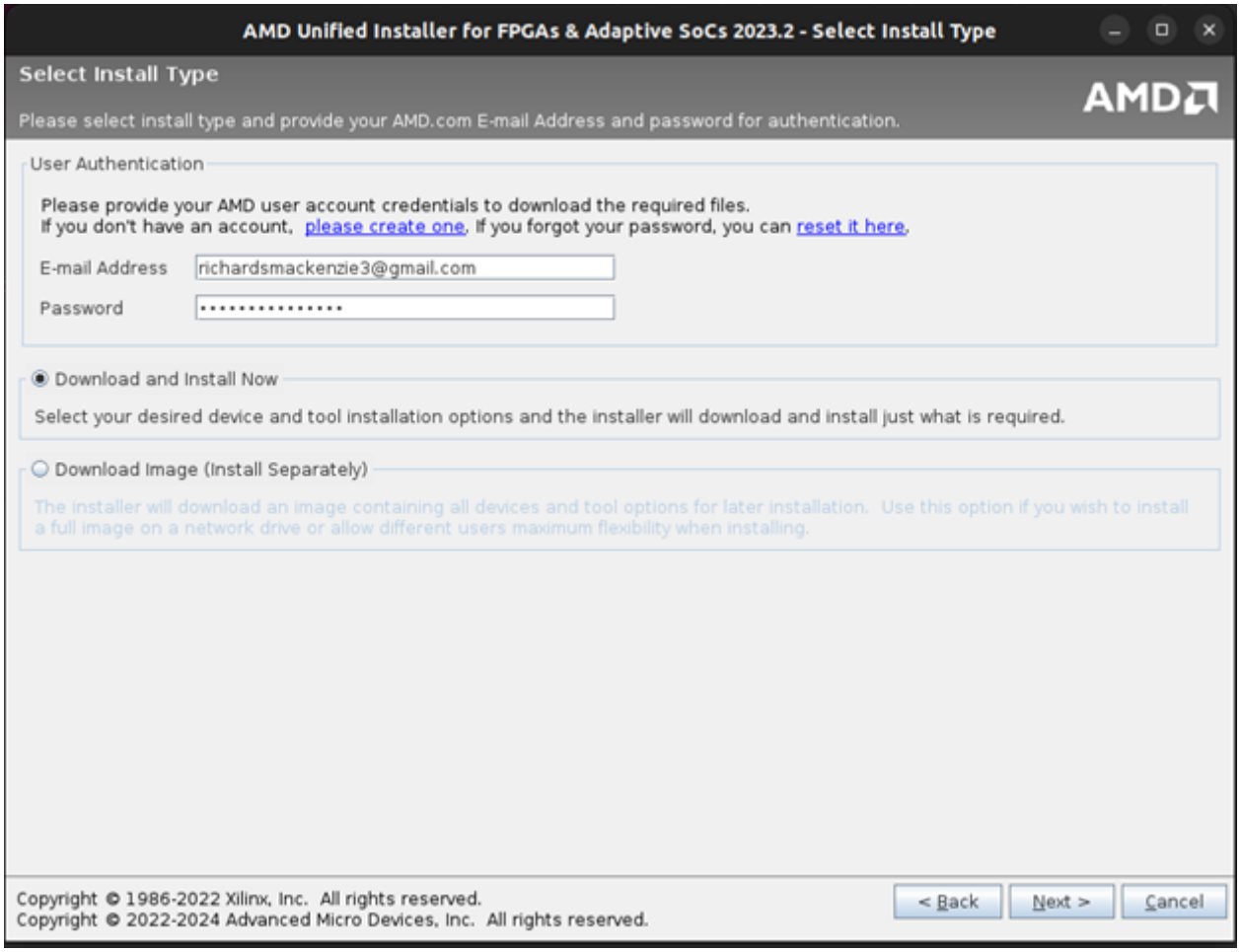
Setting up RPi pins and PMODs on KR260

Note:

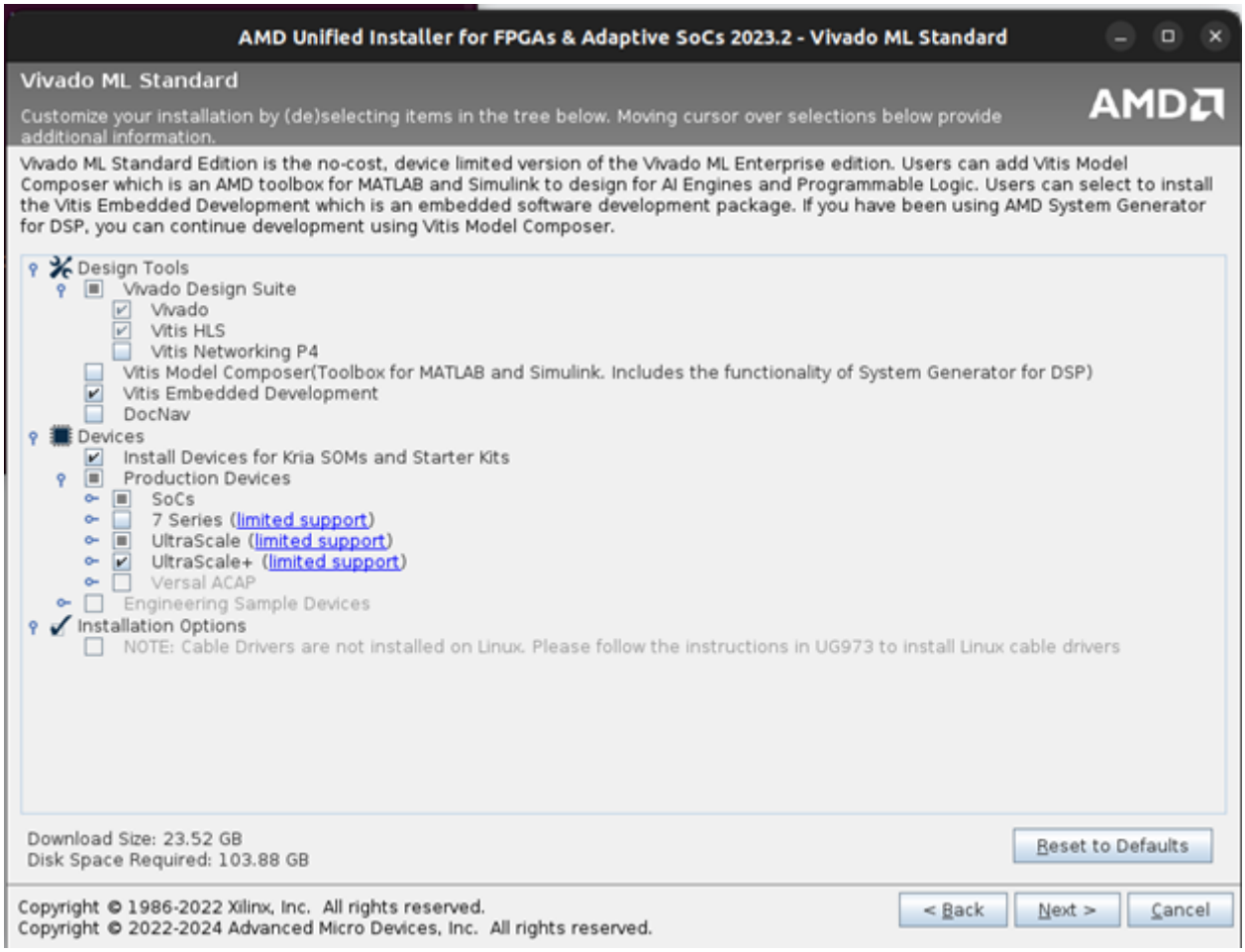
- Project name: kr260_gpio
- Block design: kr260_bd

Install Software Needed

1. Install vivado
    - Ensure you have ~105GB free
    - Get Self Extracting Web Installer from here. Can be used on windows or linux
    - Create a Xilinx account (it's free)
    - Run the installer and Login



    - Choose Vivado then Vivado ML Standard
    - Choose the following options (bare minimum options, you can add more if you like)



- Let it install. It will take a while

2. Clone and checkout DTG:

```
git clone https://github.com/Xilinx/device-tree-xlnx
cd device-tree-xlnx
git checkout xilinx_rel_v2023.2
```

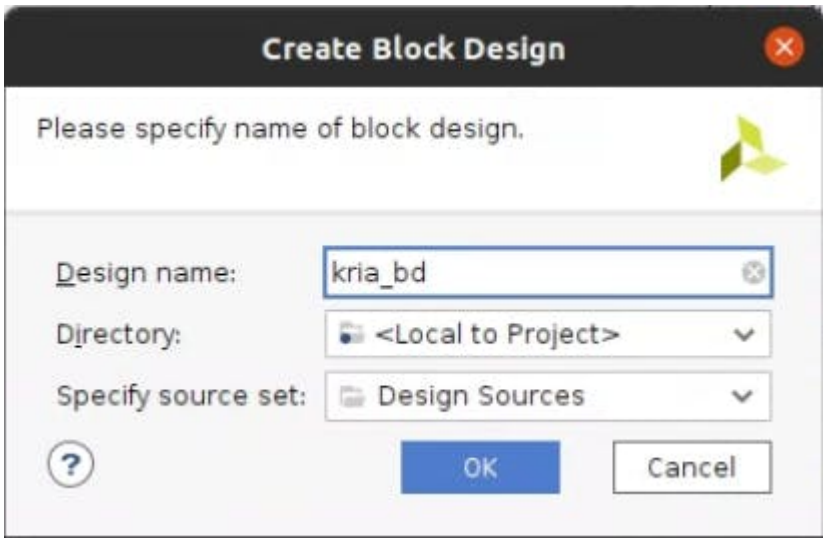
3. Clone DTC:

```
git clone https://git.kernel.org/pub/scm/utils/dtc/dtc.git
cd dtc
make
export PATH=$PATH:/dtc
```

Creating FPGA Design (Created on computer not KR260)

1. Open Vivado and create a new project named Kria_KR260

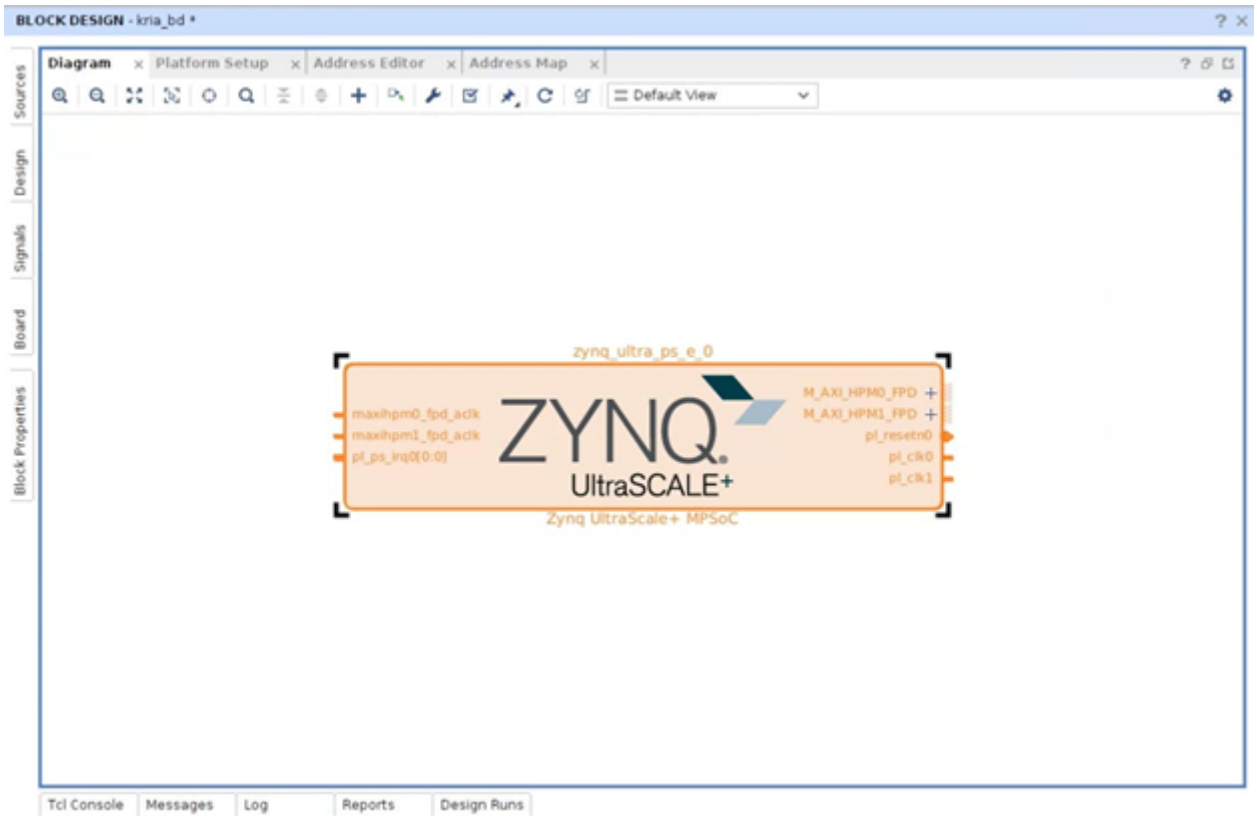
   - In the **Project Type** window, ensure the option **Do not specify sources at this time** is checked.
   - Move to the **Part** page and navigate to the **Boards** tab. Click on Refresh, then search for KR260. Once located, single-click on the Kria KR260 Robotics Starter Kit row and proceed by clicking **Next** then **Finish**.



- A new Diagram tab will open with an empty block design.

2. Click the + and add the **Zynq UltraScale+ MPSoC** Processor

- Upon addition, a green banner will appear at the top of the **Diagram** window, providing the option to **Run Block Automation**. This automation will apply specific KR260 board presets to the Zynq MPSoC IP block.
- Click the **Run Block Automation** link and ensure that the **Apply Board Preset** option is selected in the window that appears. Then, click OK.

