Hands-On 3: Vitis AXI GPIO

Objective

- The goal of this hands-on exercise is to get familiar with the AXI GPIO module from Xilinx
  - Part 1: Create a Vivado project with AXI GPIO input/output interfaces
  - Part 2: Create a Vitis project based on polling
  - Part 3: Create a Vitis project based on interrupts

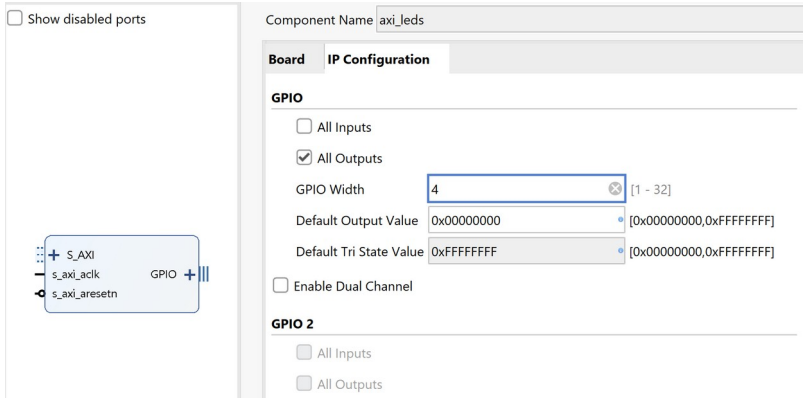
Part 1: Create a Vivado project with AXI GPIO input/output interfaces

Vivado Project

- Start by creating a **new project** in Vivado. Call it **HO3_AXI_GPIO**
- Make sure to select **VERILOG** as target language, and to select the **Pynq-Z2** as target board

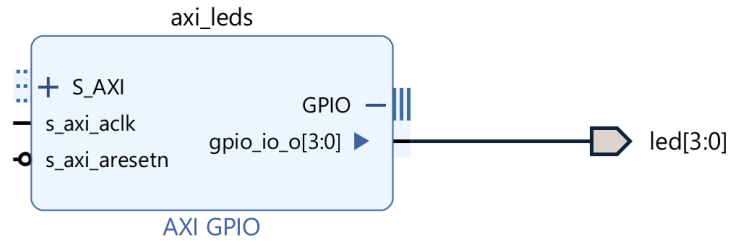
- Create a new block design. Add a **ZYNQ7 Processing System** block. Click on **Run Block Automation** to apply board presets (DRR and FIXED_IO will be enabled)

- Add an IP block of type **AXI GPIO**. Change its name to **axi_leds** (under Block Properties)
- We will use this AXI GPIO to interface the PS with the 4 LEDs in the Pynq-Z2
- Double-click on the **axi_leds** block to configure it:
  - Under IP Configuration, select “All Outputs” and change the GPIO width to 4
  - Why these values? LEDs are output interfaces, and each LED can be driven by 1 bit (so all we need are 4 bits in total)



