AXI GPIO INTERRUPT LED

Introduction

The Xilinx® LogiCORE™ IP AXI General Purpose Input/Output (GPIO) core provides a general purpose input/output interface to the AXI interface. This 32-bit soft Intellectual Property (IP) core is designed to interface with the AXI4-Lite interface.

Features

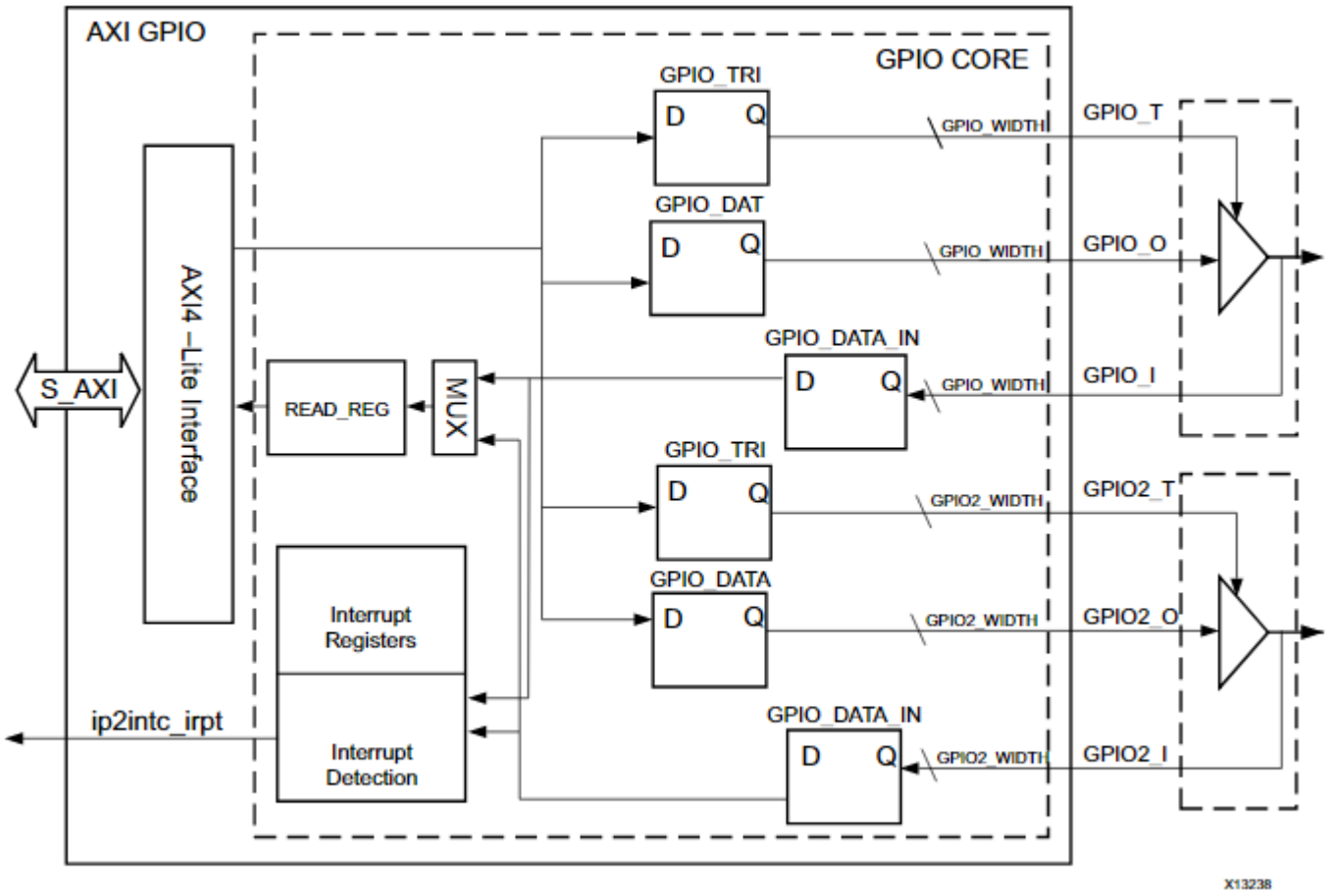
- Supports the AXI4-Lite interface specification
- Supports configurable single or dual GPIO channel(s)
- Supports configurable channel width for GPIO pins from 1 to 32 bits
- Supports dynamic programming of each GPIO bit as input or output
- Supports individual configuration of each channel
- Supports independent reset values for each bit of all registers
- Supports optional interrupt request generation

Functional Description

The AXI GPIO design provides a general purpose input/output interface to an AXI4-Lite interface. The AXI GPIO can be configured as either a single or a dual-channel device. The width of each channel is independently configurable.

The ports are configured dynamically for input or output by enabling or disabling the 3-state buffer. The channels can be configured to generate an interrupt when a transition on any of their inputs occurs.

The top-level block diagram of AXI GPIO core is shown in Figure 1-1.



Set GPIO_I(INPUT), GPIO_T set 1.
Set GPIO_O(OUTPUT), GPIO_T set 0.

Table 2-4: **Registers**

Address Space Offset[3]	Register Name	Access Type	Default Value	Description
0x0000	GPIO_DATA	R/W	0x0	Channel 1 AXI GPIO Data Register.
0x0004	GPIO_TRI	R/W	0x0	Channel 1 AXI GPIO 3-state Control Register.
0x0008	GPIO2_DATA	R/W	0x0	Channel 2 AXI GPIO Data Register.
0x000C	GPIO2_TRI	R/W	0x0	Channel 2 AXI GPIO 3-state Control.
0x011C	GIER[1]	R/W	0x0	Global Interrupt Enable Register.
0x0128	IP IER[1]	R/W	0x0	IP Interrupt Enable Register (IP IER).
0x0120	IP ISR[1]	R/TOW[2]	0x0	IP Interrupt Status Register.

AXI GPIO Data Register (GPIOx_DATA)

The AXI GPIO data register is used to read the general purpose input ports and write to the general purpose output ports. When a port is configured as input, writing to the AXI GPIO data register has no effect.

There are two GPIO data registers (GPIO_DATA and GPIO2_DATA), one corresponding to each channel. The channel 1 data register (GPIO_DATA) is always present; the channel 2 data register (GPIO2_DATA) is present only if the core is configured for dual channel (**Enable Dual Channel = 1**).

The AXI GPIO Data Register is shown in Figure 2-1, and Table 2-6 details the functionality of this register.

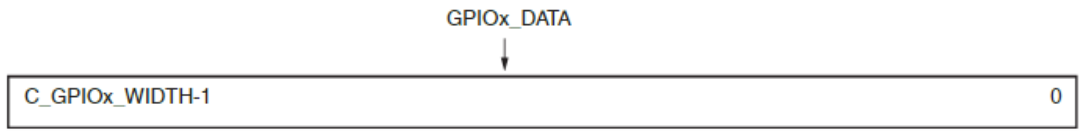


Table 2-6: **AXI GPIO Data Register Description**

Bits	Field Name	Access Type	Reset Value	Description
[GPIOx_Width-1 :0]	GPIOx_DATA	Read/Write	GPIO: Default Output Value GPIO2: Default Output Value	AXI GPIO Data Register. For each I/O bit programmed as input: • R: Reads value on the input pin. • W: No effect. For each I/O bit programmed as output: • R: Reads to these bits always return zeros • W: Writes value to the corresponding AXI GPIO data register bit and output pin.

AXI GPIO 3-State Control Register (GPIOx_TRI)

The AXI GPIO 3-state control register is used to configure the ports dynamically as input or output. When a bit within this register is set, the corresponding I/O port is configured as an input port. When a bit is cleared, the corresponding I/O port is configured as an output port.

There are two AXI GPIO 3-state control registers (GPIO_TRI and GPIO2_TRI), one corresponding to each channel. The channel 2 3-state control register (GPIO2_TRI) is present only if the core is configured for dual channel **Enable Dual Channel = 1**.

The AXI GPIO 3-state control register is shown in Figure 2-2; the register function is described in Table 2-7.