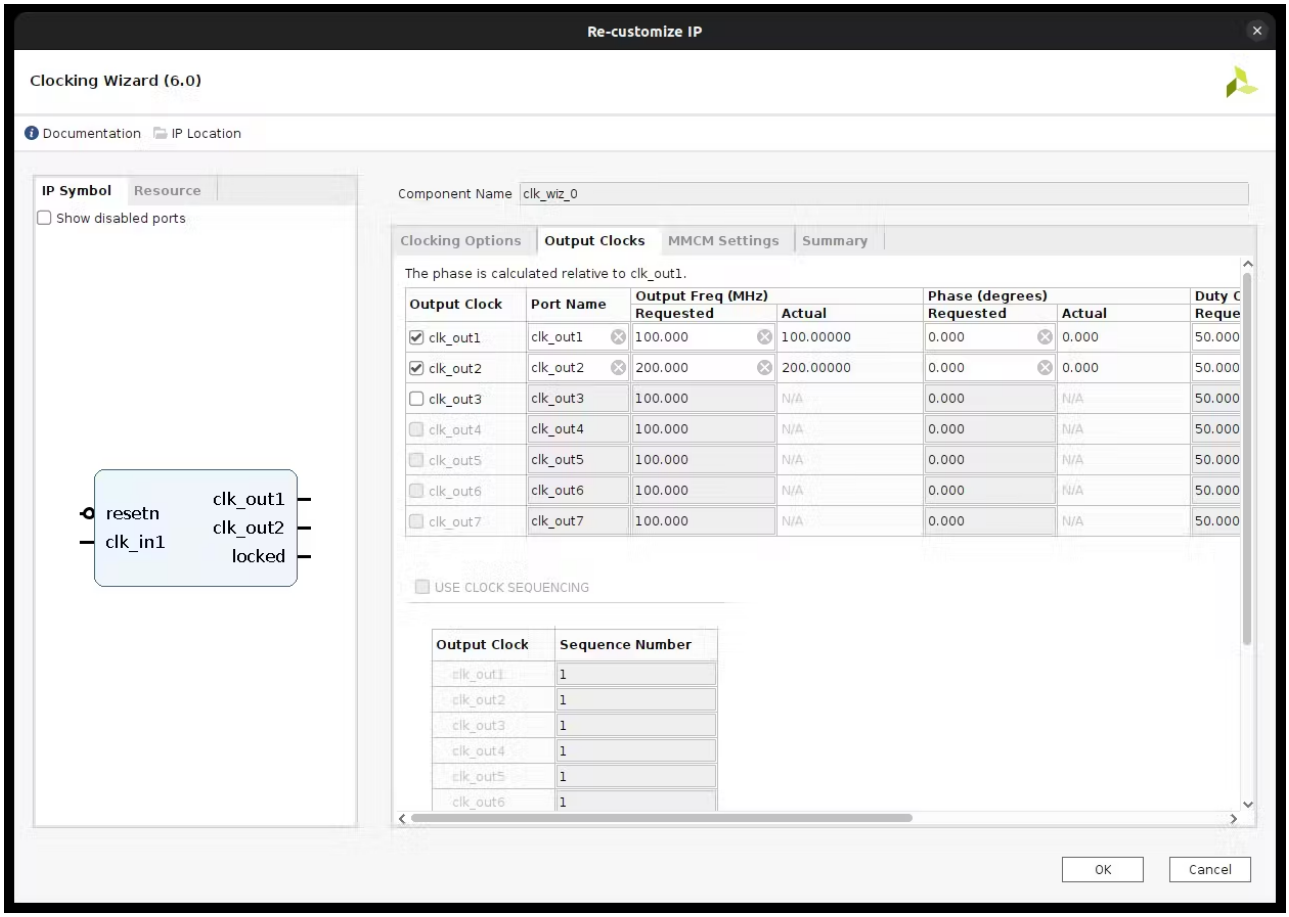


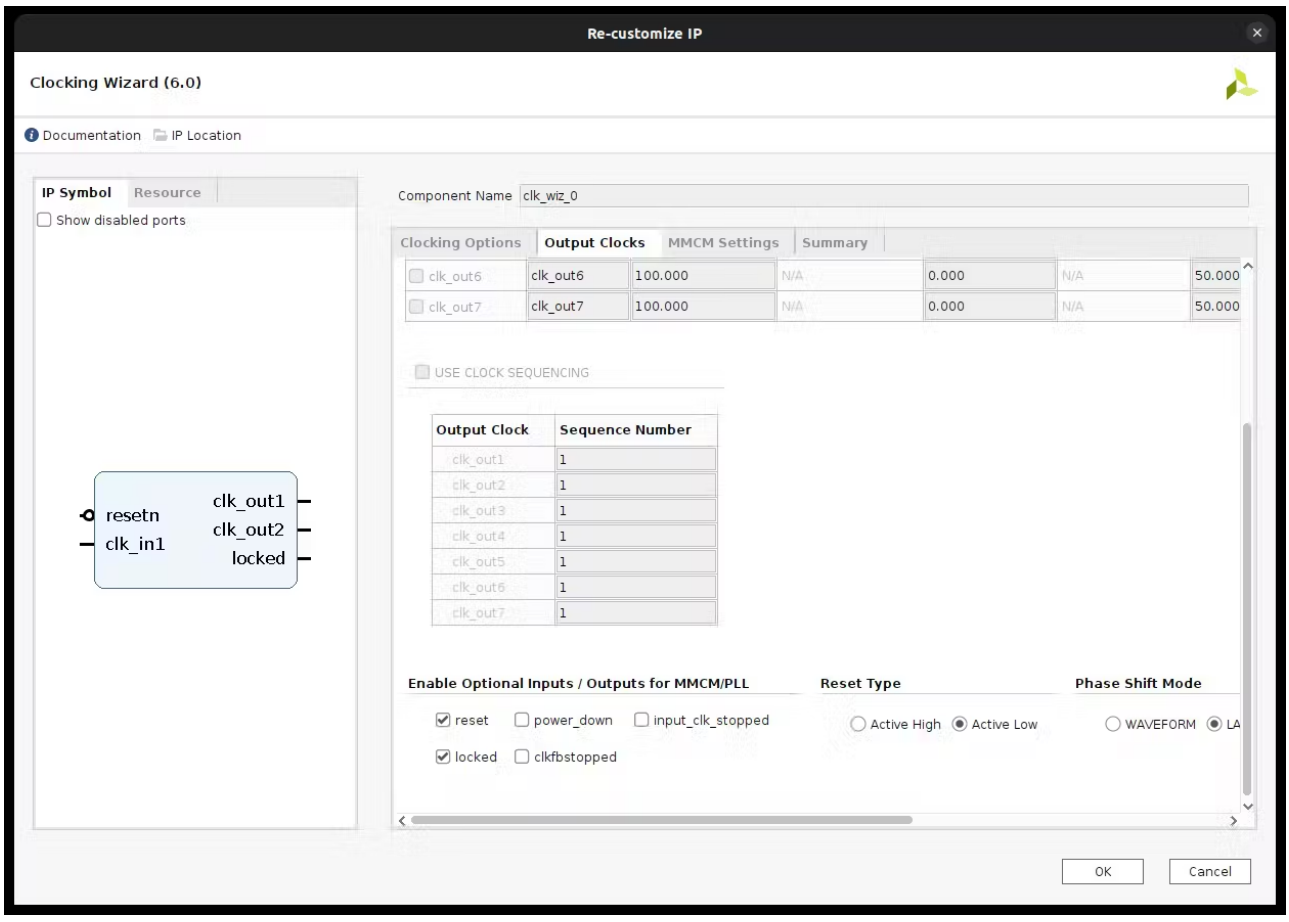
- Double-click on the **Zynq UltraScale+ MPSoC** to access the configuration window. Go to **PS-PL Configuration** -> General -> PS-PL Interfaces -> Master interface
- Disable the two full-power high-performance AXI ports within the block design.
- Enable the low-power high-performance AXI port.

3. Click the + and add a **Clocking Wizard**

- Double-click on the clocking wizard IP to access its customization options.

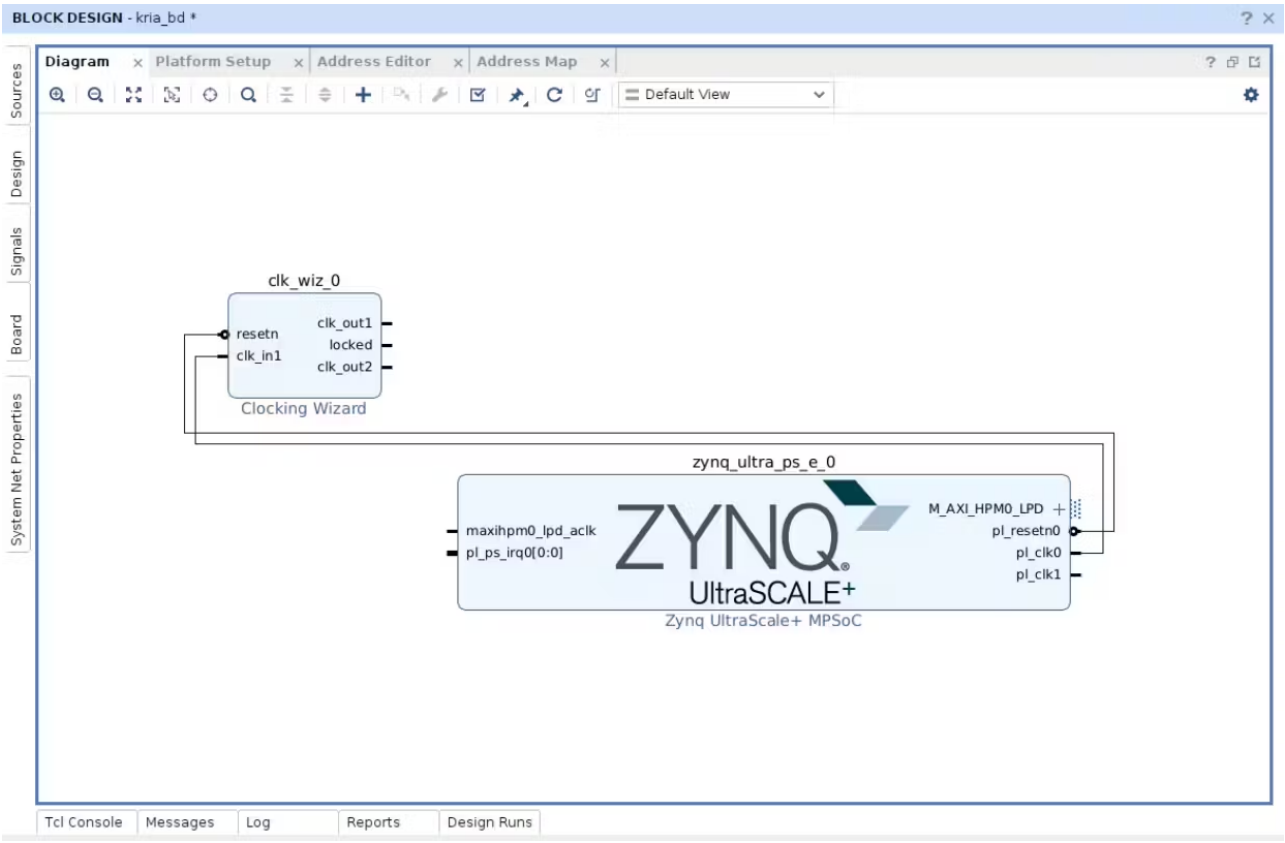
- Within the **Output Clocks** tab, enable the `clk_out1` and `clk_out2`, setting the **Requested Output Frequency** to `100 MHz` and `200MHz` respectively. Additionally, scroll down and set **Reset Type** to `Active Low`.





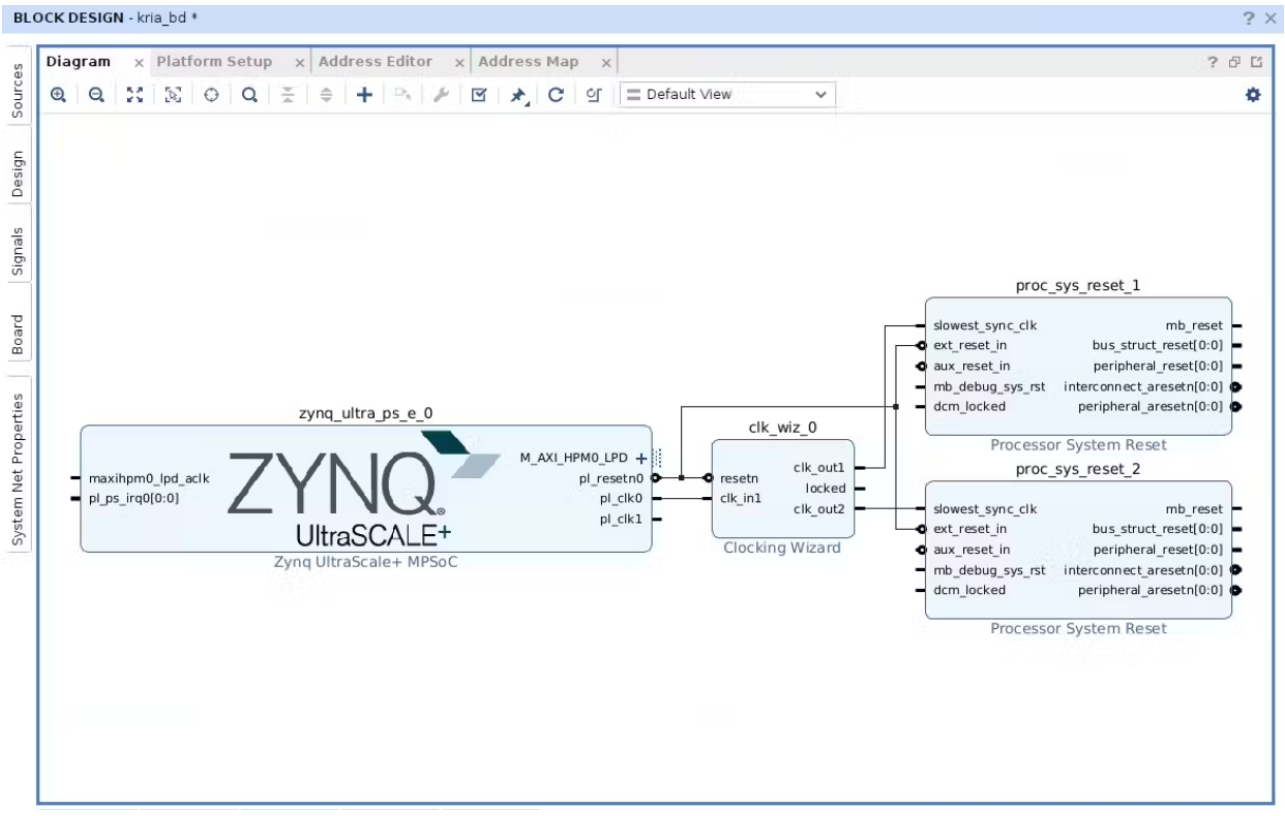
- Connect the `clk_in1` input of the clocking wizard IP to the `pl_clk0` output from the processor.

- Connect the `resetn` input of the clocking wizard IP to the `pl_reset0` output from the processor.



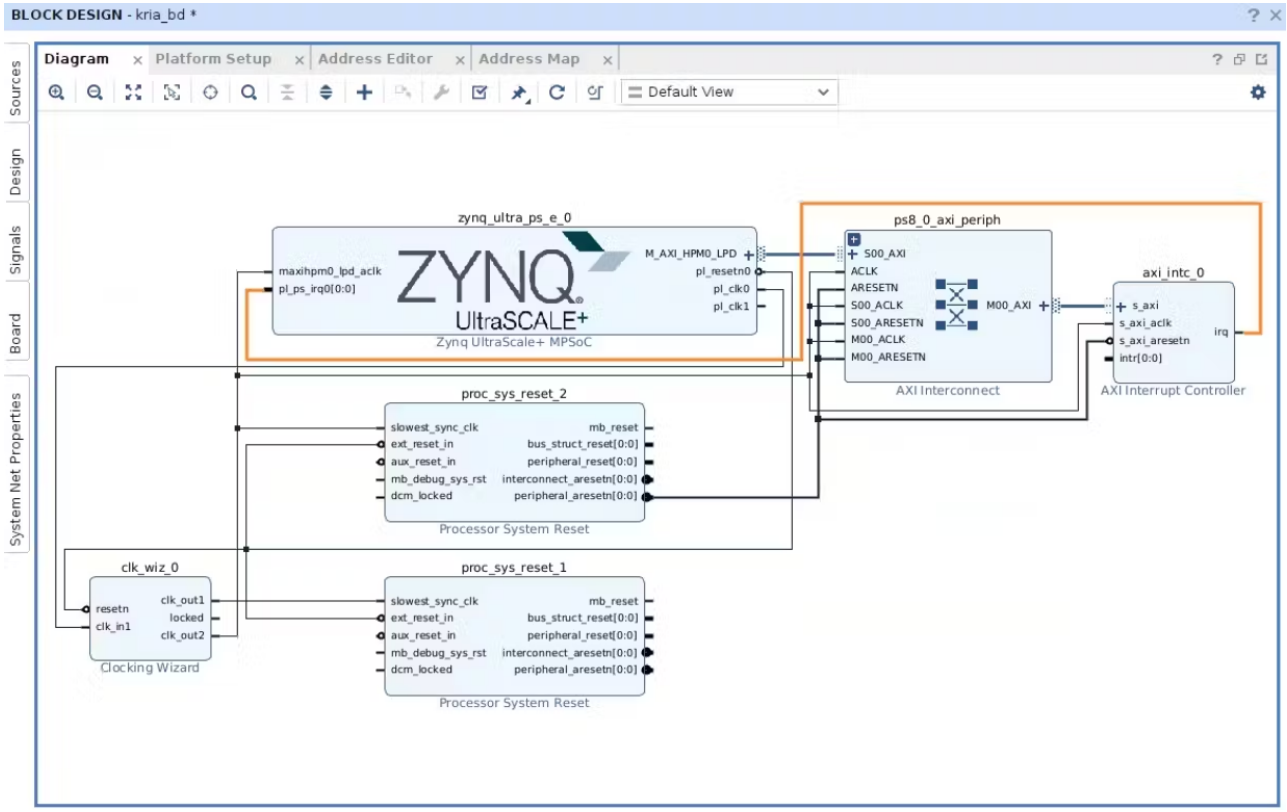
4. Add 2 **Processor System Reset** blocks

- Connect the `ext_reset_in` of each **Processor System Reset** IP to the `pl_resetn0` **output of the Zynq UltraScale+** IP.
- Connect each `clk_out` output of the clocking wizard IP to the `slowest_sync_clk` of each respective **Processor System Reset** IP.



5. Add an **AXI Interrupt Controller**

- Whenever external IO's on the board need the processors attention they send an interrupt signal to the processor. The **AXI Interrupt Controller** consolidates all these signals before sending them to the processor
- Double-click on the block and modify the **Interrupt Output Connection** from `Bus` to `Single`.
- Utilize the **Run Connection Automation** option to automate the configuration and connection of the **AXI interrupt controller** IP to the **Zynq UltraScale+ IP**.
  - In the configuration window check all.
  - Set the clock source for the **driving bridge IP**, **slave interface**, and **master interface** to `clk_out2`. This alignment ensures synchronization among different components using the same clock source within the design.
- Establish the connection by linking the `irq` output from the **AXI interrupt controller** IP to the `pl_ps_irq` input of the **Zynq UltraScale+ IP**.



