000000
[    3.760391] nwl-pcie fd0e0000.pcie: PCI host bridge to bus 0000:00
[    3.766568] pci_bus 0000:00: root bus resource [bus 00-ff]
[    3.772050] pci_bus 0000:00: root bus resource [mem 0xe0000000-0xefffffff]
[    3.778916] pci_bus 0000:00: root bus resource [mem 0x600000000-0x7fffffff pref]
[    3.787817] pci 0000:01:00.0: 4.000 Gb/s available PCIe bandwidth, limited by 5 GT/s x1 link
 of 31.504 Gb/s with 8 GT/s x4 link)
[    3.802577] pci 0000:00:00.0: BAR 8: assigned [mem 0xe0000000-0xe00fffff]
[    3.809363] pci 0000:01:00.0: BAR 0: assigned [mem 0xe0000000-0xe0003fff 64bit]
[    3.816678] pci 0000:00:00.0: PCI bridge to [bus 01-0c]
[    3.821902] pci 0000:00:00.0:   bridge window [mem 0xe0000000-0xe00fffff]
[    3.828759] pcieport 0000:00:00.0: enabling device (0000 -> 0002)
[    3.835061] nvme nvme0: pci function 0000:01:00.0
[    3.839781] nvme 0000:01:00.0: enabling device (0000 -> 0002)
[    3.840266] xilinx-dpdma fd4c0000.dma: Xilinx DPDMA engine is probed
[    3.852097] xilinx-zynqmp-dma fd500000.dma: ZynqMP DMA driver Probe success
```

And lspic command should be like:

```
root@umyir:~# lspci -v
00:00.0 PCI bridge: Xilinx Corporation Device d011 (prog-if 00 [Normal decode])
        Flags: bus master, fast devsel, latency 0, IRQ 54
        Bus: primary=00, secondary=01, subordinate=0c, sec-latency=0
        I/O behind bridge: 00000000-00000fff [size=4K]
        Memory behind bridge: e0000000-e00fffff [size=1M]
        Prefetchable memory behind bridge: None
        Capabilities: [40] Power Management version 3
        Capabilities: [60] Express Root Port (Slot-), MSI 00
        Capabilities: [100] Device Serial Number 00-00-00-00-00-00-00-00
        Capabilities: [10c] Virtual Channel
        Capabilities: [128] Vendor Specific Information: ID=1234 Rev=1 Len=018 <?>
        Kernel driver in use: pcieport

01:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd NVMe SSD Controller SM981/PM9
81 (prog-if 02 [NVM Express])
        Subsystem: Samsung Electronics Co Ltd Device a801
        Flags: bus master, fast devsel, latency 0, IRQ 55
        Memory at e0000000 (64-bit, non-prefetchable) [size=16K]
        Capabilities: [40] Power Management version 3
        Capabilities: [50] MSI: Enable+ Count=1/1 Maskable- 64bit+
        Capabilities: [70] Express Endpoint, MSI 00
        Capabilities: [b0] MSI-X: Enable- Count=33 Masked-
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [148] Device Serial Number 00-00-00-00-00-00-00-00
        Capabilities: [158] Power Budgeting <?>
        Capabilities: [168] Secondary PCI Express <?>
        Capabilities: [188] Latency Tolerance Reporting
        Capabilities: [190] L1 PM Substates
        Kernel driver in use: nvme
```

Check it appears in /dev/

```
root@umyir:~# ls /dev/nv* -l
crw-------    1 root     root      245,   0 Apr 25 19:17 /dev/nvme0
brw-rw----    1 root     disk      259,   0 Apr 25 19:17 /dev/nvme0n1
root@umyir:~#
root@umyir:~#
```

Let's create and format a partition with fdisk:

```
Disk /dev/nvme0n1: 477 GiB, 512110190592 bytes, 1000215216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x5001e5ed

Device         Boot Start        End    Sectors  Size Id Type
/dev/nvme0n1p1       2048 1000215215 1000213168  477G 83 Linux
```

And create a filesystem with mkfs.ext4:

```
root@umyir:~# mkfs.ext4 /dev/nvme0n1p1
mke2fs 1.44.3 (10-July-2018)
64-bit filesystem support is not enabled.  The larger fields afforded by this feature enable ful
l-strength checksumming.  Pass -O 64bit to rectify.
Discarding device blocks: done
Creating filesystem with 125026646 4k blocks and 31260672 inodes
Filesystem UUID: 67c7e7b7-5117-42b7-b583-db6ea50d2eac
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
        102400000

Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

Reboot. The disk should auto-mount:

```
root@umyir:~# mount
rootfs on / type rootfs (rw,size=1887164k,nr_inodes=471791)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
configfs on /sys/kernel/config type configfs (rw,relatime)
devtmpfs on /dev type devtmpfs (rw,relatime,size=1887164k,nr_inodes=471791,mode=755)
tmpfs on /run type tmpfs (rw,nosuid,nodev,mode=755)
tmpfs on /var/volatile type tmpfs (rw,relatime)
/dev/mmcblk0p1 on /run/media/mmcblk0p1 type ext4 (rw,relatime)
/dev/nvme0n1p1 on /run/media/nvme0n1p1 type ext4 (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620,ptmxmode=000)
root@umyir:~#
```