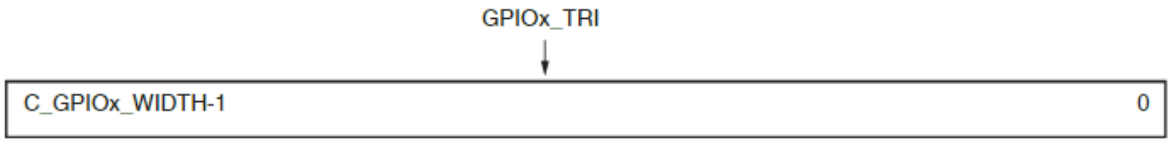


Figure 2-2: **AXI GPIO Three-State Register**

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.

Interrupts

The AXI GPIO core can be configured under the control of the Enable Interrupt parameter to generate a level interrupt when a transition occurs in any of the channel inputs. The GPIO interface module includes interrupt detection logic to identify any transition on channel inputs. When a transition is detected, it is indicated to the Interrupt Controller module. The Interrupt Controller module implements the necessary registers to enable and maintain the status of the interrupts.

To support interrupt capability for channels, the Interrupt Controller module implements the following registers:

- Global Interrupt Enable Register (GIER) – Provides the master enable/disable for the interrupt output to the processor or Interrupt Controller. See Global Interrupt Enable Register (GIER) for more details.
- IP Interrupt Enable Register (IPIER) – Implements the independent interrupt enable bit for each channel. See IP Interrupt Enable (IPIER) and IP Status Registers (IPISR) for more details.
- IP Interrupt Status Register (IPISR) – Implements the independent interrupt status bit for each channel. The IP ISR provides Read and Toggle-On-Write access. The Toggle-On-Write mechanism allows interrupt service routines to clear one or more ISR bits using a single write transaction. The IP ISR can also be manually set to generate an interrupt for testing purposes. See IP Interrupt Enable (IPIER) and IP Status Registers (IPISR)for more details.

Global Interrupt Enable Register (GIER)

The Global Interrupt Enable register provides the master enable/disable for the interrupt output to the processor. This is a single-bit read/write register as shown in Figure 2-3. This register is valid only if the Enable Interrupt parameter is set.

Note: Because this is the master bit to control interrupt generation, it must be set to generate interrupts, even if the interrupts are enabled in the IP Interrupt Enable Register (IP IER). The bit definition for Global Interrupt Enable Register is given in Table 2-8.

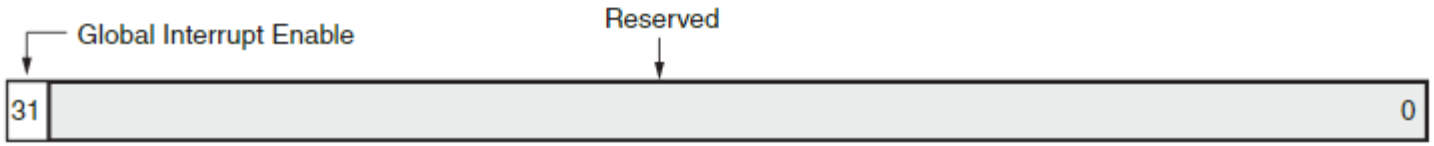


Figure 2-3: **Global Interrupt Enable Register**

Table 2-8: **Global Interrupt Enable Register Description**

Bits	Name	Core Access	Reset Value	Description
31	Global Interrupt Enable	Read/Write	0	Master enable for the device interrupt output to the system interrupt controller: 0 = Disabled 1 = Enabled
30 – 0	Reserved	N/A	0	Reserved. Set to zeros on a read.

***Programming Sequence

The following steps are helpful in accessing the AXI GPIO core.

For input ports when the Interrupt is enabled, follow these steps:

1.

Configure the port as input by writing the corresponding bit in **GPIOx_TRI** register with **the value of 1**.

2.

Enable the channel interrupt by setting the corresponding bit in the IP Interrupt Enable Register; also enable the global interrupt, by **setting bit 31 of the Global Interrupt Register to 1.**

3.

When an interrupt is received, read the corresponding bit in the GPIOx_DATA register. **Clear the status in the IP Interrupt Status Register by writing the corresponding bit with the value of 1.**

For input ports when the Interrupt is not enabled, use the following steps:

1.

Configure the port as input by writing the corresponding bit in GPIOx_TRI register with the value of 1.

2.

Read the corresponding bit in GPIOx_DATA register.

For output ports, use the following steps:

1.

Configure the port as output by writing the corresponding bit in GPIOx_TRI register with a value of 0.

2.

Write the corresponding bit in GPIOx_DATA register.

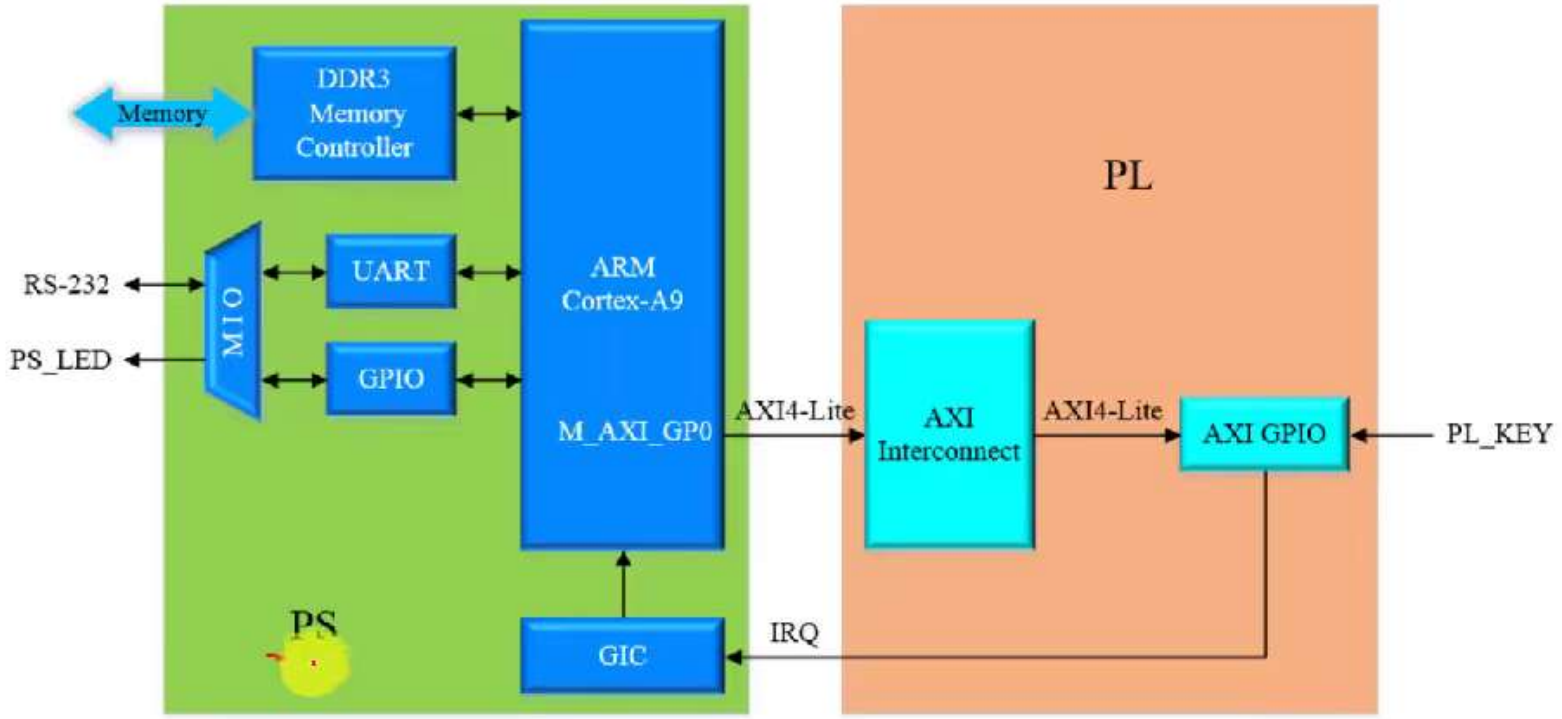
Table 2-3: **AXI GPIO Signal Description**