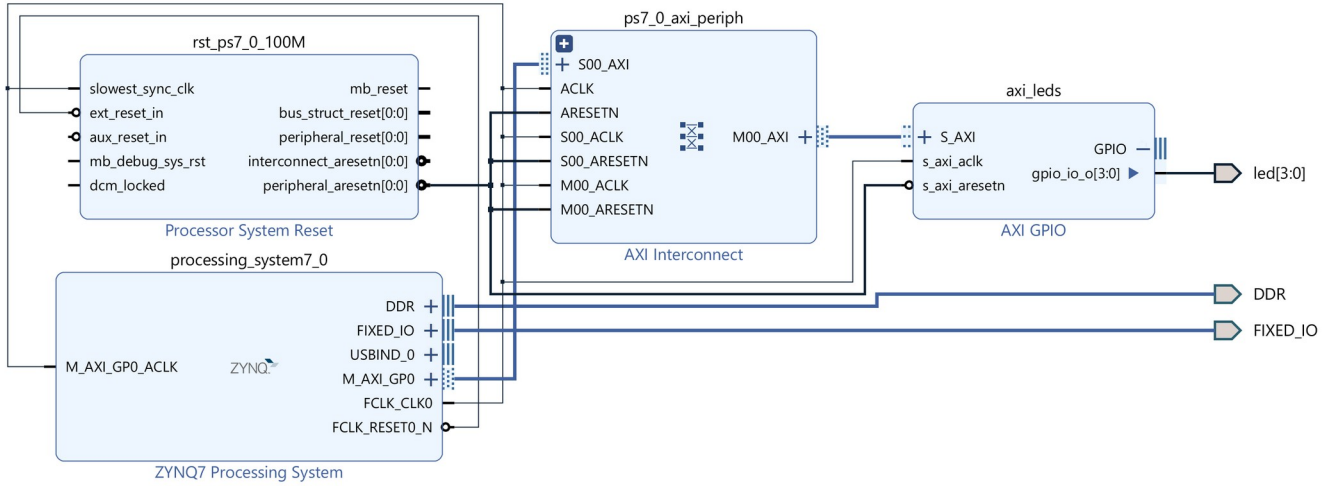
Vivado Project

- Make the output port of the AXI GPIO external
- Rename the interface to led

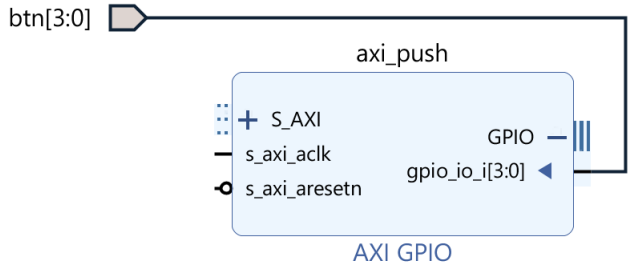


- Click now on **Run Connection Automation** so that Vivado makes the necessary connections
- What changes?

1. Vivado adds an **AXI Interconnect** block
    1. The PS can now access this block as Master, using port M_AXI_GP0 which is now enabled
    2. The AXI Interconnect can in turn access the AXI_GPIO block as Master
2. Vivado adds a **Processor System Reset** block
    - This block takes care of resetting all peripherals