6. Configure Processor Peripherals

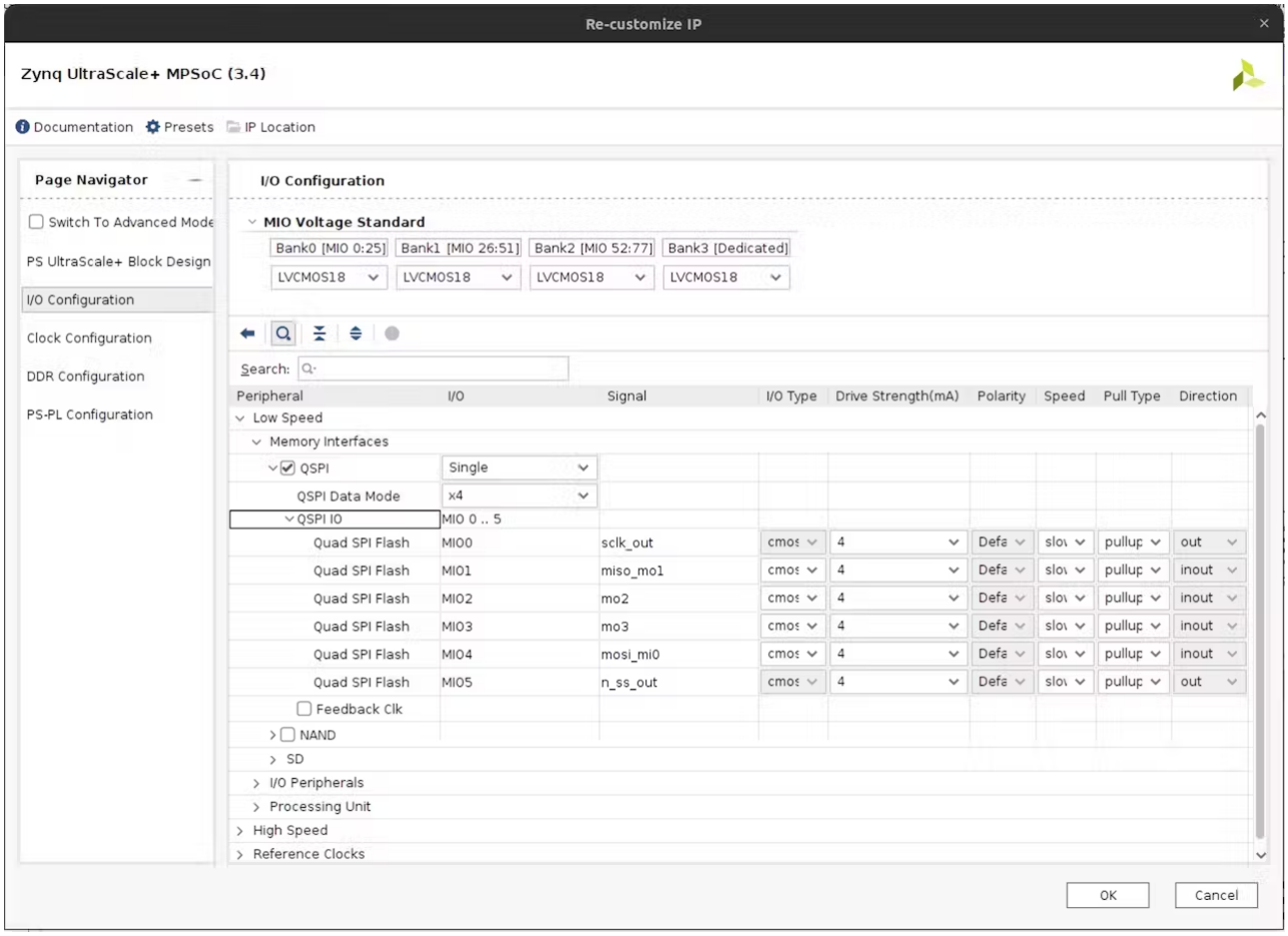
- The primary configuration required involves enabling respective ports for each peripheral on the Zynq MPSoC chip. Open the block design within the Vivado project and double-click on the **Zynq UltraScale+ IP** block to access its configuration window.

> **Note:** MIO pins are dedicated interface pins on the Zynq MPSoC chips connected directly to the ARM-cores. These pins are routed to specific package pins of the chip, inaccessible to the PL (Programmable Logic) of the FPGA, and are unchangeable, eliminating the need for manual mapping of signal names to pin numbers in a constraints file.

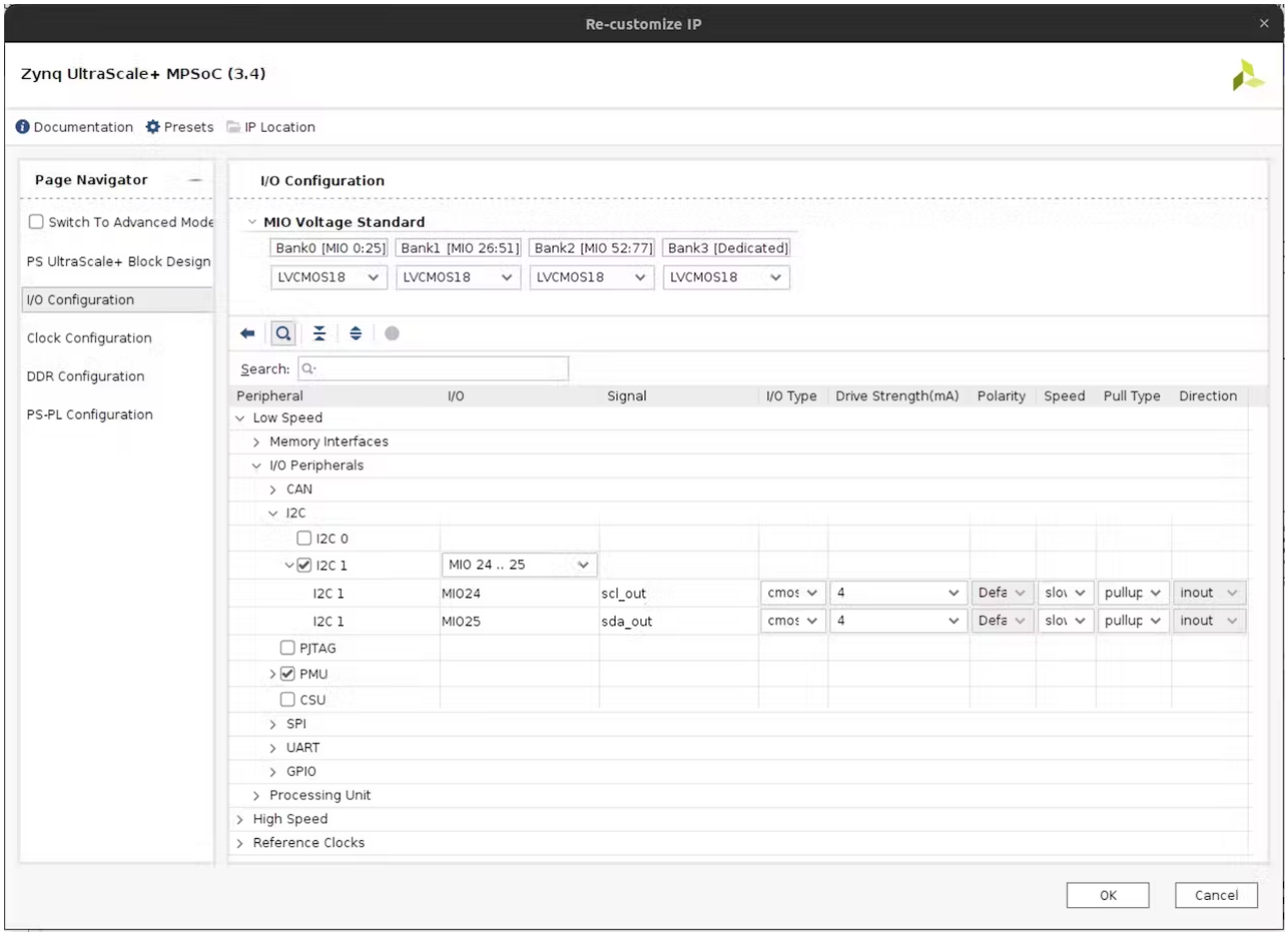
> **Note:** EMIO pins are routed from the PS side through the PL side to the external ports of the FPGA. These pins traverse through the PL side to any FPGA pin, making them interchangeable, requiring signal name mapping to pin numbers in a constraint file.

Begin with the I/O Configuration tab.

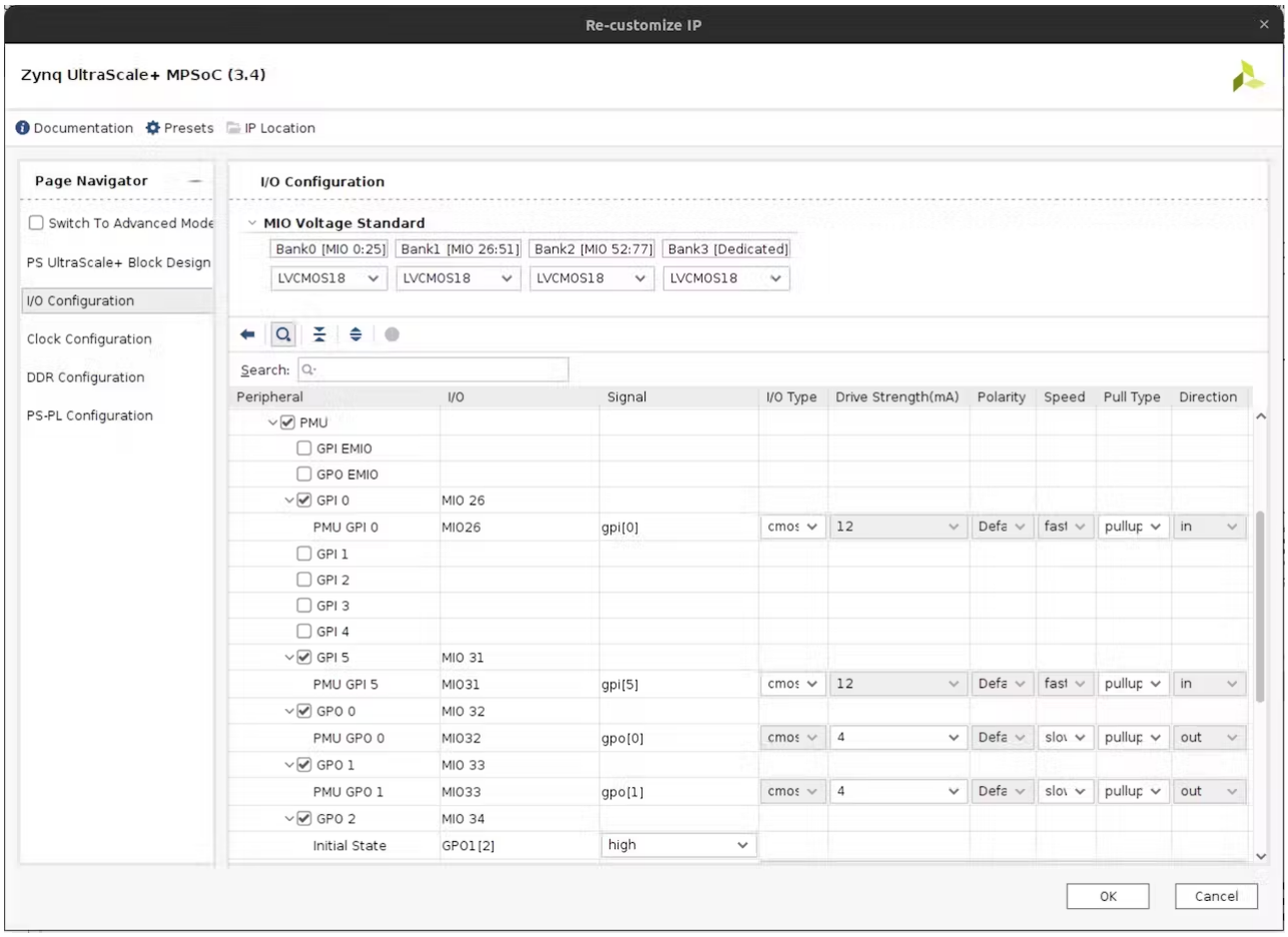
- Under Low Speed > Memory Interfaces, verify the QSPI settings.

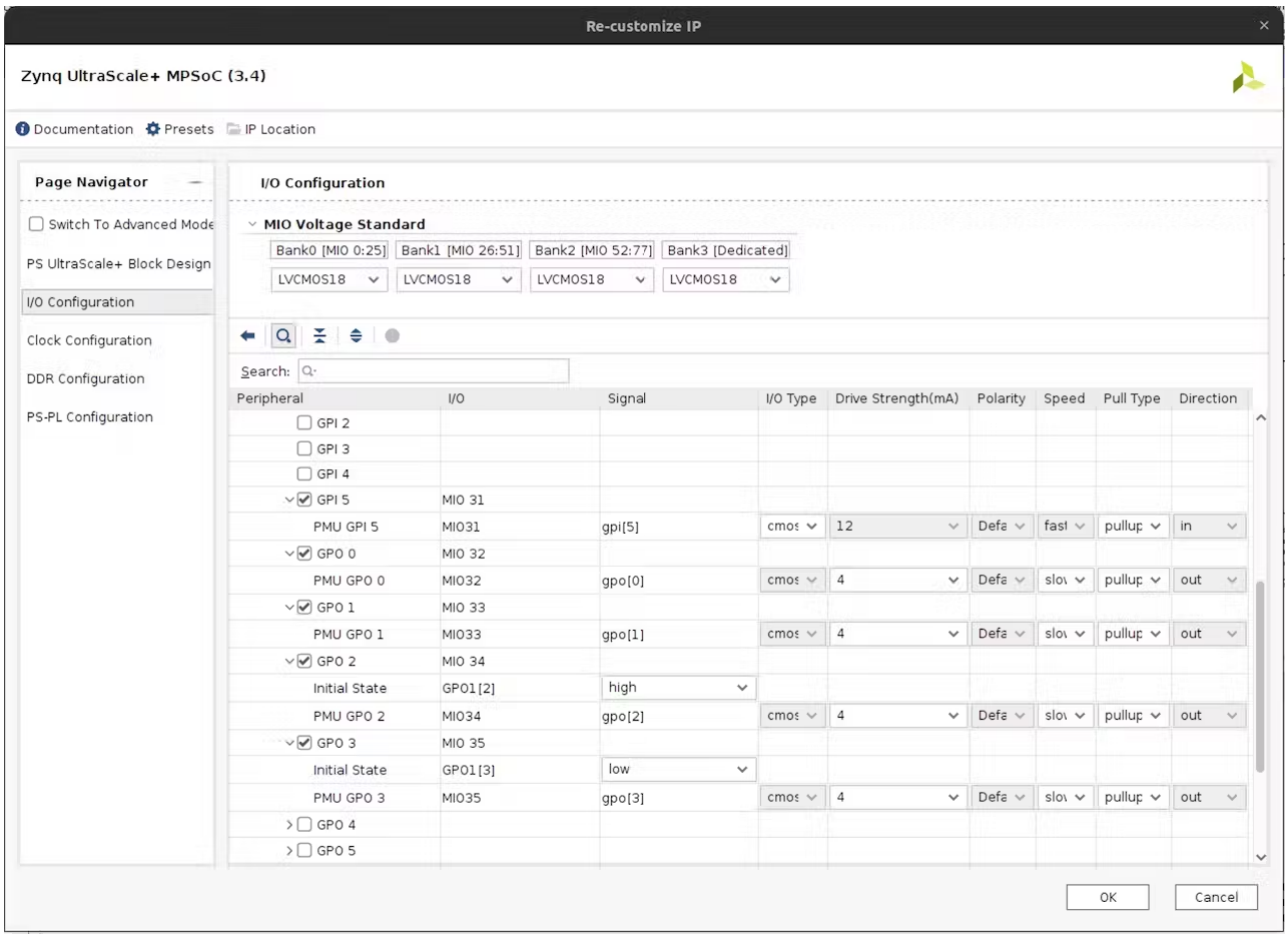


- Under Low Speed > I/O Peripherals, enable I2C1 on MIO pins 24 - 25.