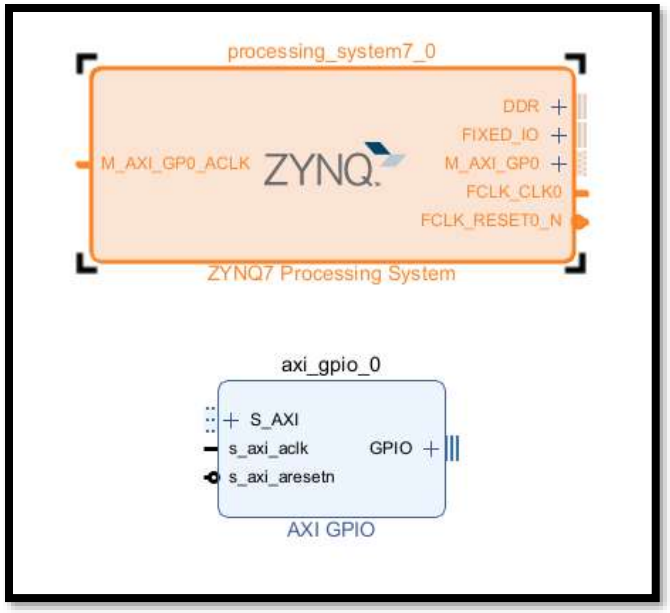
Signal Name	Interface	I/O	Initial State	Description
s_axi_aclk	Clock	I		AXI Clock.
s_axi_aresetn	Reset	I		AXI Reset, active-Low.
s_axi_*	S_AXI	NA	-	AXI4-Lite Slave Interface signals. See Appendix A of the *Vivado AXI Reference Guide* (UG1037) [Ref 3] for AXI4, AXI4-Lite and AXI Stream Signals
ip2intc_irpt	System	O	0	AXI GPIO Interrupt. active-High, level sensitive signal.
gpio_io_i (1)(3)	GPIO	I		Channel 1 general purpose input pins. Width of this port is configurable based on **GPIO Width**.
gpio_io_o(2)(3)(4)	GPIO	O	0	Channel 1 general purpose output pins. Width of this port is configurable based on **GPIO Width**.
gpio_io_t(4)	GPIO	O	1	Channel 1 general purpose 3-state pins. Width of this port is configurable based on **GPIO Width**.
gpio2_io_i(1)(3)	GPIO	I		Channel 2 general purpose input pins. Width of this port is configurable based on **GPIO2 Width**.
gpio2_io_o(2)(3)(4)	GPIO	O	0	Channel 2 general purpose output pins. Width of this port is configurable based on **GPIO2 Width**.
gpio2_io_t(4)	GPIO	O	1	Channel 2 general purpose 3-state pins Width of this port is configurable based on **GPIO2 Width**.

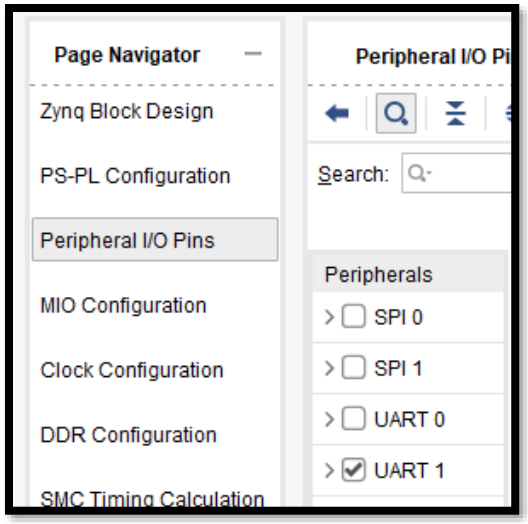
Lab purpose

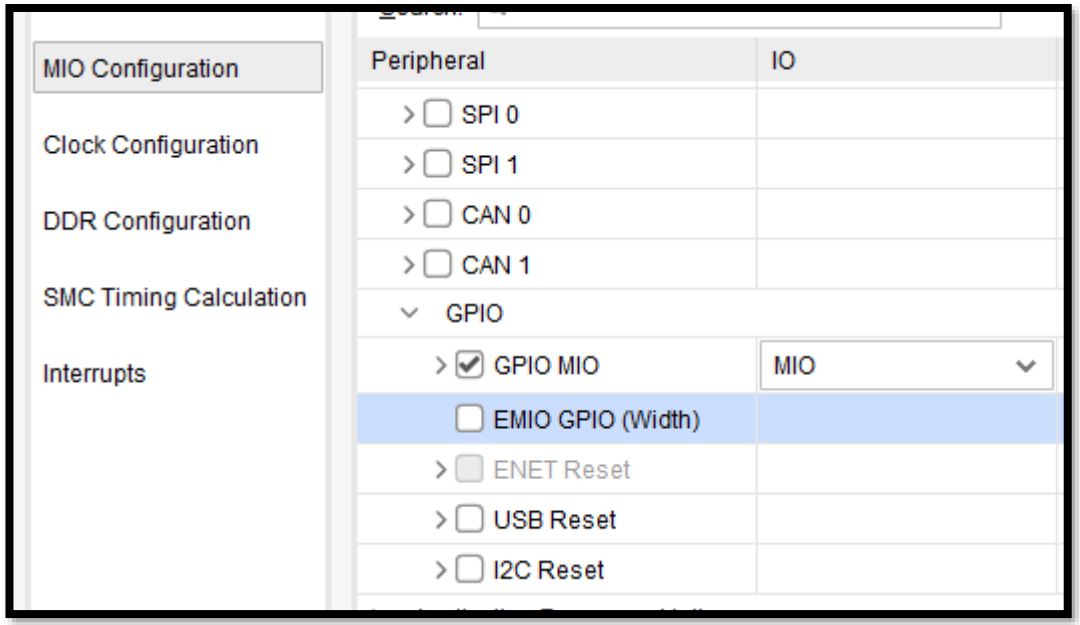
- Import AXI GPIO(IP), Using interrupts, the PL side controls the LED on the PS side.