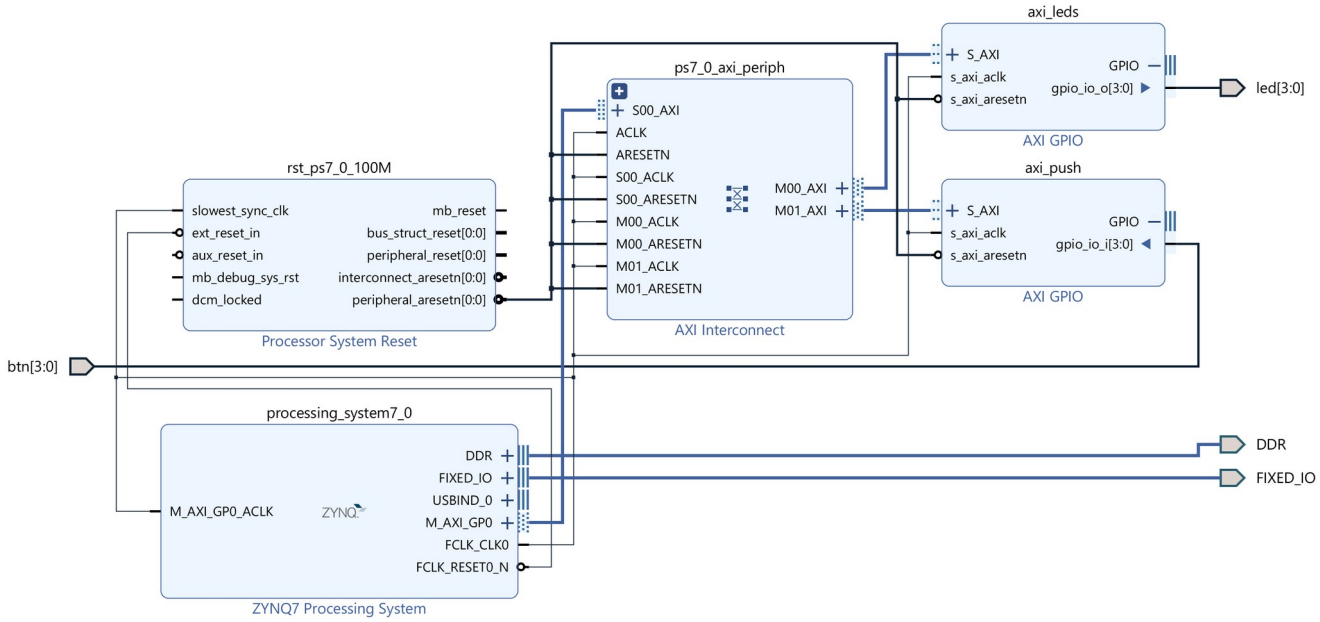
Vivado Project



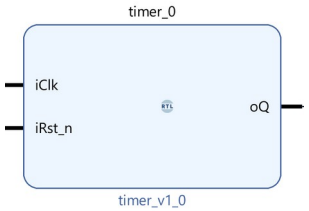
- Add a second **AXI GPIO** block. Rename it to **axi_push**, and configure it to “All Inputs” and GPIO Width equal to 4 bits
- Rename the output of this module to **btn**

- Click again on **Run Connection Automation** so that Vivado makes the necessary connections
- What changes?

1. Vivado adds a second master port to the **AXI Interconnect** block
    1. From the same port M_AXI_GP0, the PS can now access two interfaces (push buttons and LEDs)

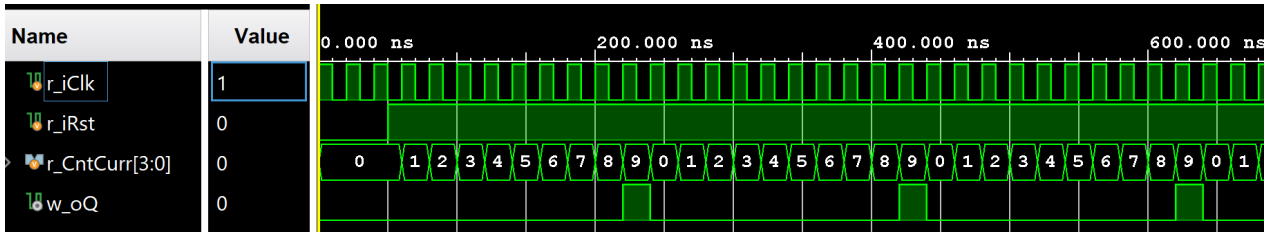


Vivado Project

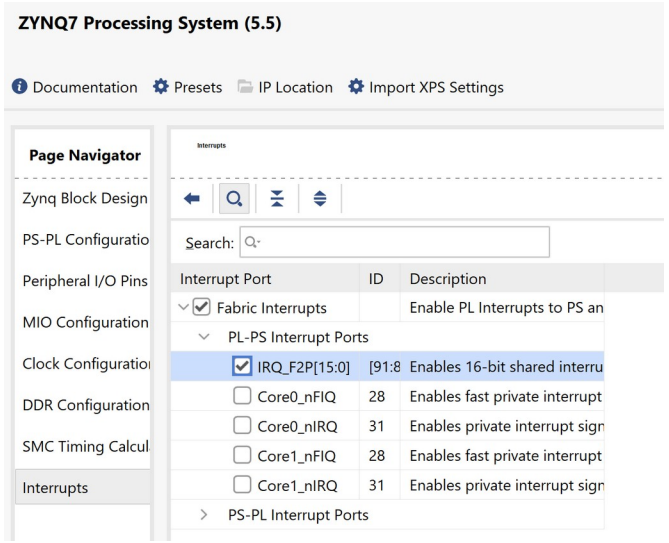


- Download from Toledo > EMBED Documents > Hands-on Materials > Week 2 the file **timer.v**
- This is a configurable time which we will use to generate interrupts signals at certain time intervals
- Add the module to the block design
- Double-click on the block to configure it