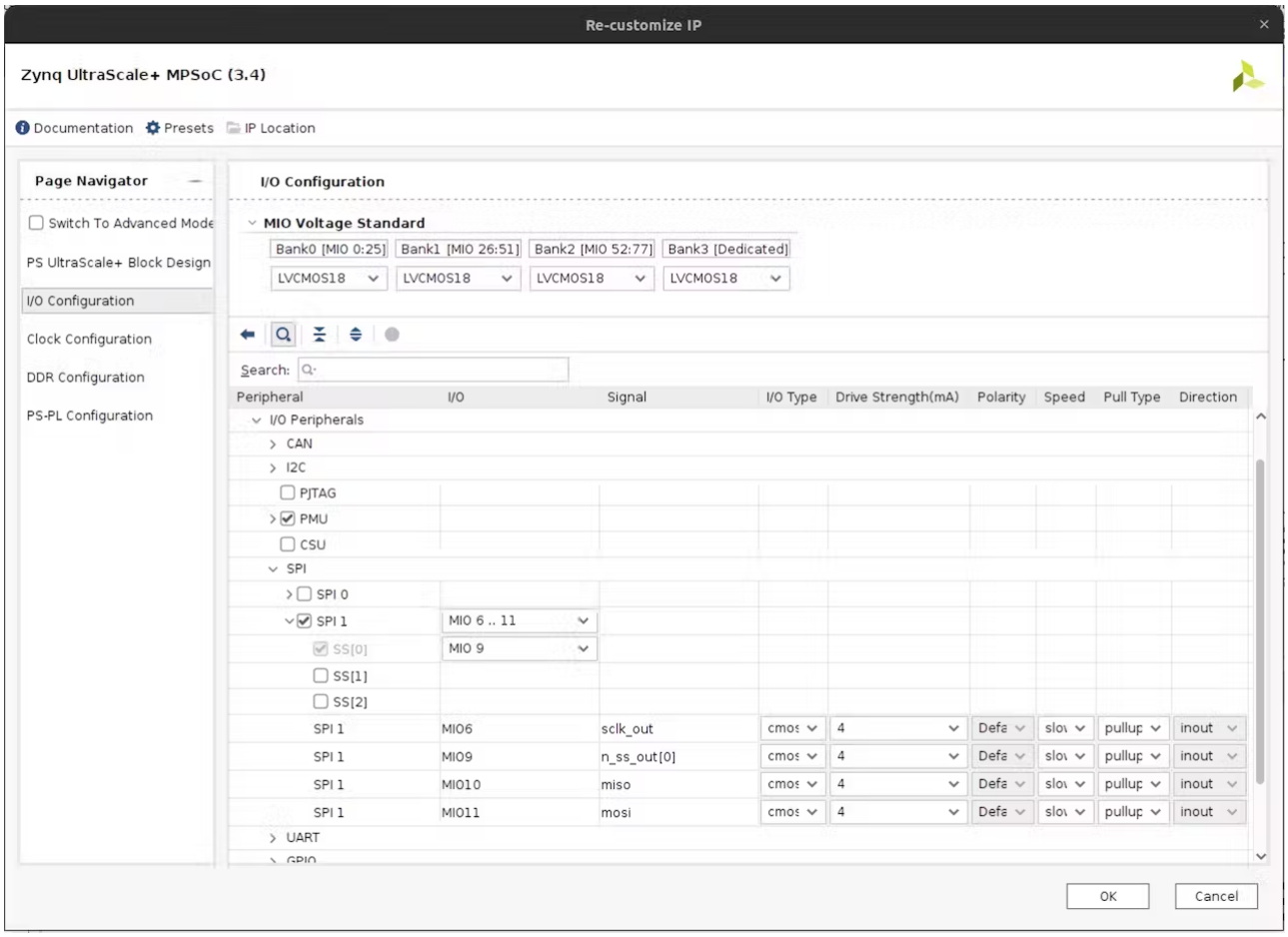


- Confirm PMU settings

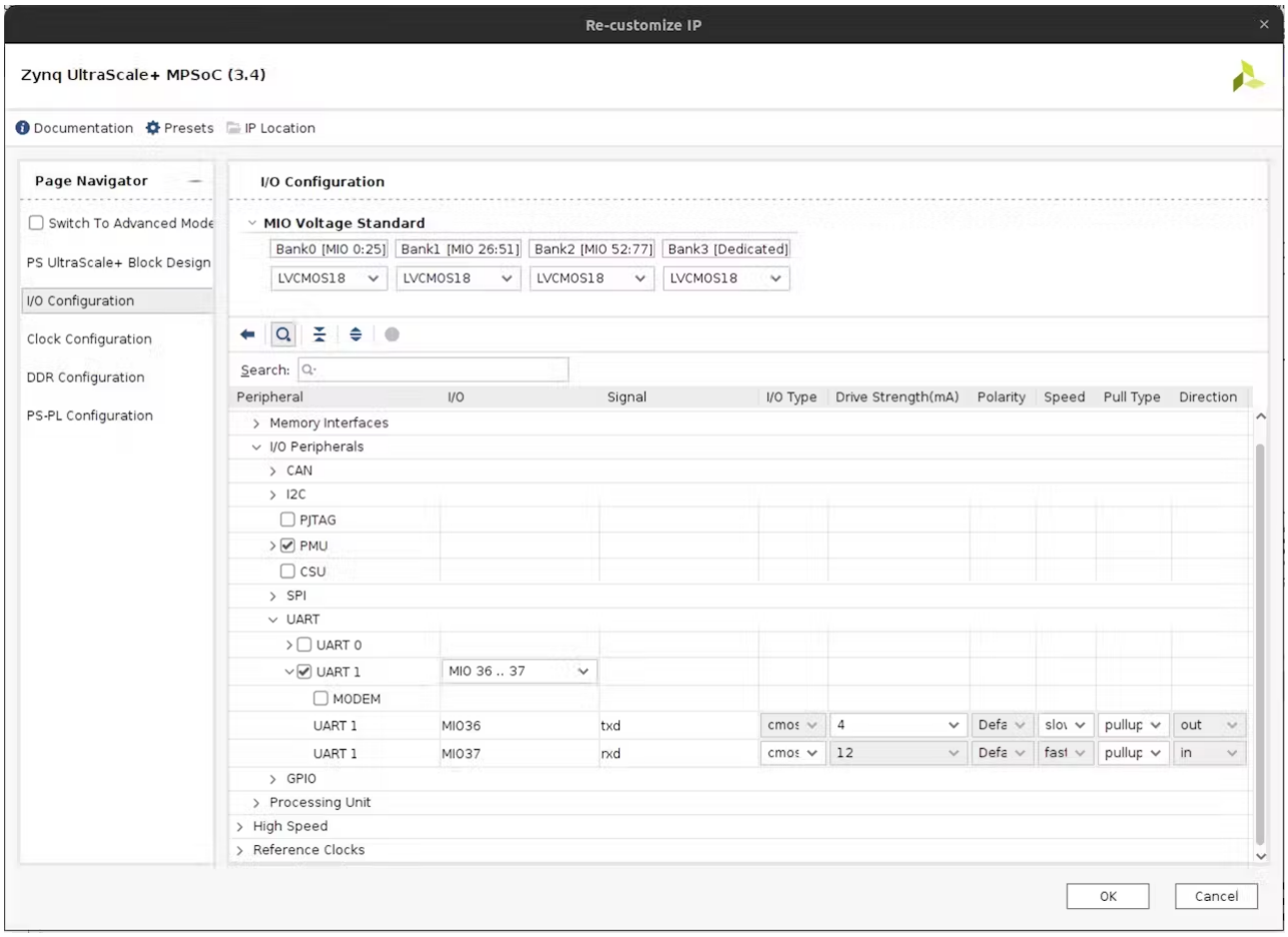




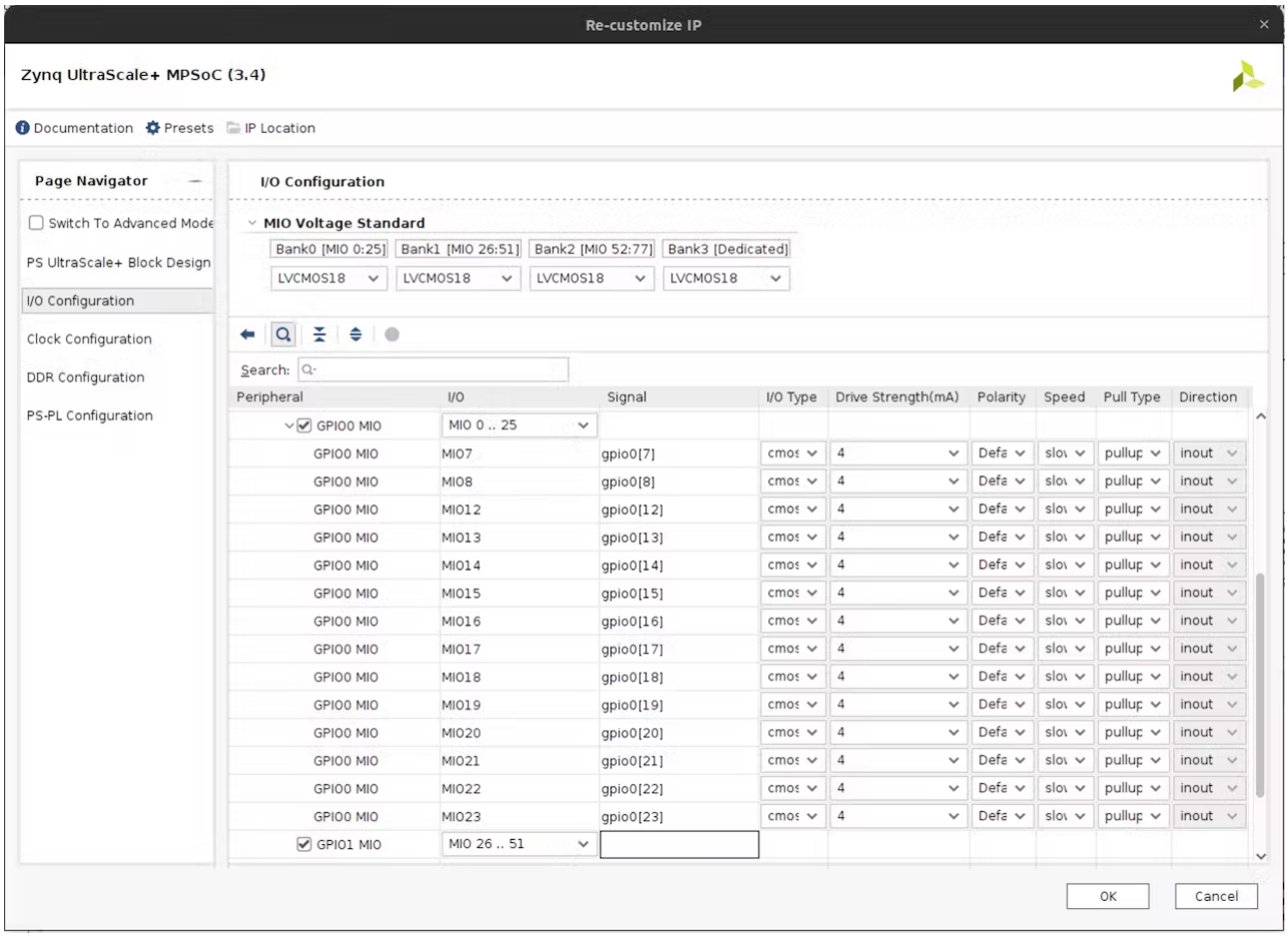
- Enable SPI1 on MIO pins 6 - 11.



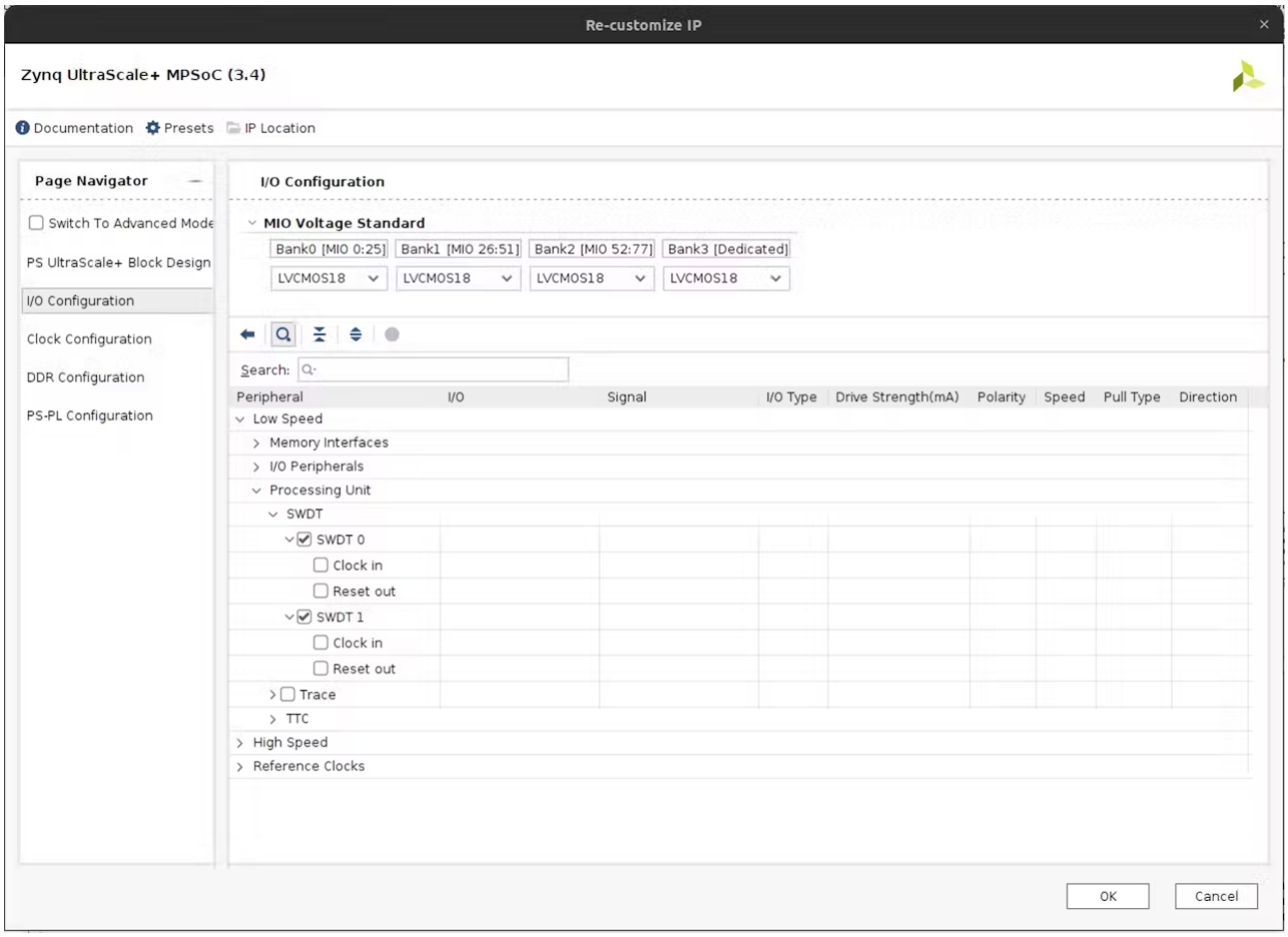
- Enable UART1 on MIO pins 36 - 37.



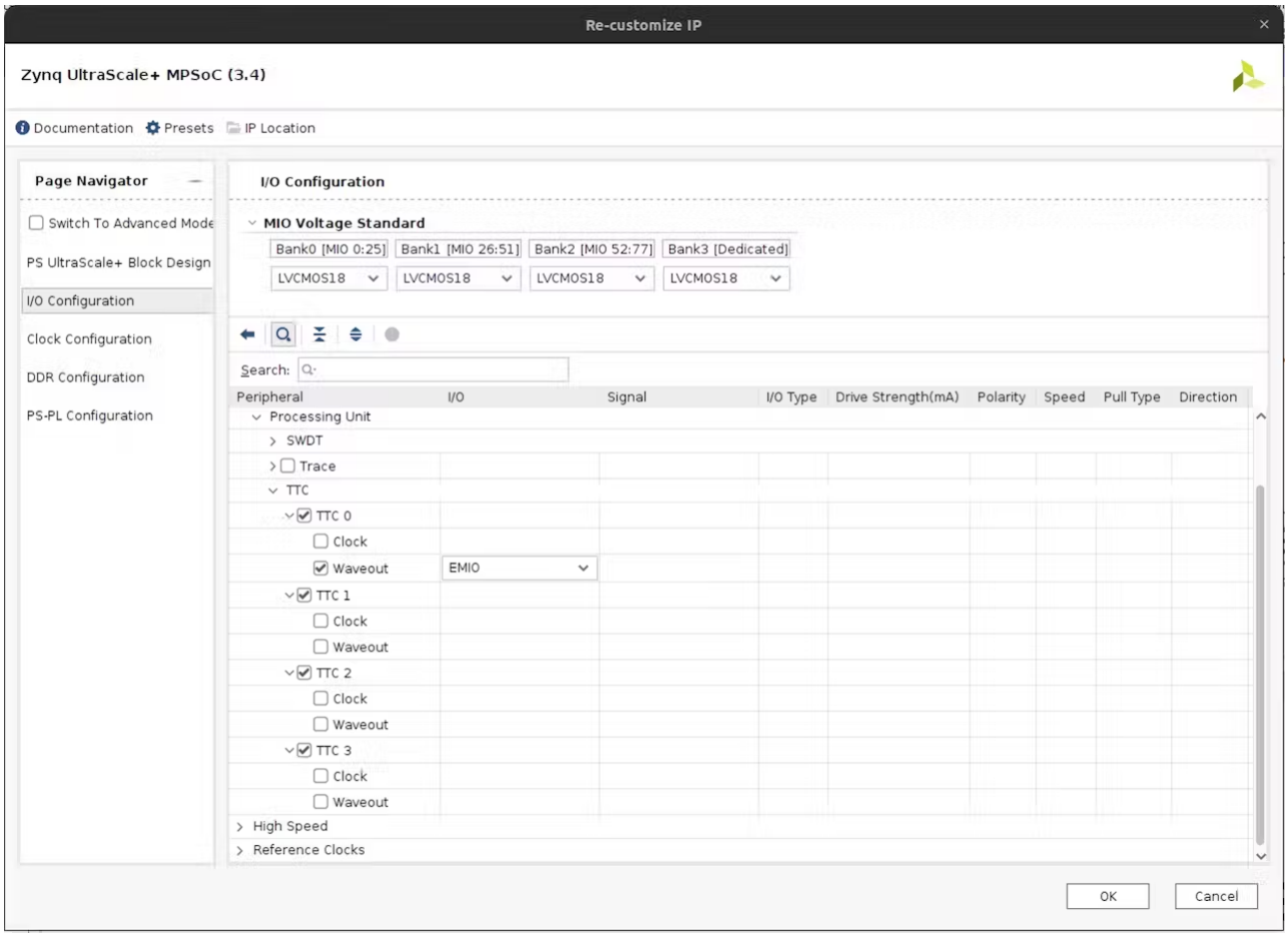
- Enable GPIO0 MIO and GPIO1 MIO.