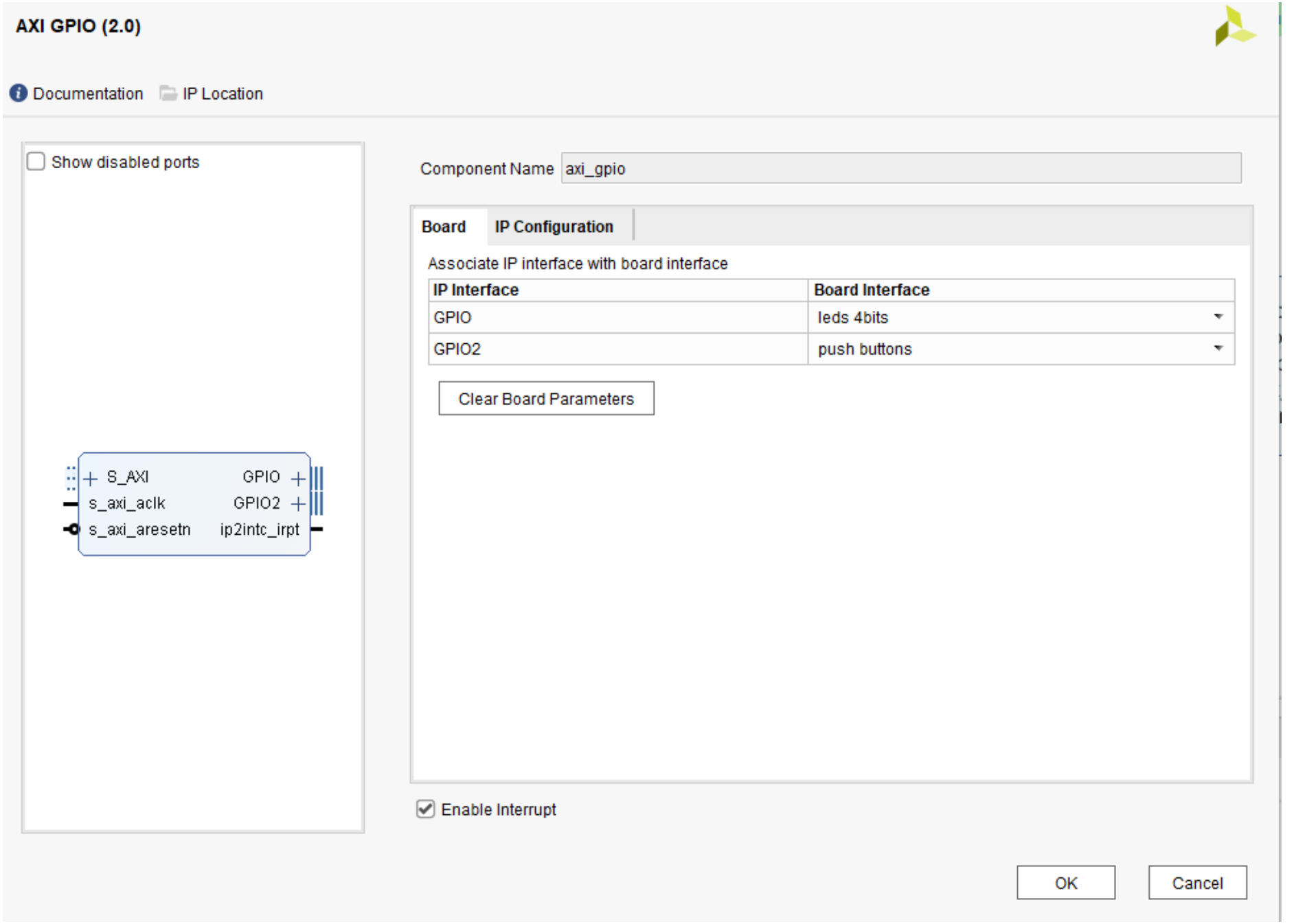
System Block diagram

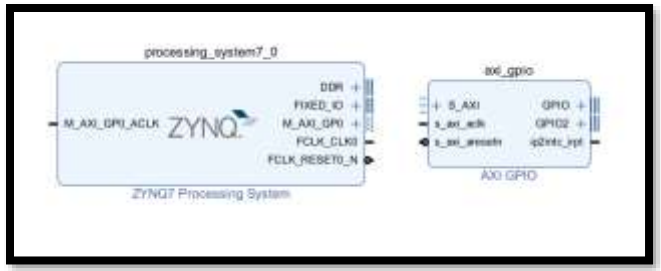




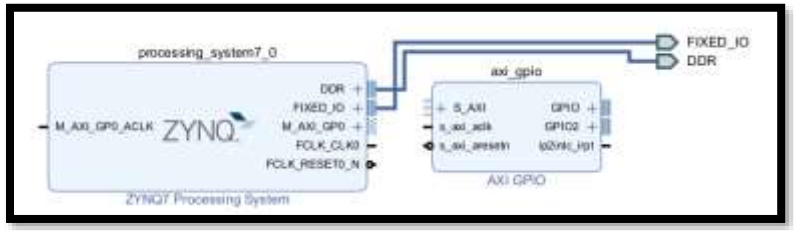




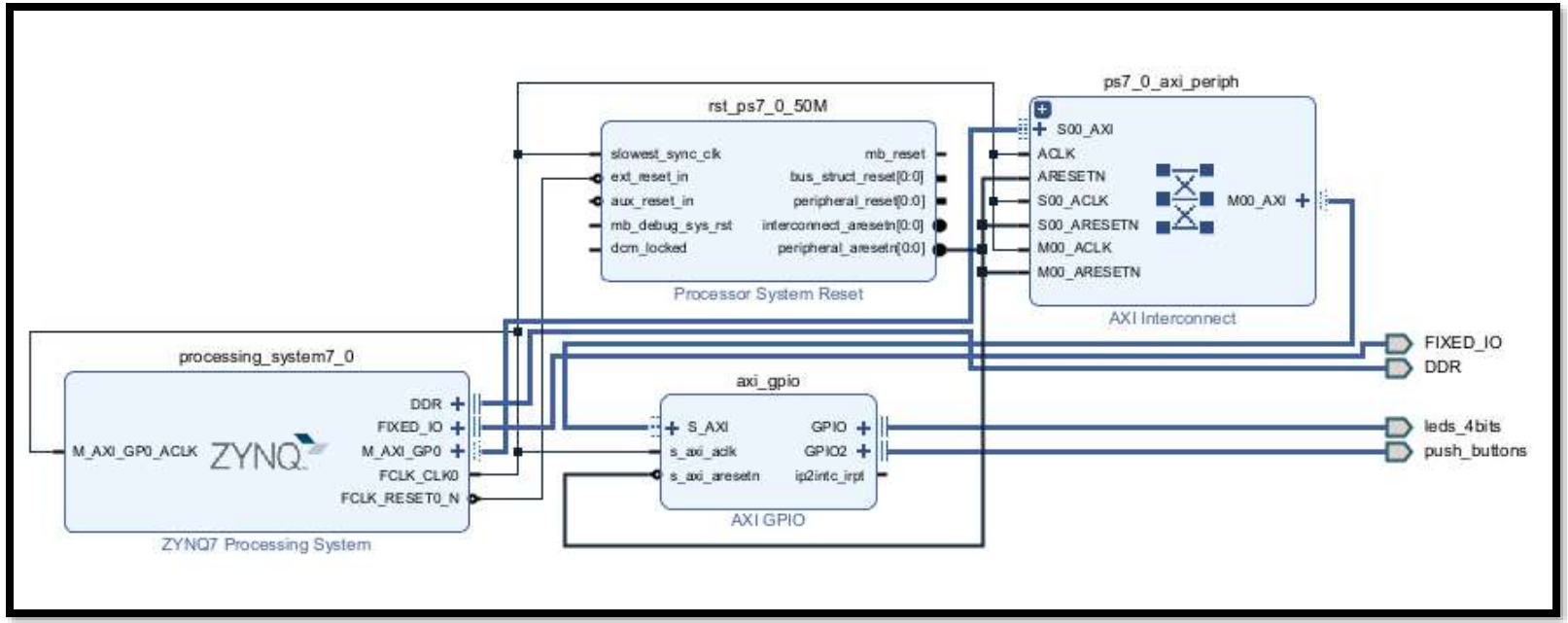


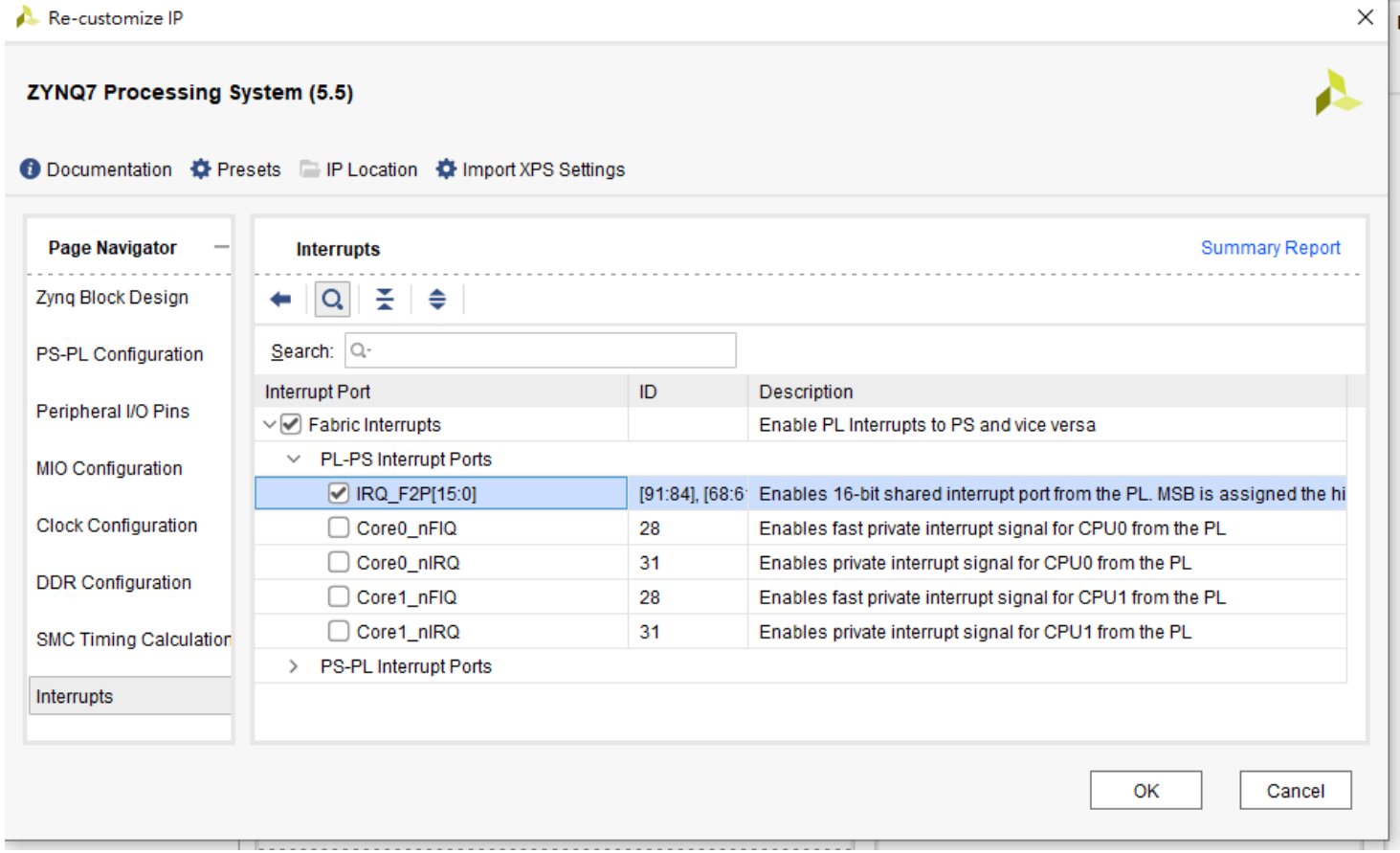