1. Set the clock frequency at **100 MHz**
   (so it matches FCLK_CLK0 provided by the PS)
2. Set it to generate **2 interrupts per second**



- Double-click now on the ZYNQ7 PS block to enable the interrupt feature
- Tick the boxes **Fabric Interrupts** and **IRQ_F2P**
- A new port IRQ_F2P will appear on the block



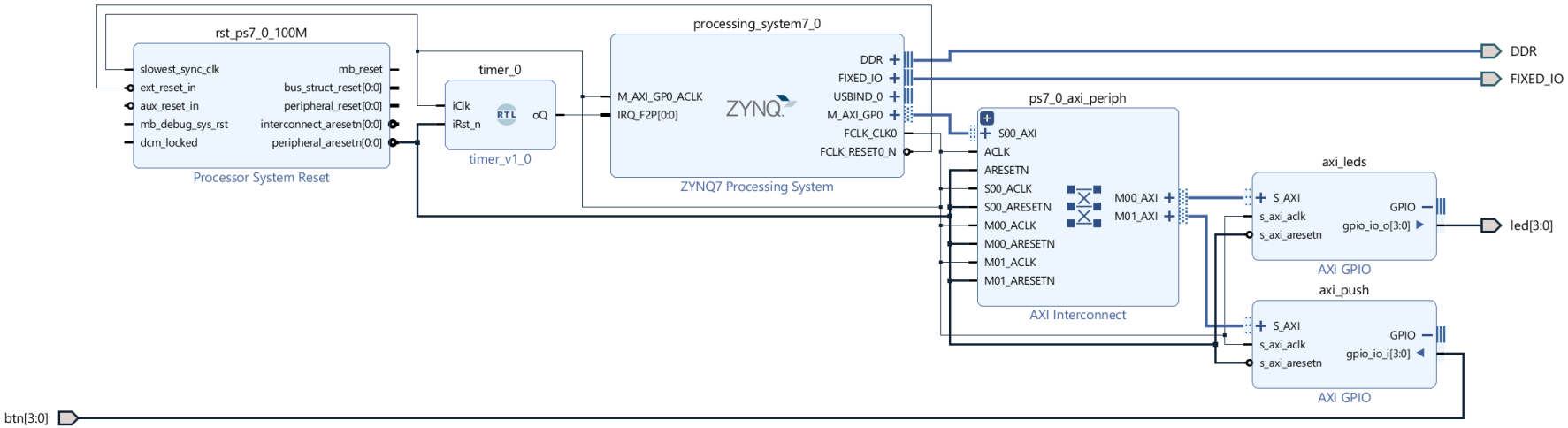
Vivado Project

- Connect the timer module as indicated below, to obtain the final block design
- Click on **Validate Block Design** to make sure all connections are correct
- Next, add the **XDC** file and **uncomment** the lines corresponding to the *leds* and *push* buttons
- Click on **Generate Bitstream** and export the **Hardware XSA**
  - Remember to pick the options **Fixed** and **Include Bitstream**
  - **Save the file to a folder (e.g. HO3\vitis)**