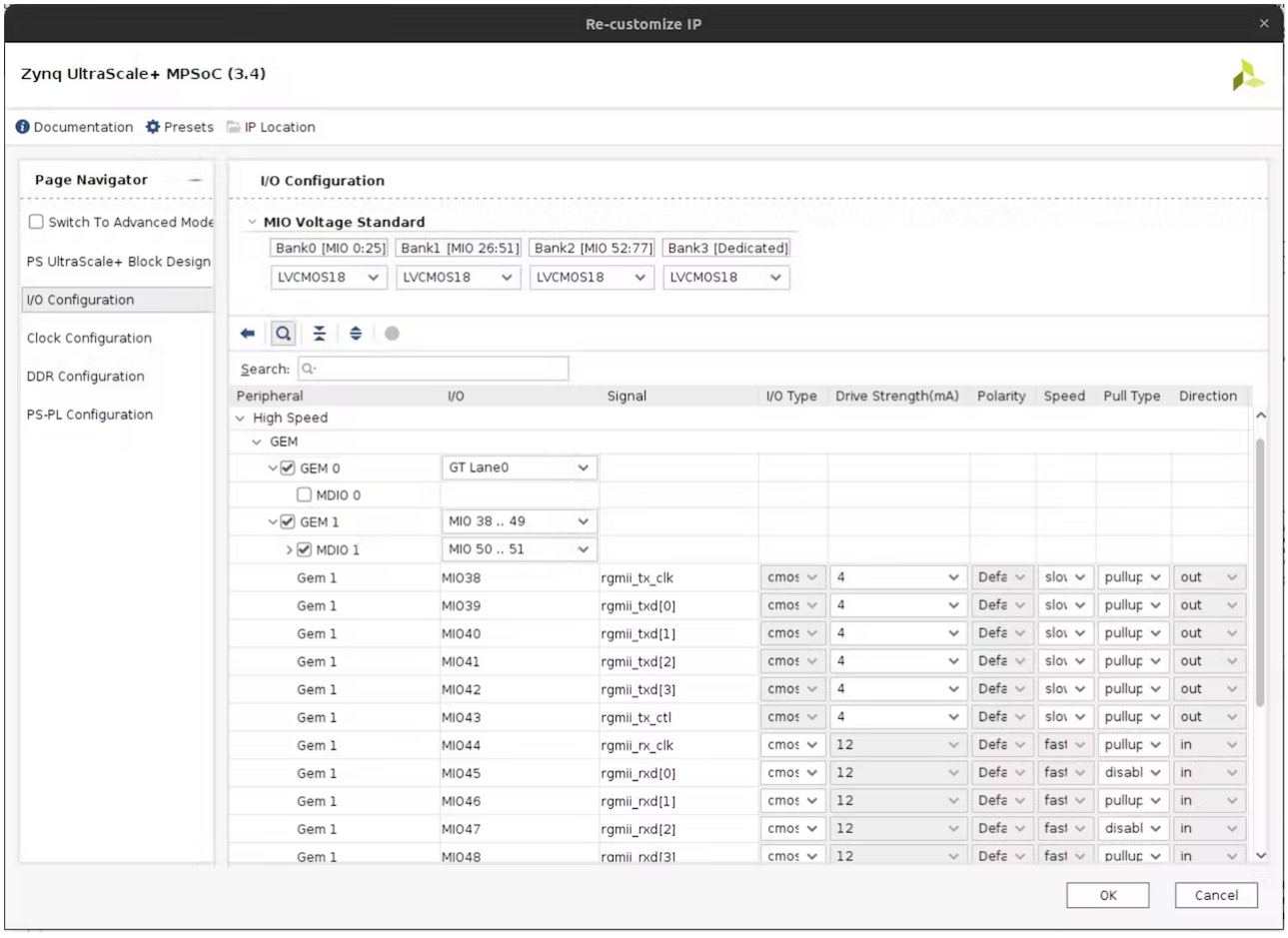
- Enable both system-wide watchdog timers SWDT 0 and SWDT 1 under Processing Unit.



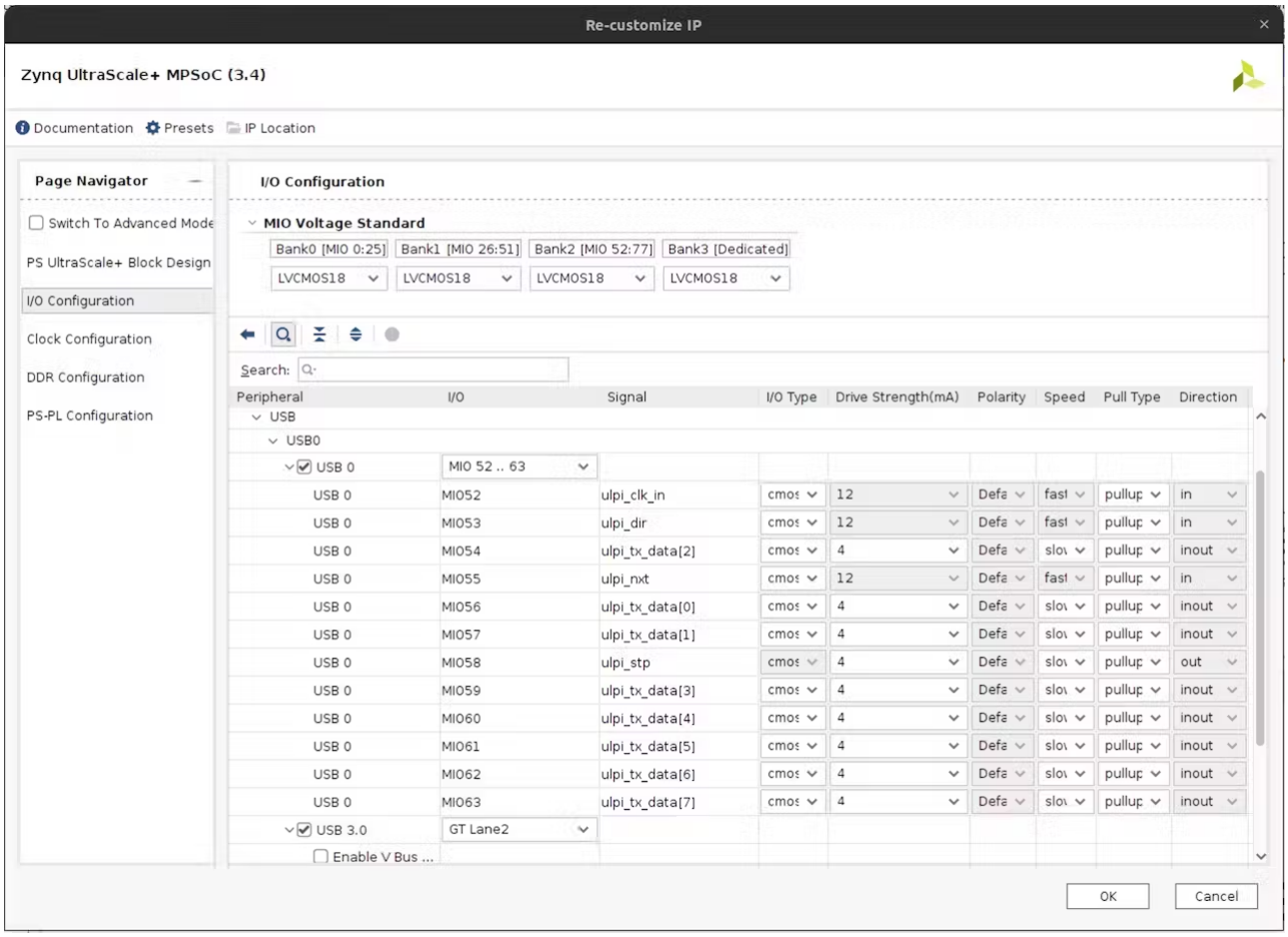
- Enable all four triple timer counters (TTC0 - TTC3), with the first outputting its wave out signal to EMIO.



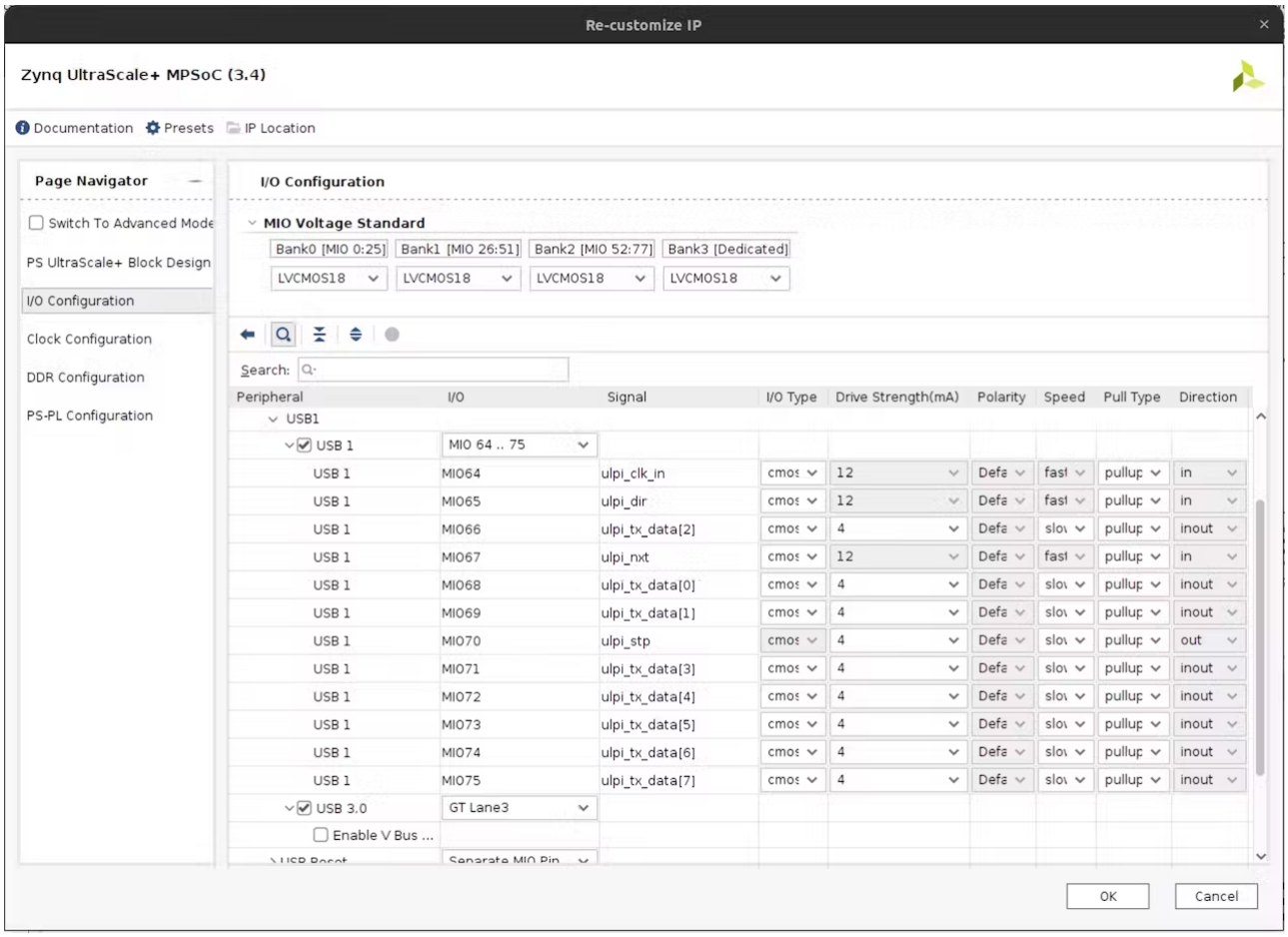
- Under High Speed for the RJ45 Ethernet ports on the KR260, enable GEM 0 on GT Lane0 and GEM 1 on MIO pins 38 - 49, with its MDIO on MIO pins 50 - 51.



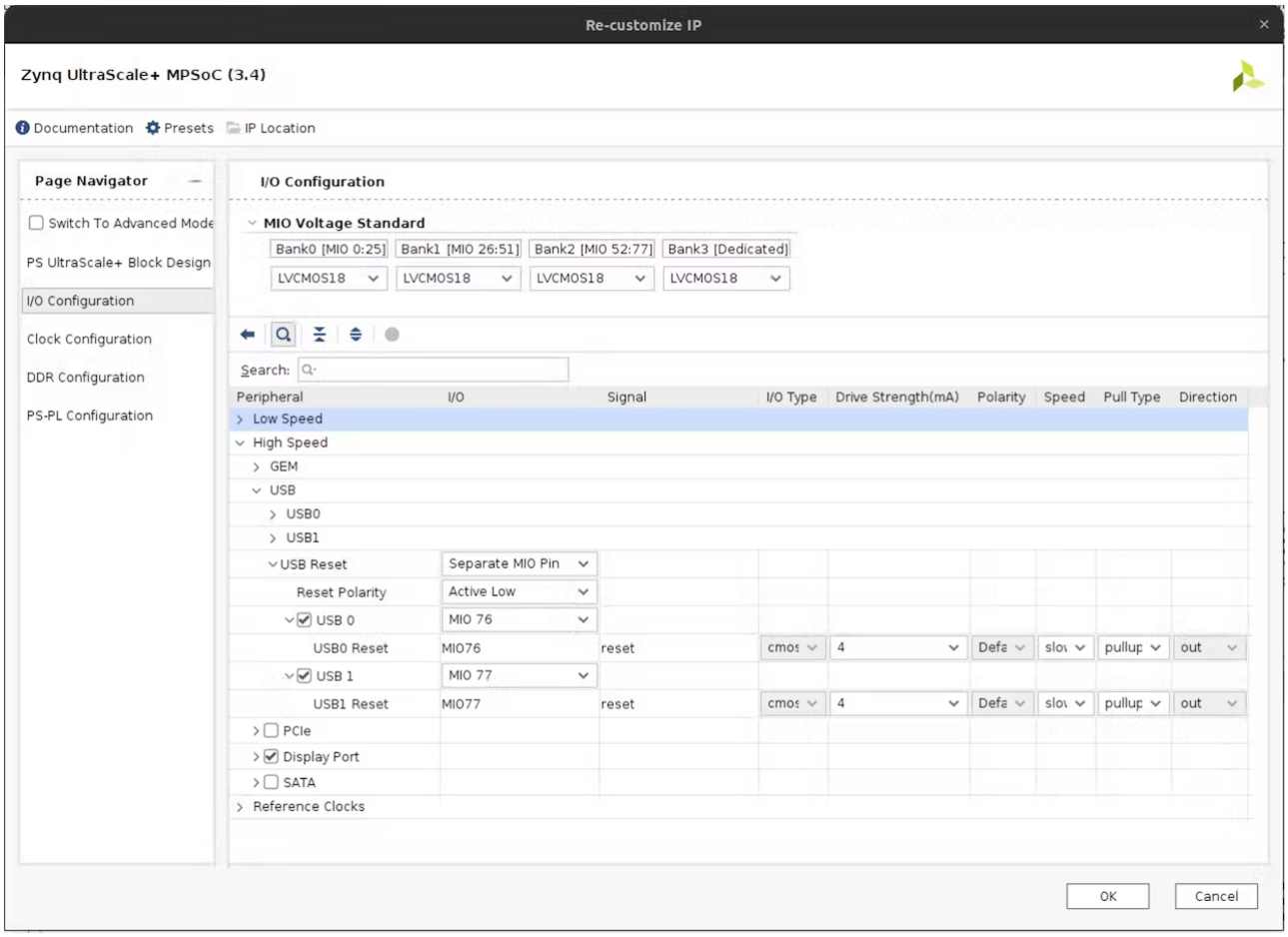
- Under USB0, enable USB0 on MIO pins 52 - 63 and USB 3.0 on GT Lane2.