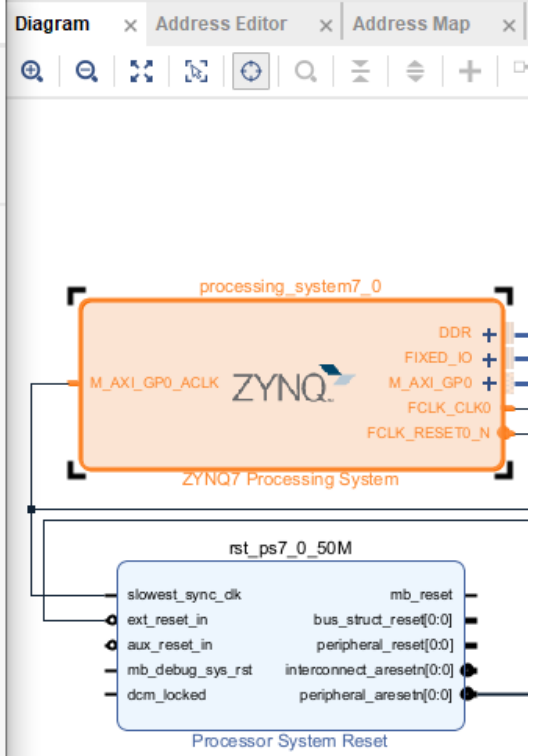
Designer Assistance available. Run Block Automation



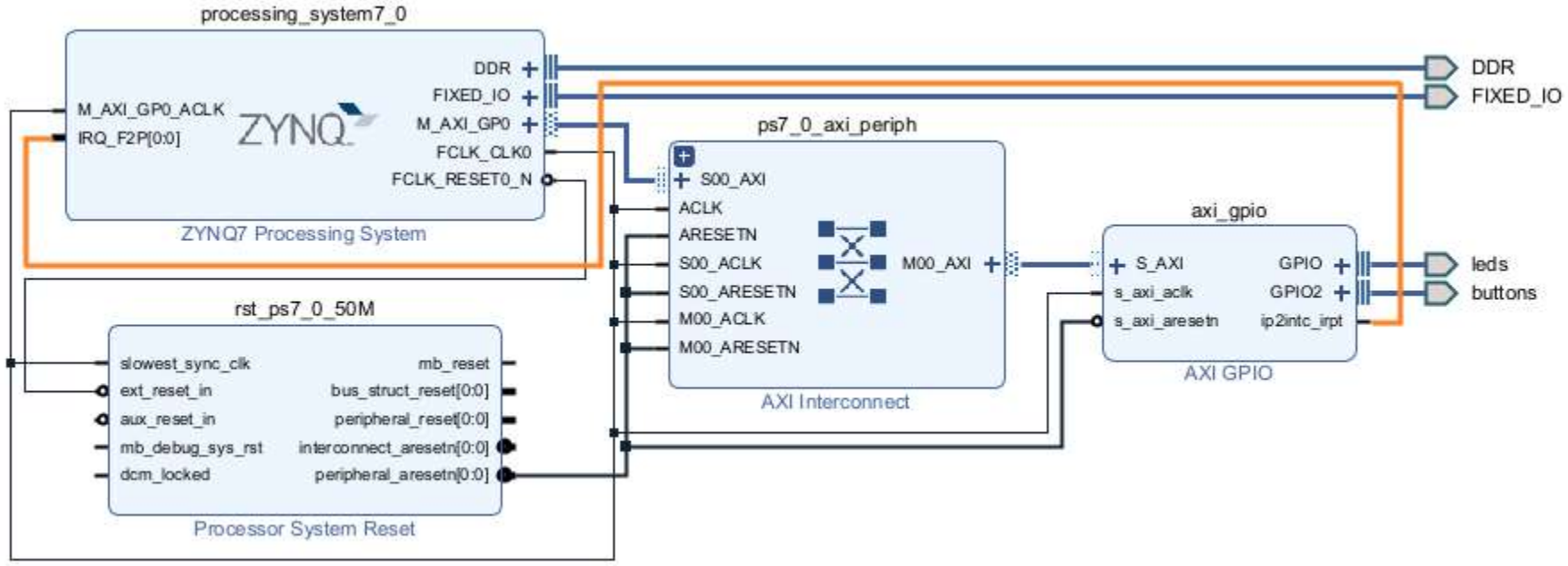
Designer Assistance available. Run Connection Automation