timer iClk → FCLK_CLK0 in ZYNQ7 block

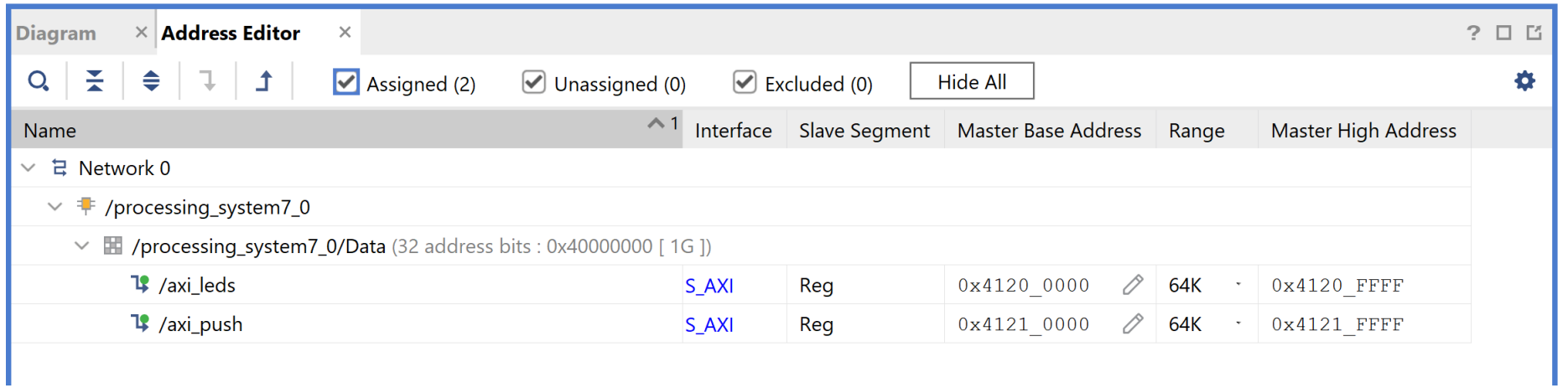
timer iRst_n → peripheral_areset_n in Processor System Reset (**active low reset**!!!)

timer oQ → IRQ_F2P in ZYNQ7 block



Vivado Project

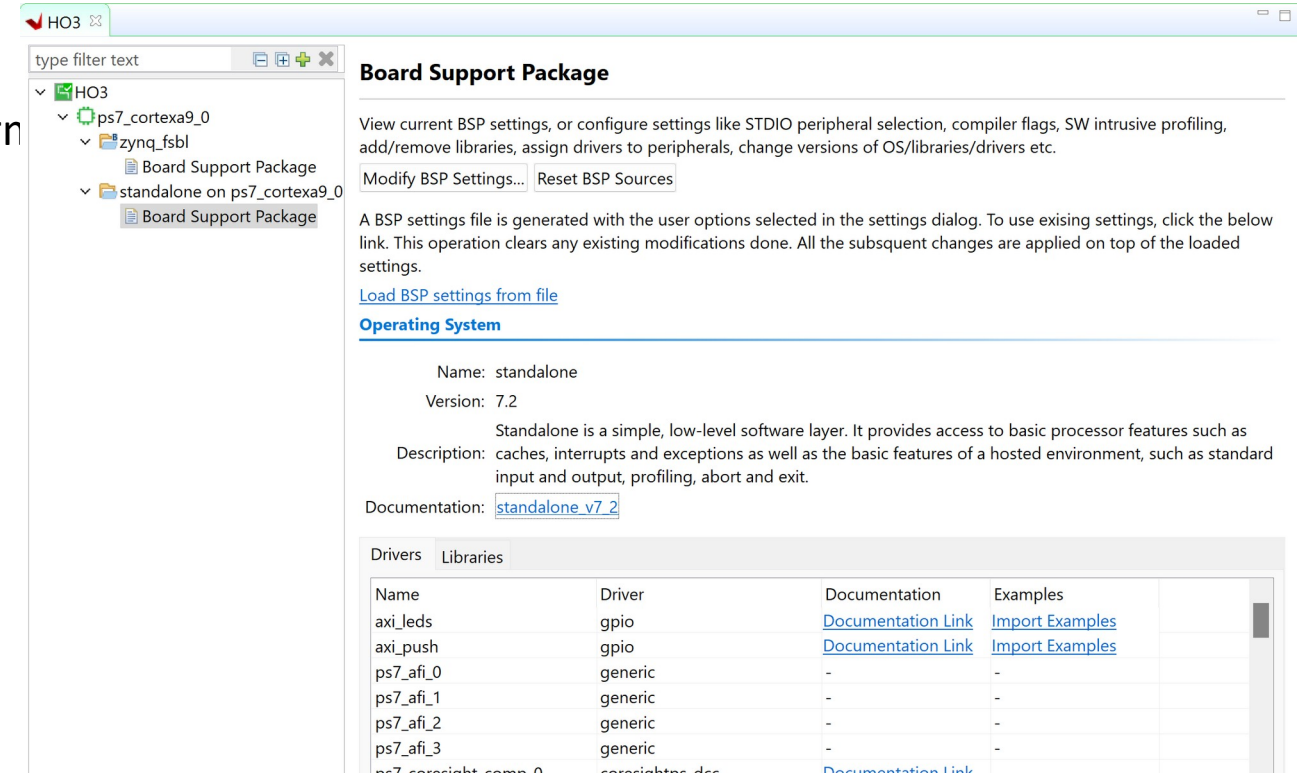
- Note that, as we add AXI interfaces to our Block Design, Vivado creates an **address map** for them
- You can check this by clicking on the **Address Editor** tab
- These addresses are the ones we will later use in the PS (aka Vitis) to access the two GPIO peripherals (leds and push buttons)