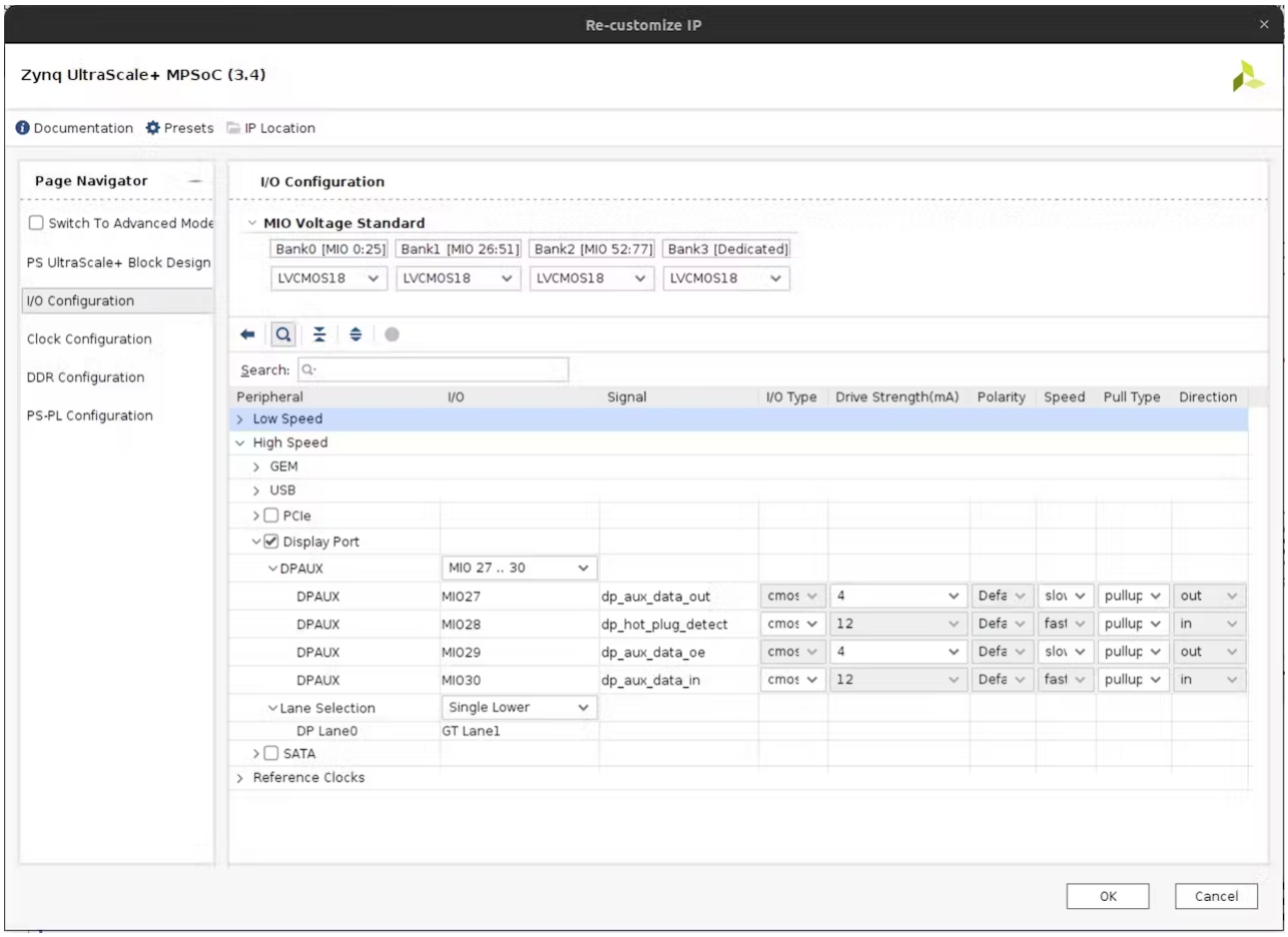
- Under USB1, enable USB1 on MIO pins 64 - 75 and USB 3.0 on GT Lane3.



- Configure the reset pins for the USB ports to use separate MIO pins with an active low polarity. Assign USB 0 reset to MIO pin 76 and USB 1 reset to MIO pin 77.

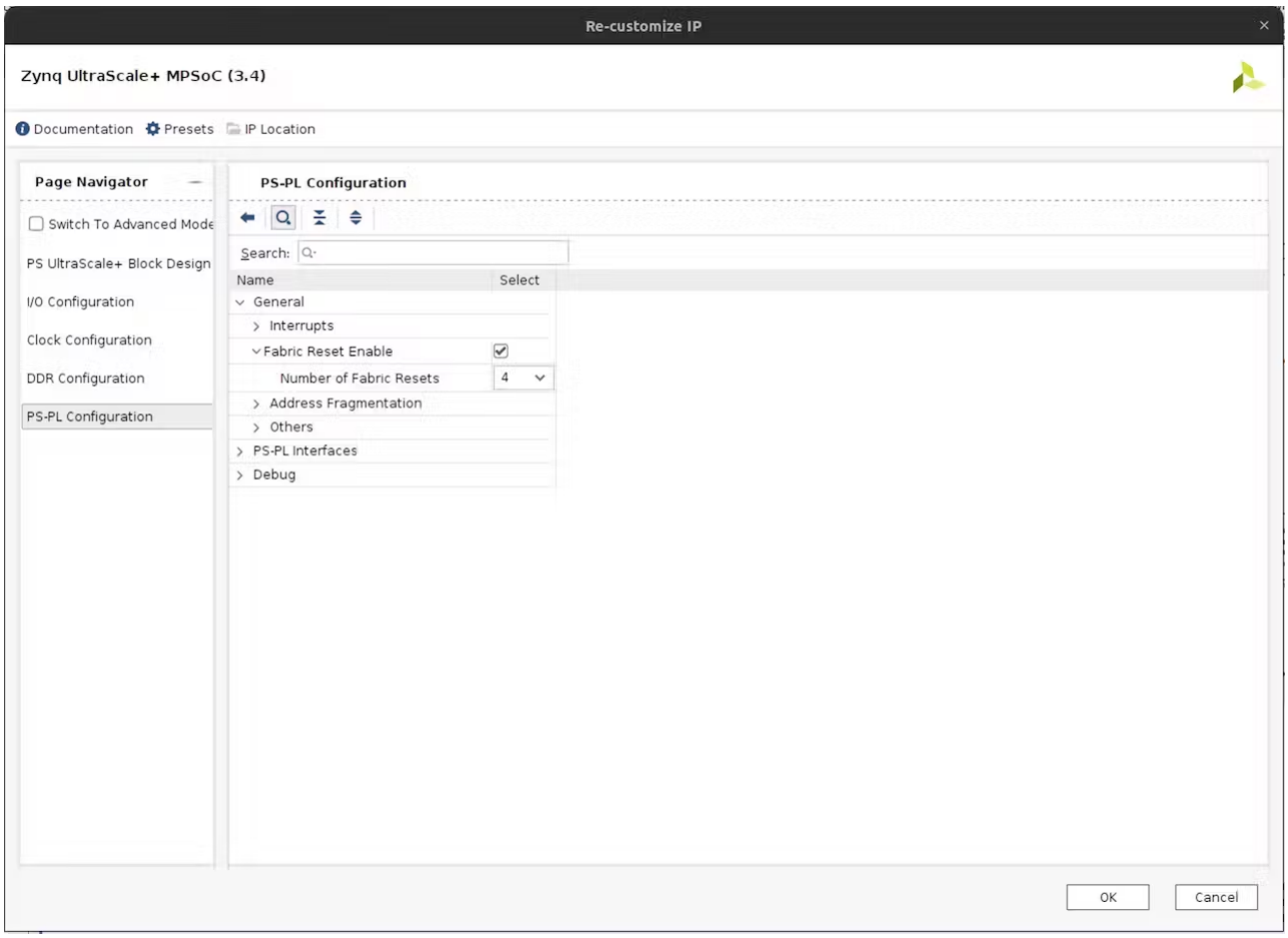


- Enable the DisplayPort on MIO pins 27 - 30 with a Single Lower lane selection on GT Lane1.

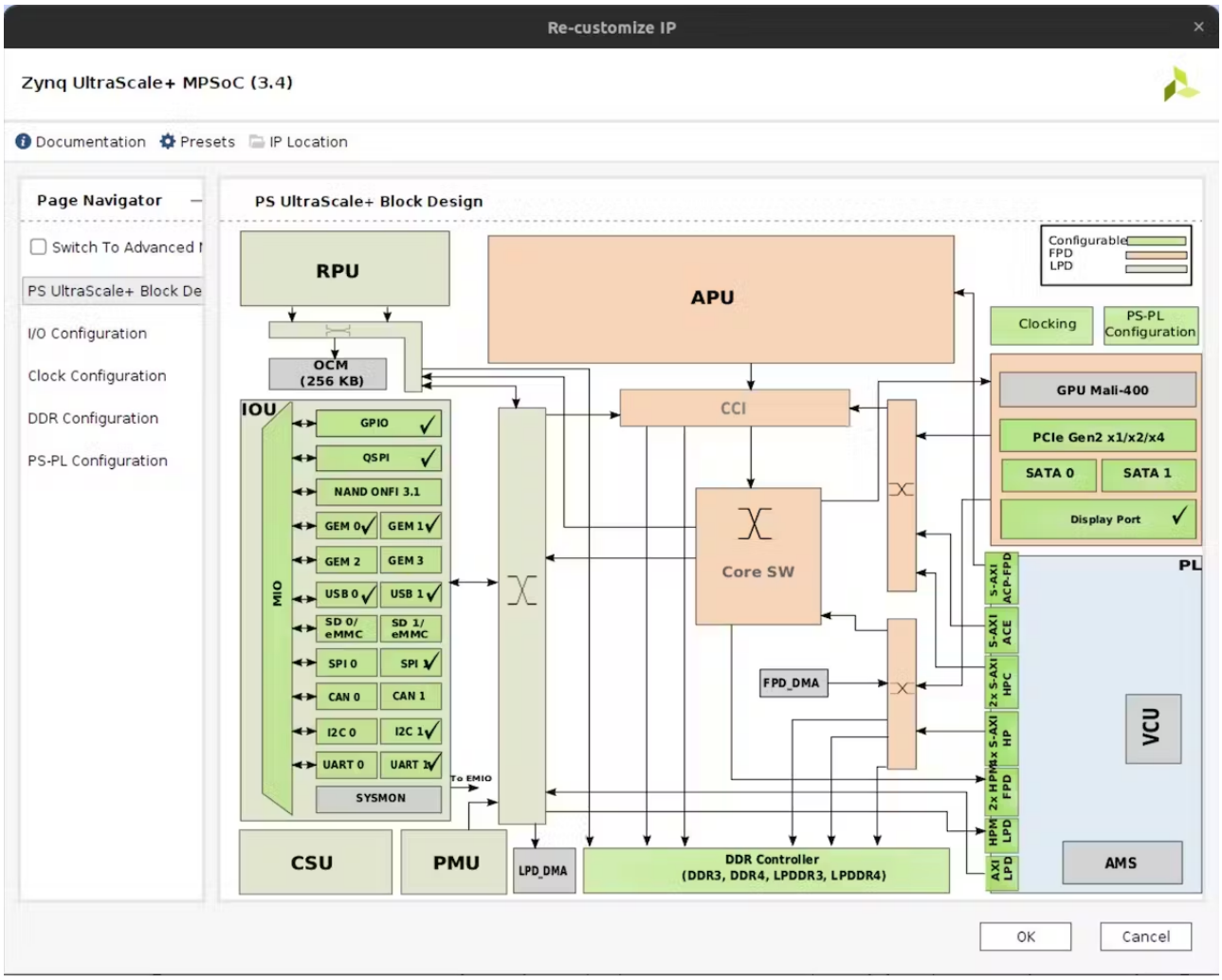


Proceed directly to the PS-PL Configuration tab (PS = Processing System, PL = Programmable Logic).

- Under General > Fabric Reset Enable, increase Number of Fabric Resets from 1 to 4.

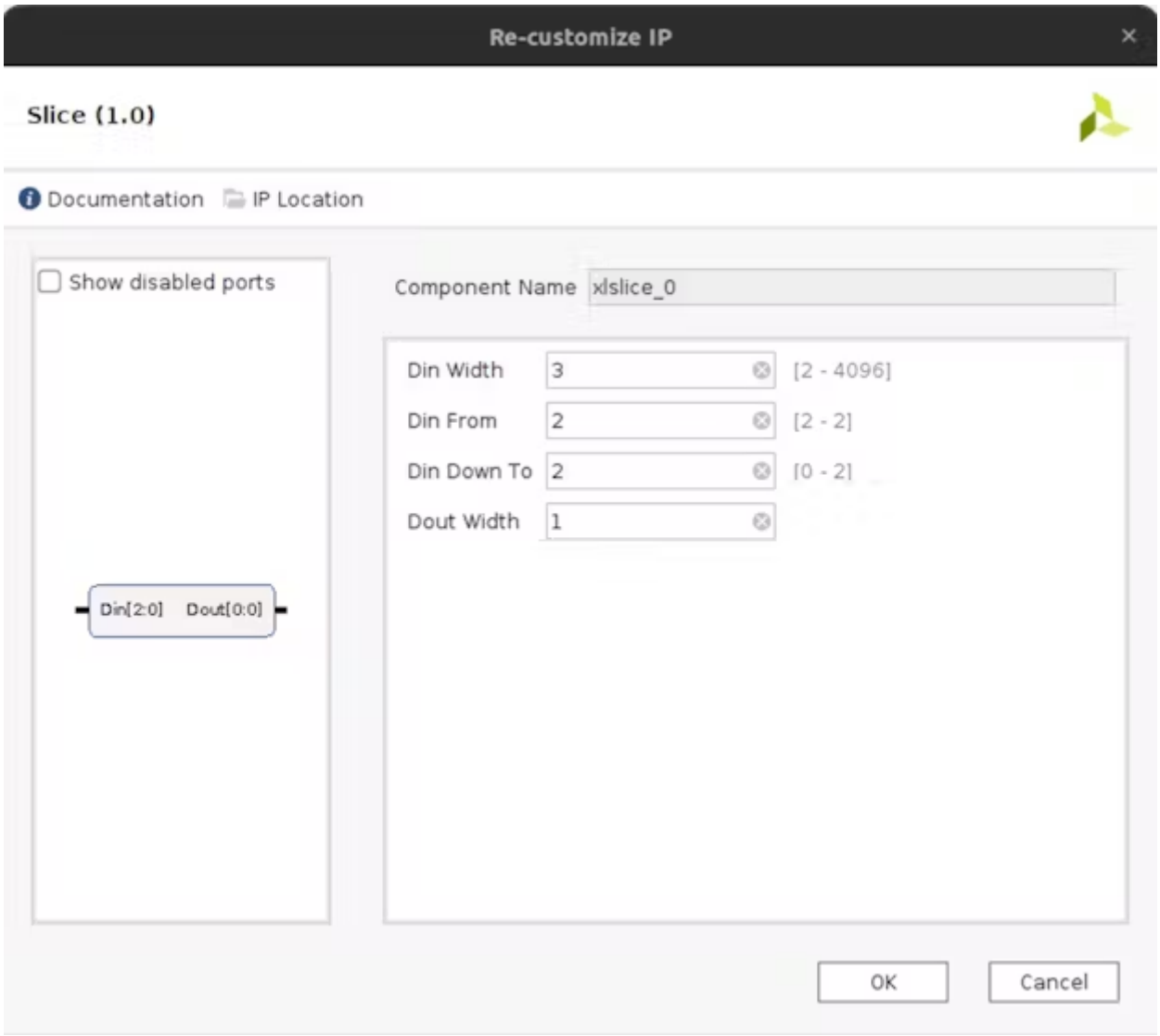


In the end, the block design tab should have check marks on each of the enabled peripherals:

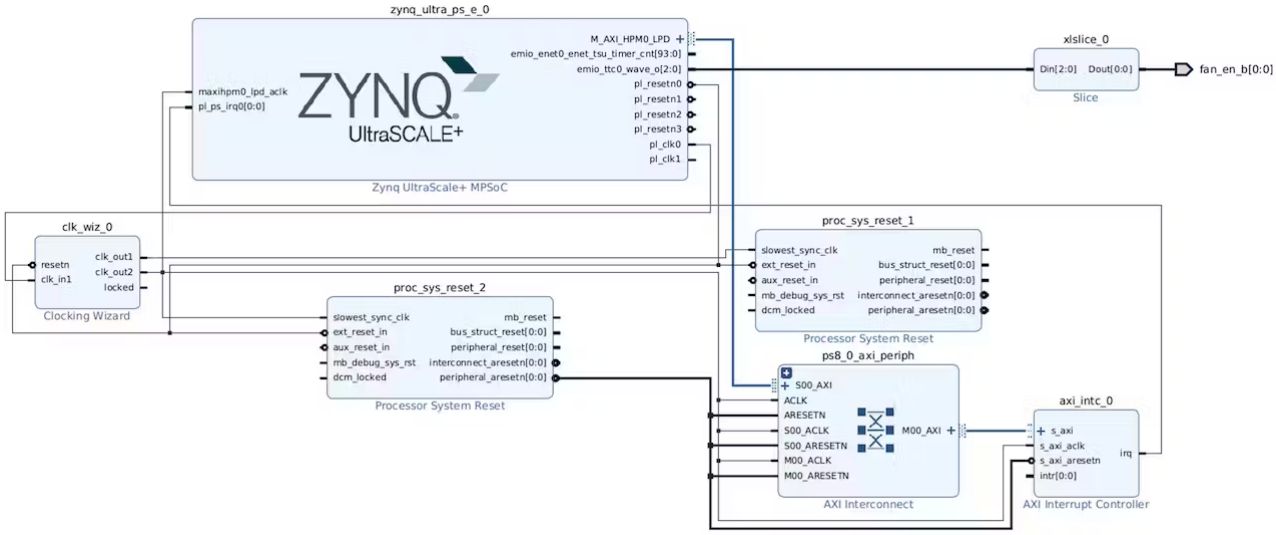


7. CPU Fan Driver IP Block Design

    - Add an Slice IP block and configure it to accept a 3-bit input, outputting specifically the least significant bit (LSB), bit 0:



- Connect the `Din` input of **xslice_0** to `emio_ttc0_wave_o` of the **Zynq MPSoC IP**.
- Right-click on the `Dout` pin of **xslice_0** and opt for the **Make External** option. This action generates a port pin and automatically establishes the connection between the Dout pin and this newly created port. Rename the port to `fan_en_b`, and then regenerate the layout.