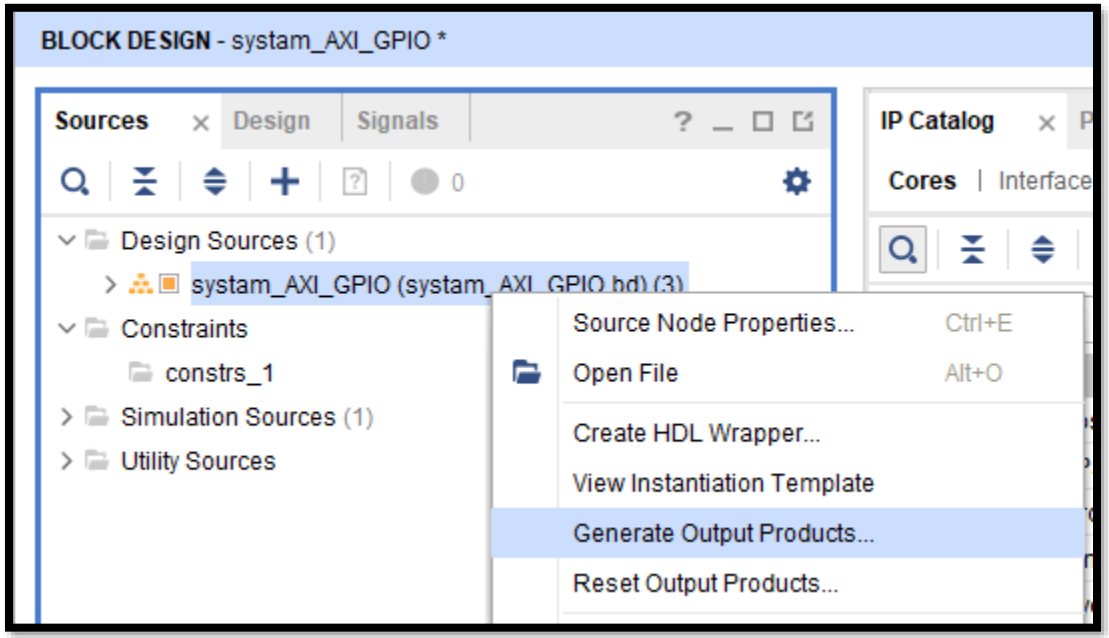


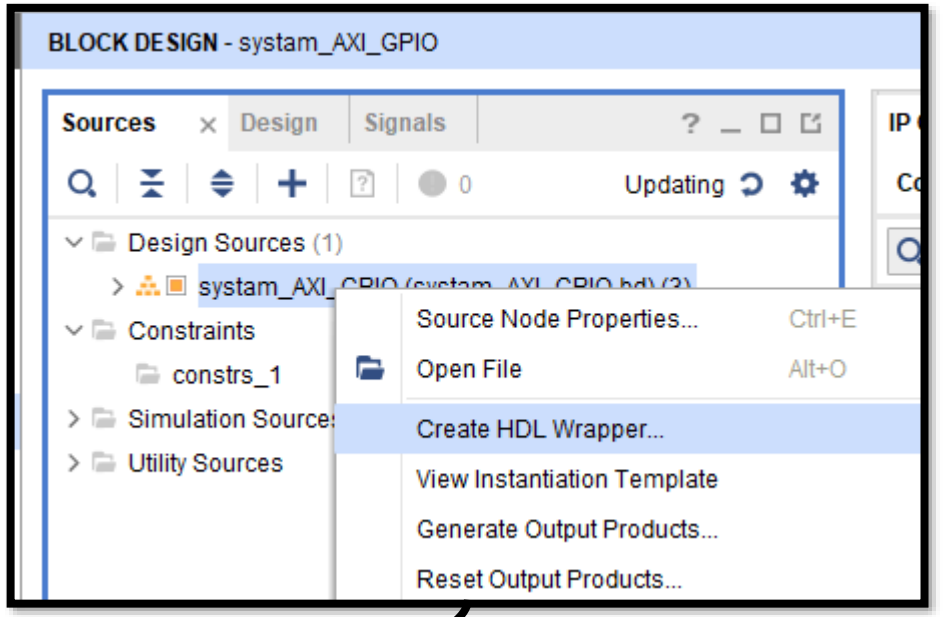


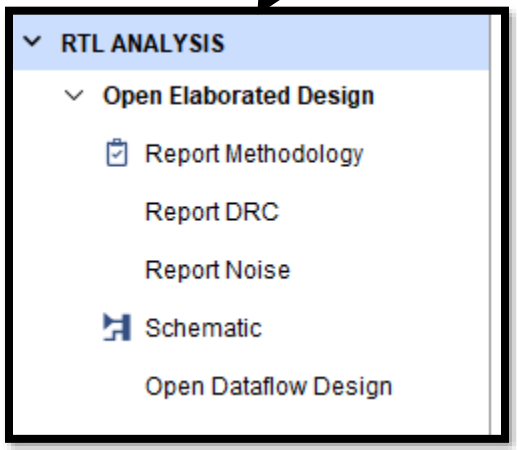
Meansure the PS can receive PL interrupt single.

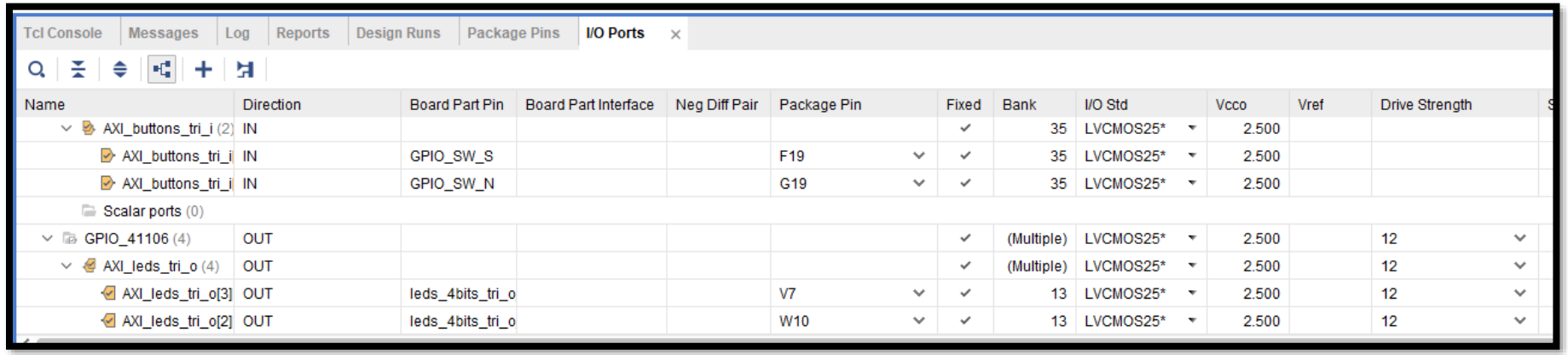


- Connecting the Port IRQ_F2P(zynq processing) to Ip2intc_irpt(AXI GPIO). Like figure orange line.