Part 2: Create a Vitis project based on polling

Vitis Project – Polling

- Launch the **Vitis 2024.1** , select a path for the Vitis workspace (for instance **HO3\vitis**)
- Click on **Create Platform Project**, give it the name HO3, import the XSA created in Vivado and build it
- Click on the **Board Support Package** to see the drivers section for associated to the platform
- We can see that Vitis has added software drivers to control our axi_gpio modules
- For each of them, you can access the **Documentation** as well as download **Examples**
- Click on Import Examples for the axi_leds
- Select the **xgpio_example**
- It will be added as new application in Vitis



Vitis Project – Polling

- Press CTRL+Click on the line file **`#include "xparameters.h"`**
- Here you can find some definitions and macros related to the HW Platform
- Among them, the definitions related to our AXI GPIO interfaces are of interest
- Note that there are 2 such interfaces
- The AXI_LEDS is mapped to address 0x4120_000
- The AXI_PUSH is mapped to address 0x4121_000
- These are the exact same addresses as defined in Vivado

```
221 /******************************************************************/
222
223 /* Definitions for driver GPIO */
224 #define XPAR_XGPIO_NUM_INSTANCES 2
225
226 /* Definitions for peripheral AXI_LEDS */
227 #define XPAR_AXI_LEDS_BASEADDR 0x41200000
228 #define XPAR_AXI_LEDS_HIGHADDR 0x4120FFFF
229 #define XPAR_AXI_LEDS_DEVICE_ID 0
230 #define XPAR_AXI_LEDS_INTERRUPT_PRESENT 0
231 #define XPAR_AXI_LEDS_IS_DUAL 0
232
233
234 /* Definitions for peripheral AXI_PUSH */
235 #define XPAR_AXI_PUSH_BASEADDR 0x41210000
236 #define XPAR_AXI_PUSH_HIGHADDR 0x4121FFFF
237 #define XPAR_AXI_PUSH_DEVICE_ID 1
238 #define XPAR_AXI_PUSH_INTERRUPT_PRESENT 0
239 #define XPAR_AXI_PUSH_IS_DUAL 0
240
241
242 /******************************************************************/
```

- The macros in the **`"xparameters.h"`** file are often used in the Vitis examples

Vitis Project – Polling

- Build the xgpio_example application and program it in the board
  - Remember: before the first programming, you should select **Run > Run Configurations** and double-click on System Project Debug to create a new configuration
- The LED0 should start blinking
- Study the C code to understand how the GPIO driver works
- You can consult:
  1. The Documentation Link provided in the Board Support Package (previous slide)
  2. The AXI GPIO Product Guide: https://docs.xilinx.com/v/u/en-US/pg144-axi-gpio
- Check how the C functions map to the configuration registers of the AXI GPIO Interface
- In the Vitis editor, press CTRL+Click to navigate to function declarations or parameters

```
/* Set the direction for all signals as inputs except the LED output */
XGpio_SetDataDirection(&Gpio, LED_CHANNEL, ~LED);
```

```
void XGpio_SetDataDirection ( XGpio *    InstancePtr,
                              unsigned   Channel,
                              u32        DirectionMask
                            )
```

Set the input/output direction of all discrete signals for the specified GPIO channel.