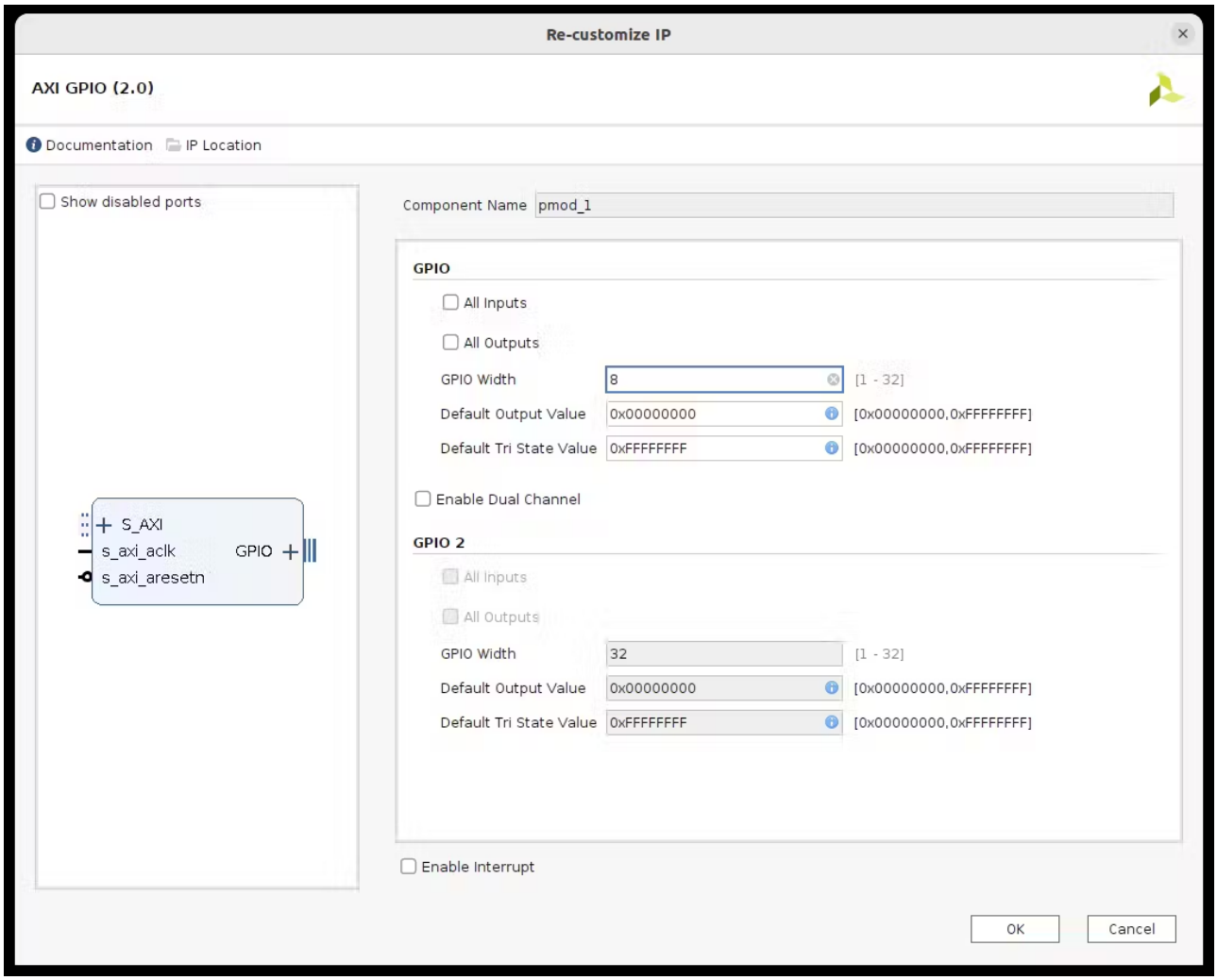
8. Enabling GPIO functionality on PMODs and RPi

> Note: The KR260 has 4 PMODs however PMOD 1's pins will be used for SPI, I2C, and UART so only 3 AXI GPIO IP's will be used for the 3 PMOD's with GPIO functionality.

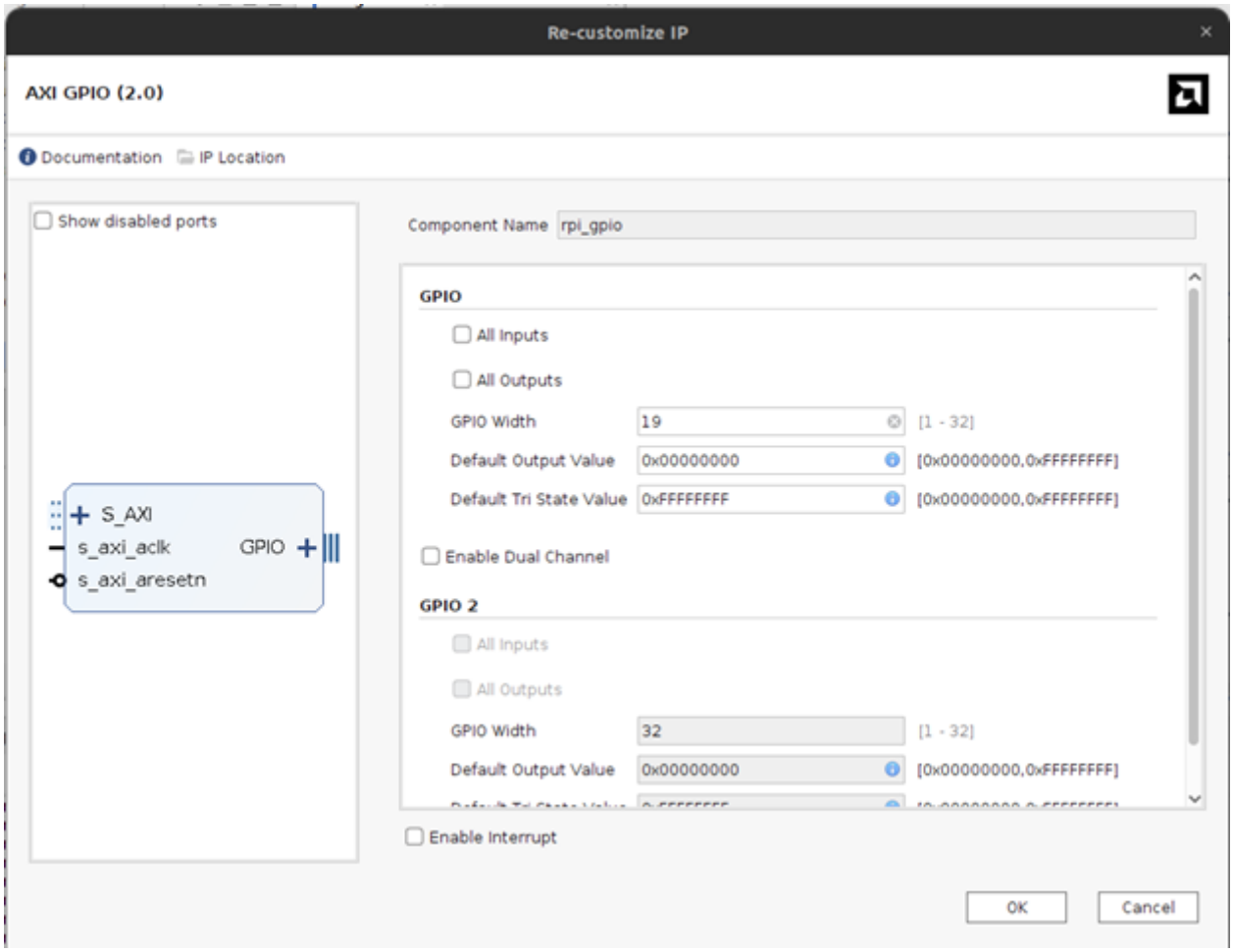
- Add 4 **AXI GPIO** blocks (3 PMODs + 1 RPi)
- Set 3 of the GPIO blocks (the PMOD ones) to have a width of 8

> Note: The width of an AXI GPIO IP represents the number of ports it has. The PMODs each have 12 host ports (pin holes) but 2 are 3.3V power and 2 are ground. These 4 pins have their functionality hardwired in so they do not need to be configured in the FPGA design. This is why the GPIO IP's for the PMODs only have a width of 8.

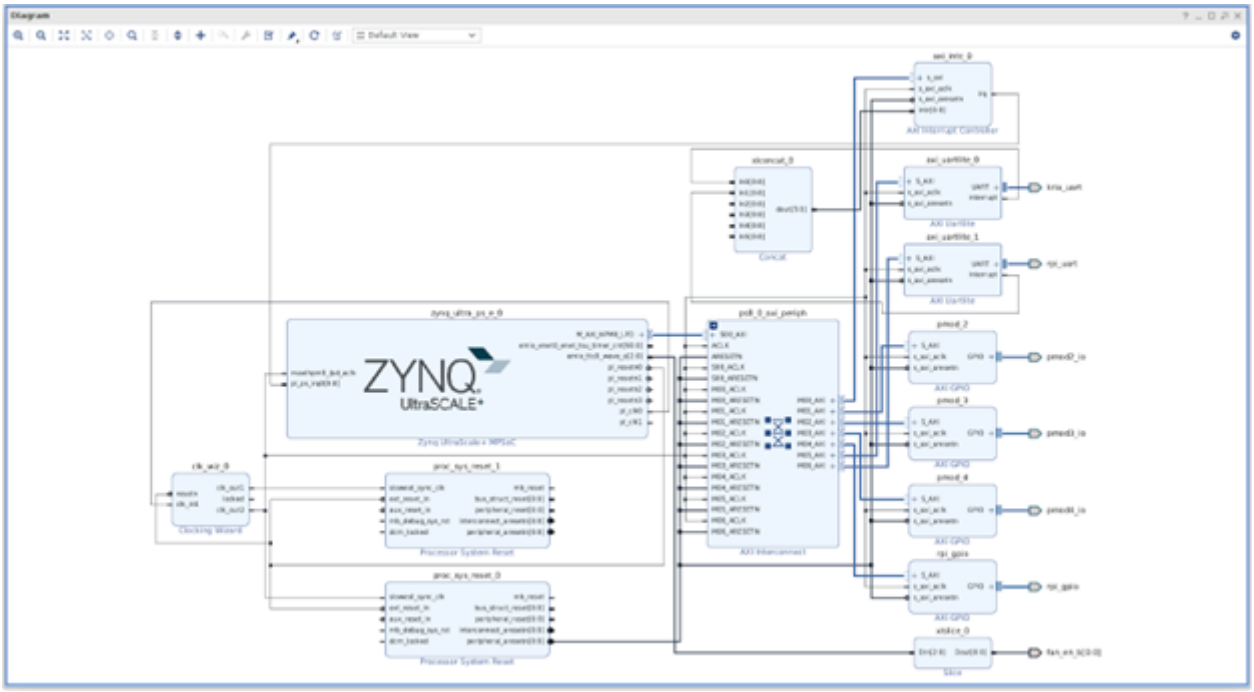
- rename each PMOD AXI GPIO to pmod_2, pmod_3, pmod_4



- Set the 4th GPIO IP to have a width of 19 and rename to rpi_gpio

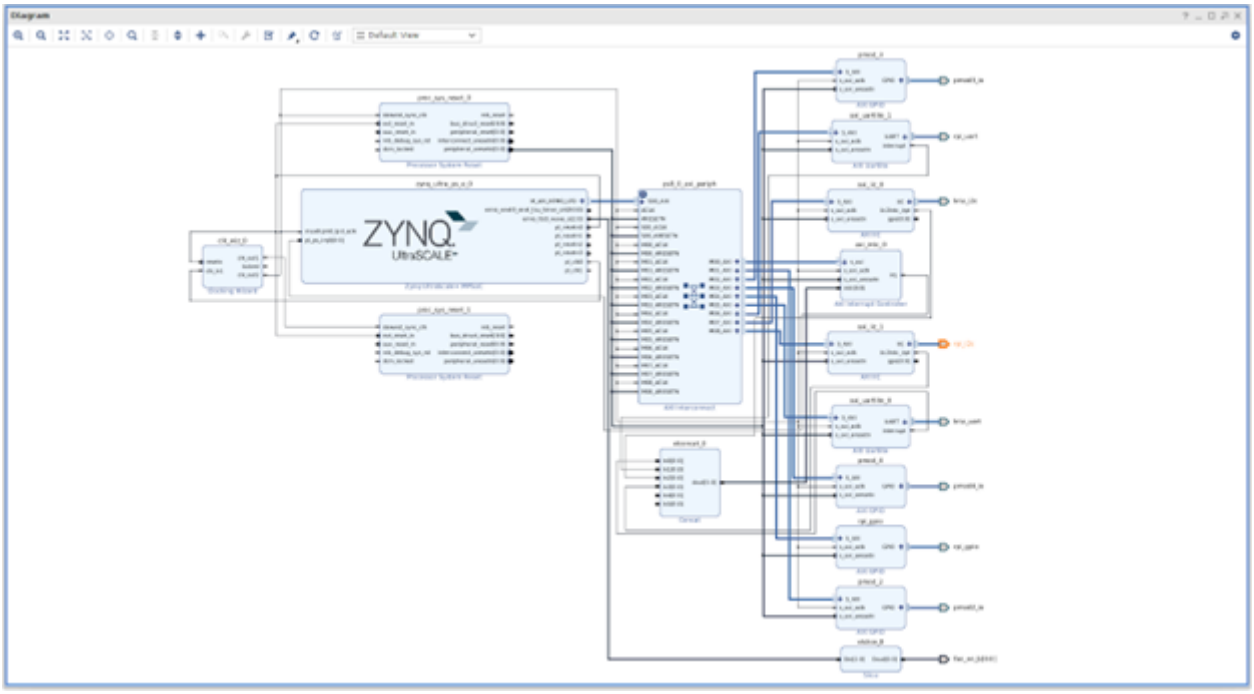


- Run **Automatic Connection** tool and then rename the uart 0 port to `kria_uart` and uart 1 port to `rpi_uart`



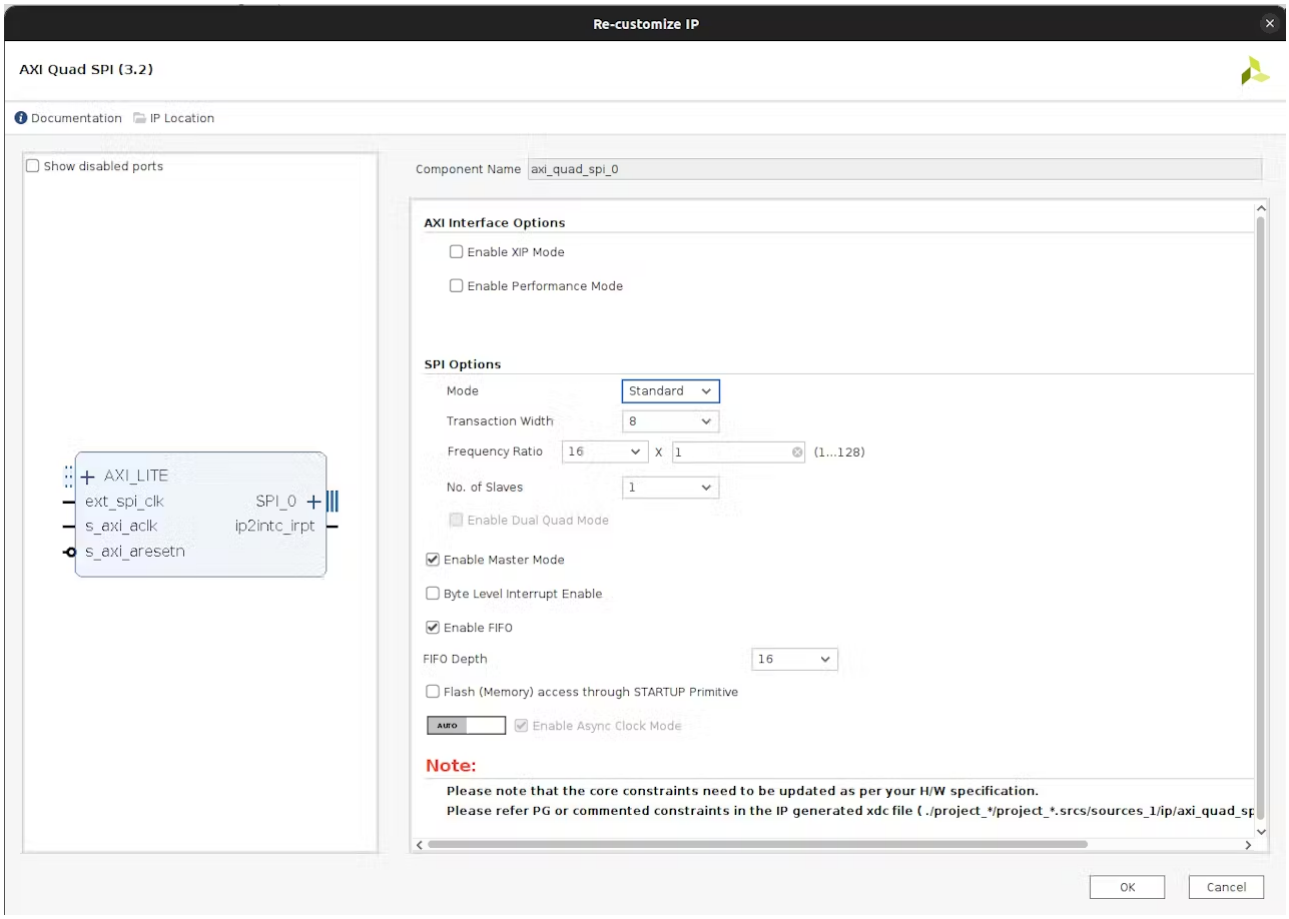
9. Add 2 AXI IIC IPs

- For both: double click and modify **SCL Clock** to `400 kHz` and set the **Address Mode** to `7 bits`
- Run **Automatic Connection** tool and then rename the iic 0 port to `kria_i2c` and uart 1 port to `rpi_i2c`
- Connect the `ii2intc_irpt` on each to `In2` and `In3` respectively on the **Concat**