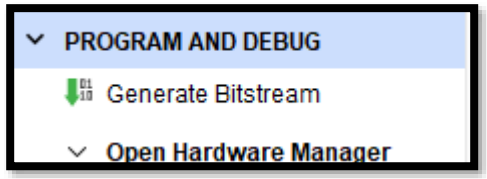


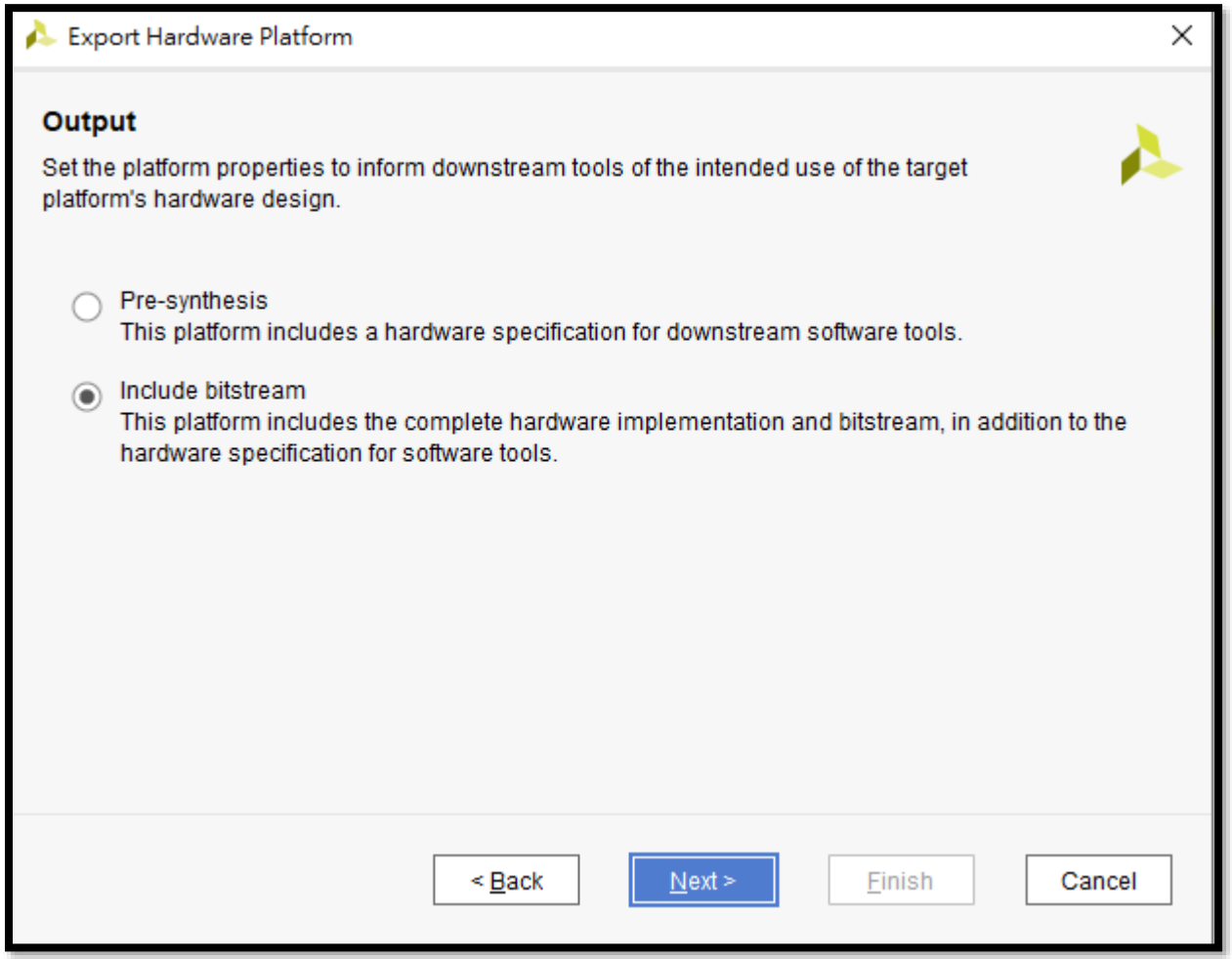






Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength
AXI_buttons_tri_i (2)	IN					✓	35	LVCMOS25*	2.500		
AXI_buttons_tri_i	IN	GPIO_SW_S			F19	✓	35	LVCMOS25*	2.500		
AXI_buttons_tri_i	IN	GPIO_SW_N			G19	✓	35	LVCMOS25*	2.500		
Scalar ports (0)											
GPIO_41106 (4)	OUT					✓	(Multiple)	LVCMOS25*	2.500		12
AXI_leds_tri_o (4)	OUT					✓	(Multiple)	LVCMOS25*	2.500		12
AXI_leds_tri_o[3]	OUT	leds_4bits_tri_o			V7	✓	13	LVCMOS25*	2.500		12
AXI_leds_tri_o[2]	OUT	leds_4bits_tri_o			W10	✓	13	LVCMOS25*	2.500		12





Shared Peripheral Interrupts (SPI)

A group of approximately 60 interrupts from various modules can be routed to one or both of the CPUs or the PL. The interrupt controller manages the prioritization and reception of these interrupts for the CPUs.

Except for IRQ #61 through #68 and #84 through #91, all interrupt sensitivity types are fixed by the requesting sources and cannot be changed. The GIC must be programmed to

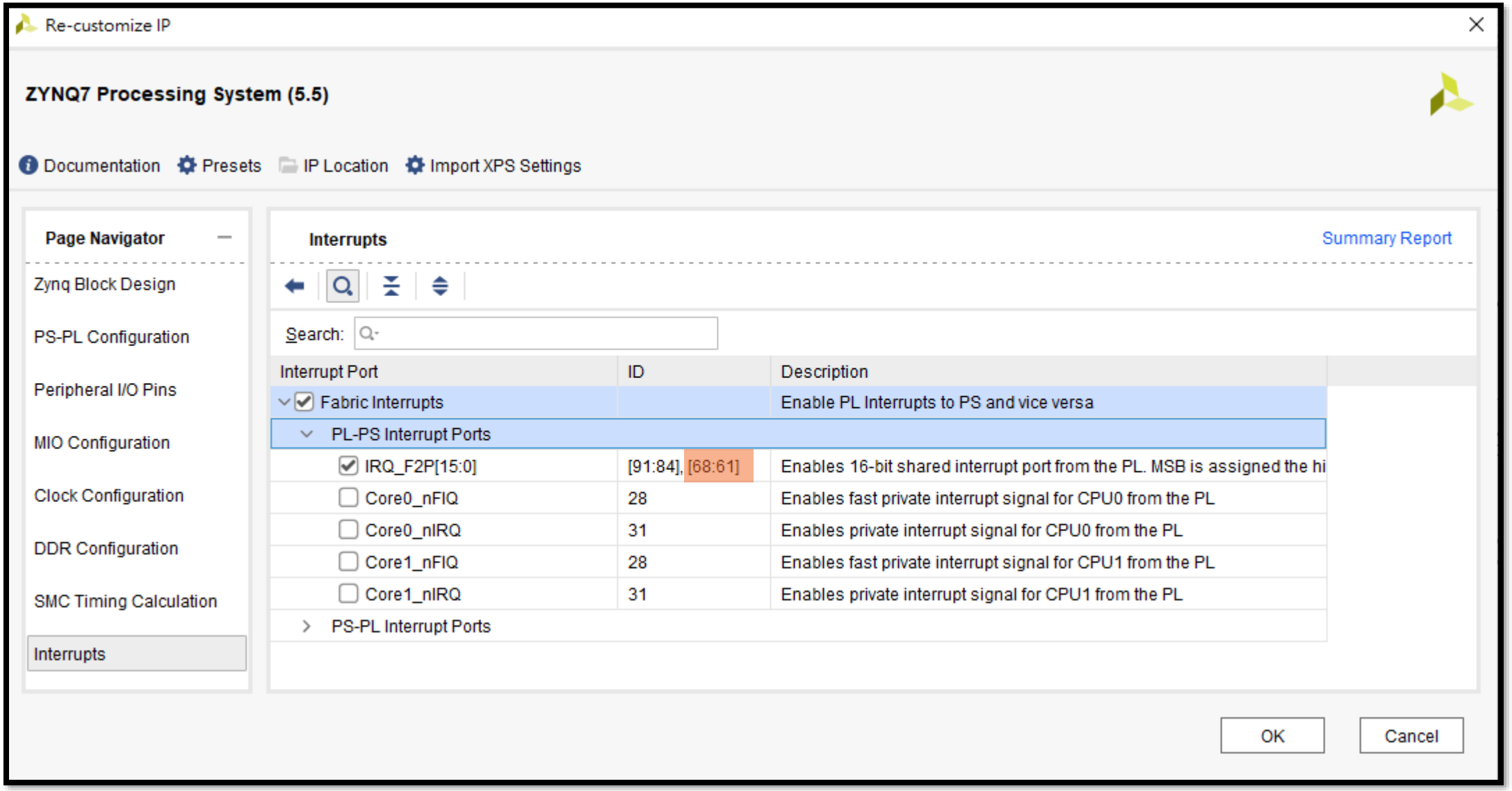
PL to PS have 16's interrupt

8's interrupt ,interrupt from PL

Just using one interrupt , so the ID number is 61

Source	Interrupt Name	IRQ ID#	Status Bits (mpcore Registers)	Required Type	PS-PL Signal Name	I/O
IOP	GPIO	52	spi_status_0[20]	High level	IRQP2F[16]	Output
	USB 0	53	spi_status_0[21]	High level	IRQP2F[15]	Output
	Ethernet 0	54	spi_status_0[22]	High level	IRQP2F[14]	Output
	Ethernet 0 Wake-up	55	spi_status_0[23]	Rising edge	IRQP2F[13]	Output
	SDIO 0	56	spi_status_0[24]	High level	IRQP2F[12]	Output
	I2C 0	57	spi_status_0[25]	High level	IRQP2F[11]	Output
	SPI 0	58	spi_status_0[26]	High level	IRQP2F[10]	Output
	UART 0	59	spi_status_0[27]	High level	IRQP2F[9]	Output
	CAN 0	60	spi_status_0[28]	High level	IRQP2F[8]	Output
PL	PL [2:0]	63:61	spi_status_0[31:29]	Rising edge/ High level	IRQF2P[2:0]	Input
	PL [7:3]	68:64	spi_status_1[4:0]	Rising edge/ High level	IRQF2P[7:3]	Input
Timer	TTC 1	71:69				
DMAC	DMAC[7:4]	75:72				
IOP	USB 1	76				
	Ethernet 1	77				
	Ethernet 1 Wake-up	78	spi_status_1[14]	Rising edge	IRQP2F[5]	Output
	SDIO 1	79	spi_status_1[15]	High level	IRQP2F[4]	Output
	I2C 1	80	spi_status_1[16]	High level	IRQP2F[3]	Output
	SPI 1	81	spi_status_1[17]	High level	IRQP2F[2]	Output
	UART 1	82	spi_status_1[18]	High level	IRQP2F[1]	Output
	CAN 1	83	spi_status_1[19]	High level	IRQP2F[0]	Output
PL	PL [15:8]	91:84	spi_status_1[27:20]	Rising edge/ High level	IRQF2P[15:8]	Input

8's interrupt ,interrupt from PL



Just using one interrupt , so the ID number is 61,refer last PG.

