APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL version) works twenty based on external interrupt testing on the PS side

The code of the key in the previous section is in the main program to constantly scan the IO state to achieve the purpose of key detection, which is no problem for small systems, but for large and complex system programs, if the main loop of main has to process a lot of transactions, the detection of IO or the triggering of external events can not respond in time, and the key cycle itself also occupies system resources, at this time the importance of external interruptions is highlighted.

External interrupts allow the MCU to process external events in real time, and when external events occur, the system's interrupt mechanism will force the CPU to suspend the executing program and instead handle interrupt events; After the interrupt is processed. It returned to the interrupted procedure and continued its implementation.

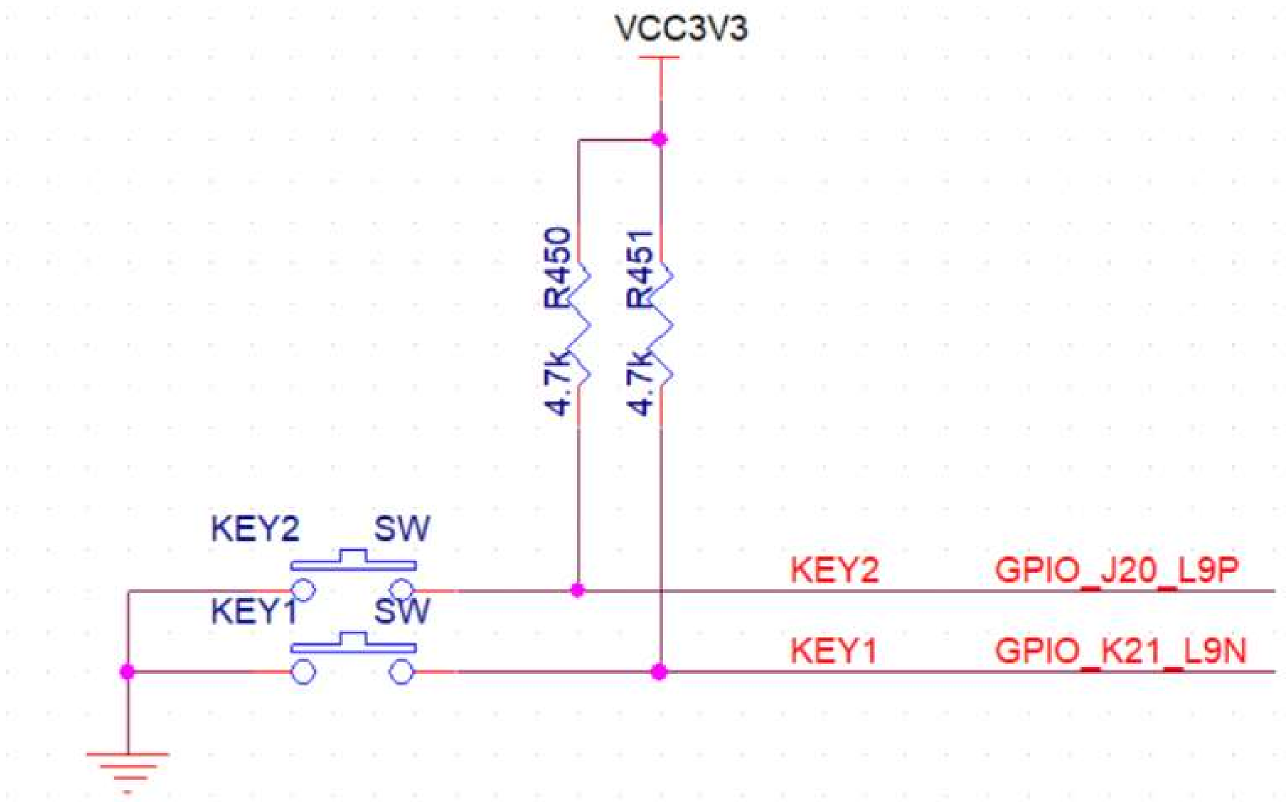
This article is demonstrated on vivado2018.3, please research for other versions

(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)