Parameters

- **InstancePtr** is a pointer to an **XGpio** instance to be worked on.
- **Channel** contains the channel of the GPIO (1 or 2) to operate on.
- **DirectionMask** is a bitmask specifying which discretes are input and which are output. Bits set to 0 are output and bits set to 1 are input.

Returns

None.

Table 2-7: **AXI GPIO Three-State Register Description**

Bits	Field Name	Access Type	Reset Value	Description
[GPIOx_Width-1 :0]	GPIOx_TRI	Read/ Write	GPIO: Default Tri State Value GPIO2: Default Tri State Value	AXI GPIO 3-State Control Register. Each I/O pin of the AXI GPIO is individually programmable as an input or output. For each of the bits: 0 = I/O pin configured as output. 1 = I/O pin configured as input.

Vitis Project – Polling



Once you understand the API, extend the design such that:

- LEDs 0 and 1 turn ON when the button below them is pressed, and are OFF otherwise
- LEDs 2 and 3 turn OFF when the button below them is pressed, and are ON otherwise

- Some hints:

1. You will need to create two GPIO instances, and initialize them and configure them correctly
2. To access the PUSH button AXI interface (of type input) you can use the `XGpio_DiscreteRead` function
3. Inverting a bit can be done by XORing with '1' →
4. Use the *printf()* function or debugging mode to monitor variables
5. Check the xparameters.h file, which contains macros for the processor's address space and the device IDs

```
XGpio Gpio_LED; /* The Instance of the GPIO Driver */
XGpio Gpio_PUSH; /* The Instance of the GPIO Driver */
```

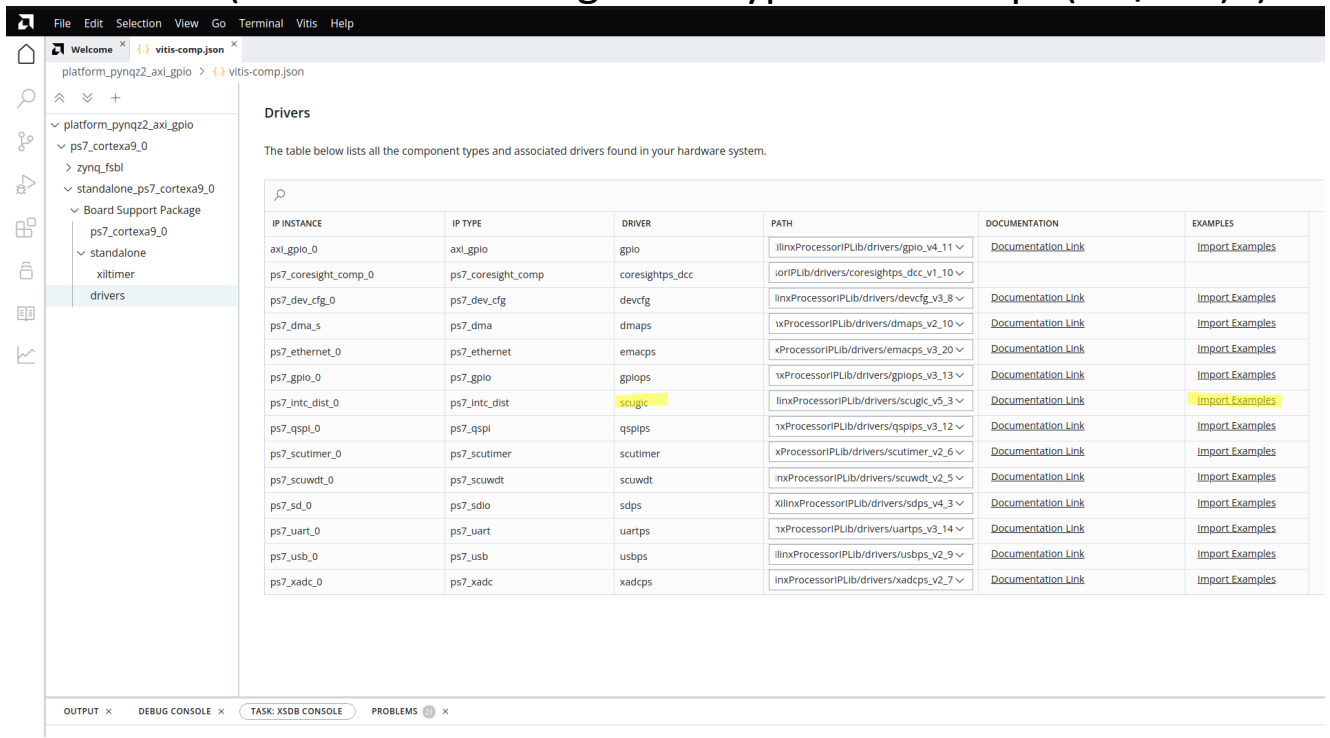
- Recall this example uses POLLING , so all functionality is placed inside a while() loop