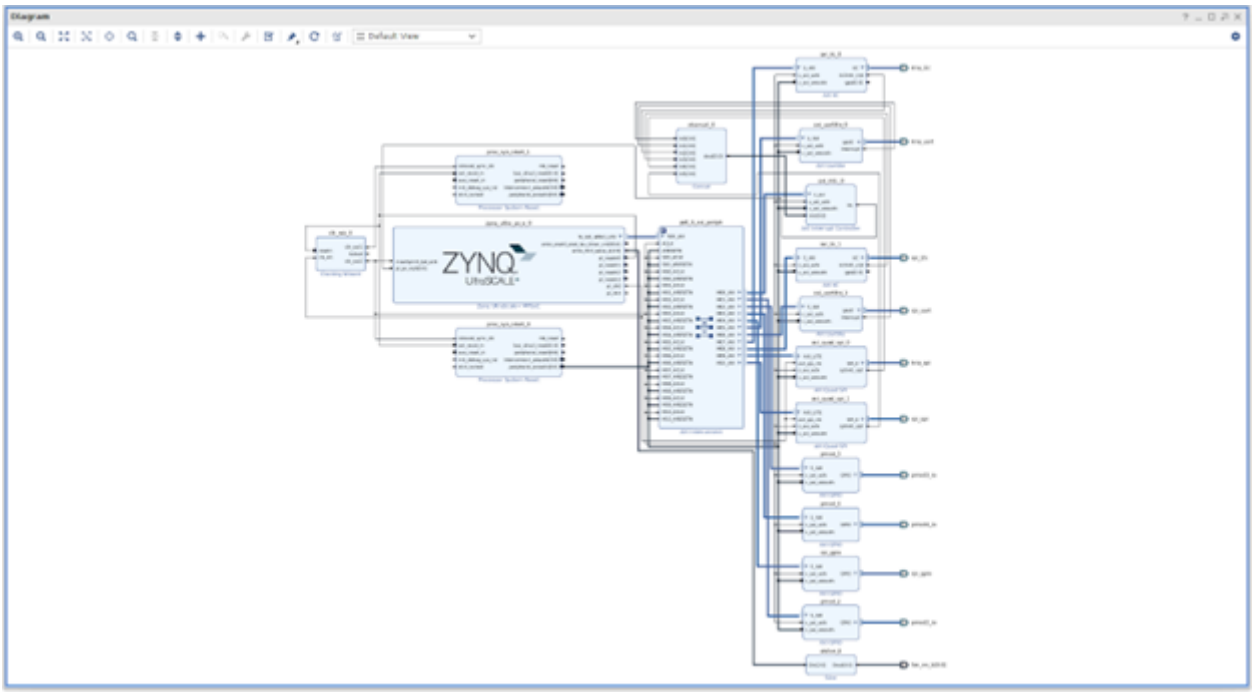


10. Add 2 AXI QUAD SPI IPs

    - For both: double click and modify **Mode** to `Standard`, **Transaction Width** to `8`, and select **Enable Master Mode**



    - Run **Automatic Connection** tool and then rename the spi 0 port to `kria_spi` and spi 1 port to `rpi_spi`
    - Connect both to the **Concat** block via their `ip2_intc_irpt`



11. Validate, Generate block design, Create HDL Wrapper

    - Validate the design by clicking on the Checked Box button in the top bar of the diagram tab
    - Click **Generate Block Design** in the left **Flow Navigator** bar
    - Right click on the block design in the sources tab and choose **Create HDL Wrapper** and then let Vivado auto-manage it

12. Add Constraints

    > Note: Constraints map the pins on the FPGA design to the physical pins on the board

    - Click the plus button in the Sources tab and choose **Add or Create Constraints**
    - Upload the constraint files included in this set of documentation files

13. Generate bitstream in Vivado:

    - Access **Settings** from the **Flow Navigator** window, and within the **Bitstream** tab, enable the `-bin_file` option.
    - Run **Synthesis**, **Implementation**, and **Generate Bitstream**
    - After generating the bitstream, go to the terminal and run the following command (changing file paths as needed):

      ```
      cp
      /home/mackenzie/kria_kr260/kria_kr260.runs/impl_1/kr260_bd_wrapper.bin
      kr260.bit.bin
      ```

      This command copies and renames the bin file to `kr260.bit.bin` in the home directory

14. Export hardware in Vivado:

    - Go to File -> Export -> Export Hardware
    - Make sure to choose "Include bitstream"

- If the project doesn't show up, ensure that "Project is an extensible Vitis platform" is unchecked in project settings

15. Run XSCT:

- Open the XSCT program by running `/home/mackenzie/Xilinx/Vitis/2023.2/bin/xsct` in the terminal
- Run the following commands in XSCT (changing file paths as needed):

```
xsct% hsi open_hw_design
/home/mackenzie/kria_kr260/kr260_bd_wrapper.xsa
xsct% hsi set_repo_path /home/mackenzie/device-tree-xlnx
xsct% hsi create_sw_design device-tree -os device_tree -proc
psu_cortexa53_0
xsct% hsi set_property CONFIG.dt_overlay true [hsi::get_os]
xsct% hsi generate_target -dir kr260_dts
xsct% hsi close_hw_design kr260_bd_wrapper
xsct% exit
```

16. Modify DTSI to allow for SPI:

- File originally contains:

```
axi_quad_spi_0: axi_quad_spi@80070000 {
 bits-per-word = <8>;
 clock-names = "ext_spi_clk", "s_axi_aclk";
 clocks = <&zynqmp_clk 71>, <&misc_clk_0>;
 compatible = "xlnx,axi-quad-spi-3.2", "xlnx,xps-spi-2.00.a";
 fifo-size = <16>;
 interrupt-names = "ip2intc_irpt";
 interrupt-parent = <&axi_intc_0>;
 interrupts = <2 0>;
 num-cs = <0x1>;
 reg = <0x0 0x80070000 0x0 0x10000>;
 xlnx,num-ss-bits = <0x1>;
 xlnx,spi-mode = <0>;
};
axi_quad_spi_1: axi_quad_spi@800a0000 {
 bits-per-word = <8>;
 clock-names = "ext_spi_clk", "s_axi_aclk";
 clocks = <&zynqmp_clk 71>, <&misc_clk_0>;
 compatible = "xlnx,axi-quad-spi-3.2", "xlnx,xps-spi-2.00.a";
 fifo-size = <16>;
 interrupt-names = "ip2intc_irpt";
 interrupt-parent = <&axi_intc_0>;
 interrupts = <5 0>;
 num-cs = <0x2>;
 reg = <0x0 0x800a0000 0x0 0x10000>;
 xlnx,num-ss-bits = <0x2>;
```

```
 xlnx,spi-mode = <0>;
};
```

- Change to:

```
axi_quad_spi_0: axi_quad_spi@80070000 {
 bits-per-word = <8>;
 clock-names = "ext_spi_clk", "s_axi_aclk";
 clocks = <&zynqmp_clk 71>, <&misc_clk_0>;
 compatible = "xlnx,axi-quad-spi-3.2", "xlnx,xps-spi-2.00.a";
 fifo-size = <16>;
 interrupt-names = "ip2intc_irpt";
 interrupt-parent = <&axi_intc_0>;
 interrupts = <2 0>;
 num-cs = <0x1>;
 reg = <0x0 0x80070000 0x0 0x10000>;
 xlnx,num-ss-bits = <0x1>;
 xlnx,spi-mode = <0>;
 spidev@0 {
     compatible = "rohm,dh2228fv";
     reg = <0>;
     spi-max-frequency = <25000000>;
     status = "okay";
     };
};
axi_quad_spi_1: axi_quad_spi@800a0000 {
 bits-per-word = <8>;
 clock-names = "ext_spi_clk", "s_axi_aclk";
 clocks = <&zynqmp_clk 71>, <&misc_clk_0>;
 compatible = "xlnx,axi-quad-spi-3.2", "xlnx,xps-spi-2.00.a";
 fifo-size = <16>;
 interrupt-names = "ip2intc_irpt";
 interrupt-parent = <&axi_intc_0>;
 interrupts = <5 0>;
 num-cs = <0x2>;
 reg = <0x0 0x800a0000 0x0 0x10000>;
 xlnx,num-ss-bits = <0x2>;
 xlnx,spi-mode = <0>;
 spidev@0 {
     compatible = "rohm,dh2228fv";
     reg = <0>;
     spi-max-frequency = <25000000>;
     status = "okay";
     };
     spidev@1 {
     compatible = "rohm,dh2228fv";
     reg = <1>;
     spi-max-frequency = <25000000>;
     status = "okay";
     };
};
```

17. Create dtbo from XSA:
    - Go to the `Xilinx/Vitis/2023.2/bin/kr260_dts` directory
    - Run the following command:

```
dtc -@ -O dtb -o pl.dtbo pl.dtsi
```

    - Go back to the home directory by running `cd`
    - Copy the dtbo file to the desired location:

```
cp Xilinx/Vitis/2023.2/bin/comms_dts/pl.dtbo kr260.dtbo
```

18. Create a new folder to hold the .bit.bin, .dtbo, and shell.json files
19. Create a shell.json file:
    - Inside the new folder, create a file named `shell.json`
    - Add the following content to the `shell.json` file:

```
{
      "shell_type" : "XRT_FLAT",
      "num_slots" : "1"
}
```

20. Copy the folder with the 3 files to the KR260

On the KR260

11. Put the 3 files into a new firmware folder:
    - In the terminal, navigate to the folder you've just added:

```
sudo mkdir /lib/firmware/xilinx/kr260_custom
sudo cp gpio.* shell.json /lib/firmware/xilinx/kr260_custom/
```

12. Run `sudo xmutil listapps` and you should see **kr260_custom**
13. Load the new overlay:

```
sudo xmutil unloadapp
sudo xmutil loadapp kr260_custom
```

Testing

Testing GPIOs

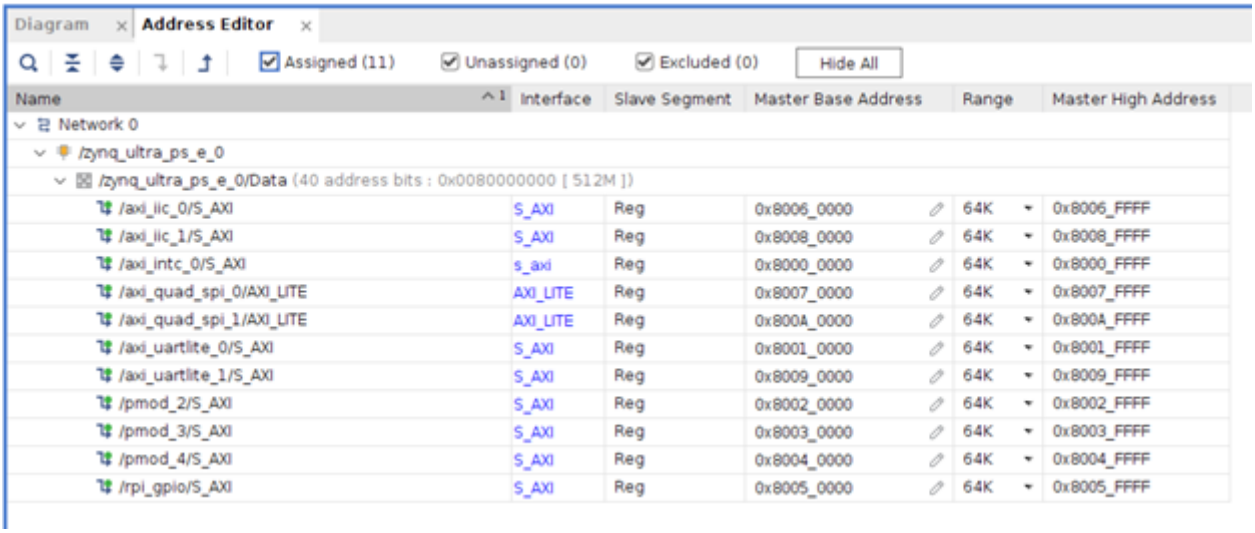
- List all available GPIOs:

```
ls /sys/class/gpio/
```

- For each gpiochip print the address it corresponds to:

```
cat /sys/class/gpio/gpiochip508/label
```

- Each gpiochip address should map to the addresses for the GPIO blocks you see in Vivado under **Address Editor**



When I did it I ended up with the following (You may end up with something different):

GPIO Name	GPIO Chip
PMOD2 (80020000.gpio)	318
PMOD3 (80030000.gpio)	310
PMOD4 (80040000.gpio)	302
RPI (80050000.gpio)	283
ZYNQMP_GPIO	334
FIRMWARE:ZYNQMP-FIRMWARE	508

slg7xl45106

- Each GPIO Chip number is the 1st pin of that block. So to PMOD2 pin 1 would be gpiochip 318, PMOD2 pin 2 would be gpiochip 319, and so on.
- Attach an LED to the pin you want to test and ground
- In CLI (using pin 318 as an example)

    - Set 318 as an output pin: `echo 318 | sudo tee /sys/class/gpio/export echo out | sudo tee /sys/class/gpio/gpio318/direction`
    - Turn on the LED: `echo 1 | sudo tee /sys/class/gpio/gpio318/value`
    - Turn off the LED: `echo 0 | sudo tee /sys/class/gpio/gpio318/value`
- In Python:
    - Ensure periphery is installed
    - Run blink.py ensuring that you change `gpio_out` to use whatever pin you're testing

Testing Uart

- Verify that the ports were added: `sudo dmesg | grep serial` You should see

```
[   3.114618] ff010000.serial: ttyPS1 at MMIO 0xff010000 (irq = 51, base_baud =
6249999) is a xuartps
[   17.662680] systemd[1]: Created slice Slice /system/serial-getty.
[ 1150.955943] 80010000.serial: ttyUL0 at MMIO 0x80010000 (irq = 69, base_baud =
0) is a uartlite
[ 1150.972285] 80090000.serial: ttyUL1 at MMIO 0x80090000 (irq = 70, base_baud =
0) is a uartlite
```

- You'll need to modify the permissions for the device: `sudo chmod 777 /dev/ttyUL0` (for each one)
- Install pyserial: `python -m pip install pyserial`
- Create a loopback connection between the txd (transmit) and rxd (receive) pins on either the RPi headers or PMOD1, whichever you want to test
- Run the uartTest.py file in this directory. Make sure to change `ser.port` to wichever one you're testing.
- You should see `Hi, this is a kria test` on the console

Testing I2c

- Verify I2C ports were added: `i2cdetect -l` You should see:

```
i2c-1   unknown     Cadence I2C at ff030000             N/A
i2c-2   unknown     ZynqMP DP AUX                       N/A
i2c-3   unknown     i2c-1-mux (chan_id 0)               N/A
i2c-4   unknown     i2c-1-mux (chan_id 1)               N/A
i2c-5   unknown     i2c-1-mux (chan_id 2)               N/A
i2c-6   unknown     i2c-1-mux (chan_id 3)               N/A
i2c-7   unknown     xiic-i2c 80080000.i2c               N/A
i2c-8   unknown     xiic-i2c 80060000.i2c               N/A
```

- Grant the necessary permissions to execute and control the peripheral:

```
sudo chmod 777 /dev/i2c-7
sudo chmod 777 /dev/i2c-8
```

- i2c-7 is PMOD1, i2c-8 is RPi
- Connect an I2C device
- To check the connected device, utilize the following command: `i2cdetect -y -r 7` or `i2cdetect -y -r 8` depending on where you've connected your device
- You'll receive a response similar to the following, in this case, there is a device connected at address 23:

```
0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- 23 -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

- From here the code to test the connection will depend on what device you're connecting
  - You will need to install smbus: `python -m pip install smbus`
  - In your code the `bus` will be either `7` or `8` depending on where your device is connected

Testing SPI

- List all SPI devices: `ls /dev | grep spi` You should see:

```
spidev3.0
spidev4.0
spidev4.1
```

- modify permissions for each device: `sudo chmod 777 /dev/spidev3.0`
- Install spidev: `python -m pip install spidev`
- Create loopback connection bettween MOSI and MISO pins
- run spitest.py in this directory ensuring that spi.open is set to the correct device you're testing: either `(3, 0)`, `(4, 0)`, or `(4,1)`
- You should see `[70, 65, 66, 73, 65, 78]`