Code definition

- File name “AXI_GPIO_main.c”

```
#include "xparameters.h"
#include "xgpio.h"
#include "xil_exception.h"
#include <stdio.h>
#include "xscugic.h"
#include "xil_printf.h"
#include "xgpiops.h"
#include "sleep.h"
```

- Refer the Page 19 ,know the first interrupt pin is 61.
- Need the two define about AXI_GPIO and AXI_INTR

```
#define GPIO_DEVICE_ID      XPAR_XGPIOPS_0_DEVICE_ID            //ps gpio
#define INTC_DEVICE_ID      XPAR_SCUGIC_SINGLE_DEVICE_ID        //ps intr
#define GPIO_INTERRUPT_ID   XPAR_XGPIOPS_0_INTR                 //PS intr #52
#define INTC_HANDLER    XScuGic_InterruptHandler
#define AXI_GPIO_DEVICE_ID      XPAR_GPIO_0_DEVICE_ID           //axi gpio
#define INTC_GPIO_INTERRUPT_ID  XPAR_FABRIC_AXI_GPIO_IP2INTC_IRPT_INTR      //axi intr #61
#define INTC        XScuGic

#define gpio_input      0
#define gpio_output     1
//PS MIO
#define PS_LED1         10
#define PL_EMIO         54
#define Enable          1
#define disable         0
#define High            1
#define Low             0
#define LED_DELAY       1000000

// axi GPIO channel 1
#define GPIO_CHANNEL1       1

#define BUTTON_CHANNEL   2  /* Channel 1 of the GPIO Device */
#define LED_CHANNEL  1  /* Channel 2 of the GPIO Device */

#define LED1        0x01
#define LED2        0x02
#define LED3        0x04
#define LED4        0x08
#define LED_FULL    0x0F
#define BTN1        0x00
#define BTN2        0x01
```

Code definition

```
int SetupInterruptSystem(XScuGic *GicInstancePtr,XGpio *AXI_Gpio,u16 AXI_GpioIntrId);
void IntrHandler();


XScuGic Intc; /* The Instance of the Interrupt Controller Driver */
XScuGic_Config *IntcConfig; /* Instance of the interrupt controller */

XGpioPs Gpio; /* The Instance of the GPIO Driver */
XGpioPs_Config *ConfigPtr;

XGpio AXI_Gpio;
INTC Intc; /* The Instance of the Interrupt Controller Driver */
```

Head define the function, we need the pointer to point the AXI_GPIO

```c
int main(){
    xil_printf("AXI GPIO INTERRUPT TEST");


    /* Initialize the PS_GPIO driver. */
    ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
    XGpioPs_CfgInitialize(&Gpio, ConfigPtr,ConfigPtr->BaseAddr);
    xil_printf("Initialize the PS_GPIO driver Successfully\n\r");

    /* Initialize the AXI_GPIO driver. find the xgpio.h */
    XGpio_Initialize(&AXI_Gpio, AXI_GPIO_DEVICE_ID);
    xil_printf("Initialize the AXI_GPIO driver Successfully\n\r");

    //Set the direction for the PS_pin to be output and
    XGpioPs_SetDirectionPin(&Gpio, PS_LED1, gpio_output);
    XGpioPs_SetOutputEnablePin(&Gpio, PS_LED1, Enable);
    xil_printf("PS_pin setting AXI_GPIO driver Successfully\n\r");

    //set the AXI_GPIO . refer the xgpio.h
//  XGpio_SetDataDirection(&AXI_Gpio, LED_CHANNEL,0x01);
//  xil_printf("set XGpio_SetDataDirection Successfully\n\r");



    SetupInterruptSystem(&Intc, &AXI_Gpio, INTC_GPIO_INTERRUPT_ID);
    xil_printf("SetupInterruptSystem Successfully  \r\n");
```

```
int SetupInterruptSystem(XScuGic *GicInstancePtr,XGpio *AXI_Gpio,u16 AXI_GpioIntrId)
{

    XScuGic_Config *IntcConfig; /* Instance of the interrupt controller */

    IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
//  xil_printf("XScuGic_LookupConfig Successfully  \r\n");
    XScuGic_CfgInitialize(GicInstancePtr, IntcConfig,IntcConfig->CpuBaseAddress);
//  xil_printf("XScuGic_CfgInitialize Successfully  \r\n");
//  xil_printf("================================\r\n");

    /*
     * Connect the interrupt controller interrupt handler to the hardware interrupt handling logic in the processor.
     */
    Xil_ExceptionInit();
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,(Xil_ExceptionHandler)XScuGic_InterruptHandler, GicInstancePtr);
    /* Enable interrupts in the Processor. */
    Xil_ExceptionEnableMask(XIL_EXCEPTION_IRQ);

//  print("Xil_ExceptionInit Successfully\r\n");
    /*
     * Connect the device driver handler that will be called when an
     * interrupt for the device occurs, the handler defined above performs
     * the specific interrupt processing for the device.
     */
    XScuGic_Connect(GicInstancePtr, AXI_GpioIntrId,(Xil_ExceptionHandler)IntrHandler,(void *)AXI_Gpio);
    /* Set the handler for AXI_Gpio interrupts. */
//  XGpioPs_SetCallbackHandler(AXI_Gpio, (void *)Gpio, IntrHandler);

//  print("XScuGic_Connect Successfully\r\n");
    /* Enable the interrupt for the AXI_Gpio device. */
    XScuGic_Enable(GicInstancePtr, AXI_GpioIntrId);
```

Mostly function are seam , just edit the GPIO point and AXI_GPIO_INTR_ID(61)

```
    XScuGic_SetPriorityTriggerType(GicInstancePtr, AXI_GpioIntrId,0xA0, 0x01);

    /*
     * Enable the AXI GPIO channel interrupts so that push button can be
     * detected and enable interrupts for the AXI GPIO device
     */
    XGpio_InterruptGlobalEnable(AXI_Gpio);
    XGpio_InterruptEnable(AXI_Gpio, BUTTON_CHANNEL);

    return XST_SUCCESS;
}
```

- The function different to EMIO and MIO.
- When the system are setting done, in the last , need to Enable the Global interrupt and interrupt.
- Because block diagram in vivado, have two GPIO1 and GPIO2 in one AXI_GPIO, channel 1 is LED , channel 2 is BUTTON. About input/output different, in this case, using the BUTTON channel.

```
void IntrHandler(){
    key_press = 1;
    print("Interrupt Successfully\r\n");
    sleep(3);
    //disable interrupt
    XGpio_InterruptDisable(&AXI_Gpio, BUTTON_CHANNEL);
    print("Disable Interrupt Successfully\r\n");
}
```

Action and intrhandler

```
    while(1){
//      xil_printf("SetupInterruptSystem Successfully ,key_press = %d\r\n",key_press);
        if(key_press){
            led_value = ~led_value;
            key_press = 0;
            //clear the interrupt statue
            XGpio_DiscreteWrite(&AXI_Gpio, LED_CHANNEL, led_value);
            xil_printf("Light change statue Successfully  \r\n");

            sleep(1);
            XGpio_InterruptClear(&AXI_Gpio, BUTTON_CHANNEL);
            xil_printf("XGpio_InterruptClear Successfully  \r\n");
            //enable interrupt
            XGpio_InterruptEnable(&AXI_Gpio, BUTTON_CHANNEL);
        }

    }
    return 0;
}
```


AMD