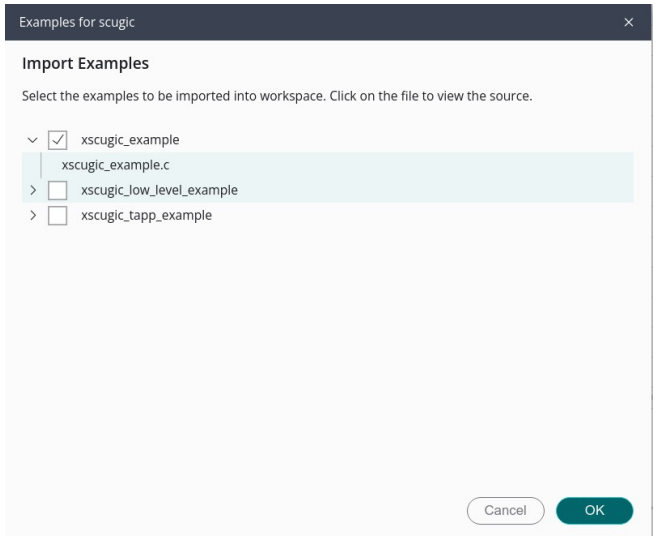
Part 3: Create a Vitis project based on interrupts

Vitis Project – Interrupts

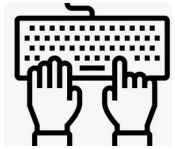
UG585 (TRM)
UG643 (Interrupt wrapper API)

- In the same Vitis project, go to the drivers section of the platform BSP (Board Support Package)
- Click import examples from the scugic driver.
- The code uses the ps7_scugic driver for the Generic Interrupt Controller
- Check the documentation resources to understand what each API function does, change to code to make it work. (BASEADDR of scugic ok? Type of Interrupt (SPI/SGI)?)





Vitis Project – Interrupts



Edit the hello_world.c to obtain the previous functionality using interrupts instead of polling, that is:

- LEDs 0 and 1 turn ON when the button below them is pressed, and are OFF otherwise
- LEDs 2 and 3 turn OFF when the button below them is pressed, and are ON otherwise

- Since our timer creates only 2 interrupts per second, you will observe a delay between the pressing of a button and the switching ON/OFF of the LEDs
- If there is no delay, maybe you are still programming the polling example. Double-check!
- Increasing the number of interrupts generated by the timer would solve the issue