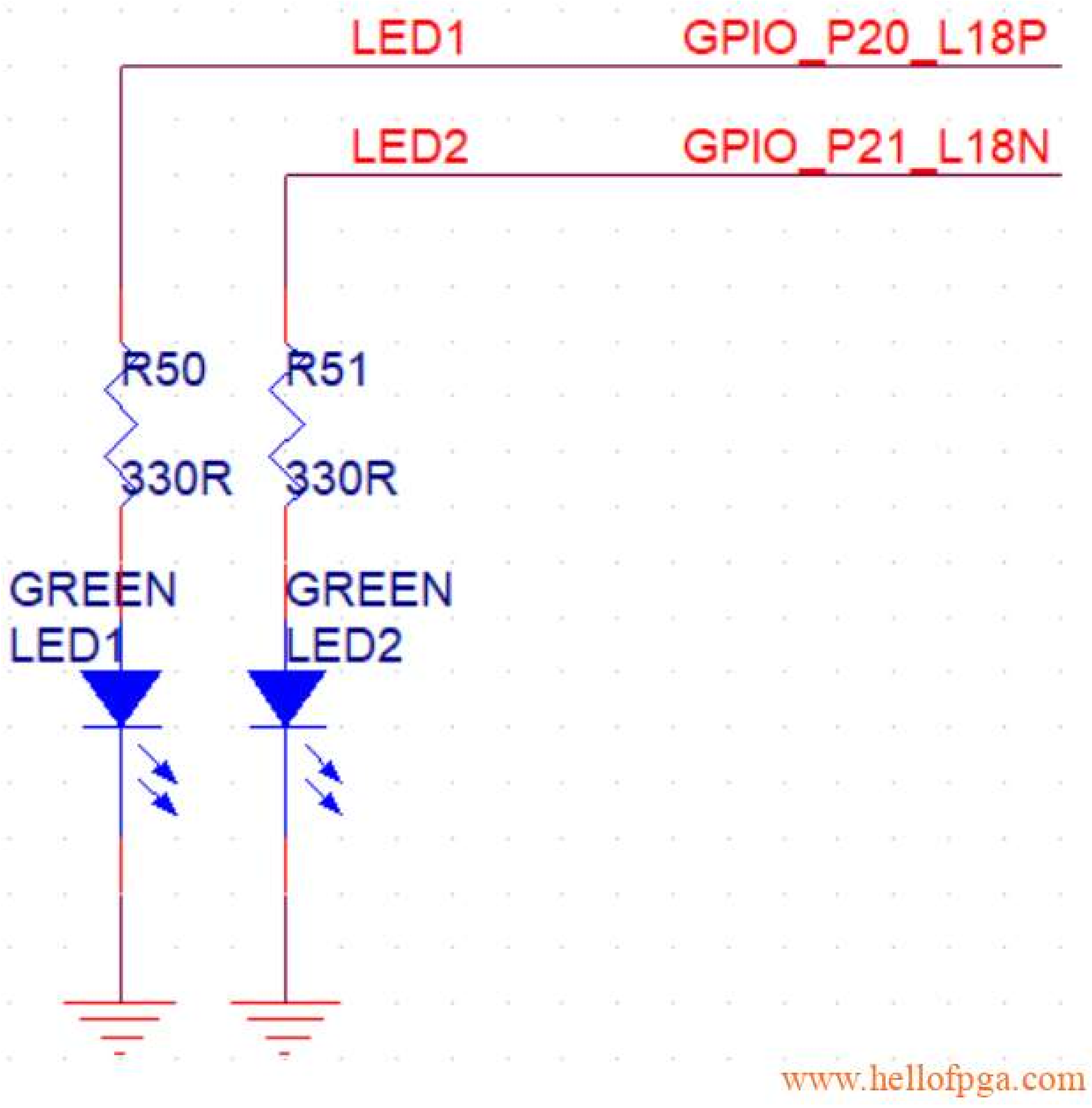
Introduction to hardware

This experiment still uses the key lighting method to demonstrate the external interrupt, let the key trigger the function of the external interrupt, and add the command to change the lamp state in the interrupt function.

Looking at the schematic diagram first, it can be seen from the schematic diagram that there are two buttons connected to the board, and the default is to pull up to 3.3V through the pull-up resistor, that is, when the button is not pressed, the signal pin of the key is 3.3V, and when the button is pressed, the signal pin is pulled down to GND, that is, 0V It can also be seen that the two buttons are connected to the J20 and K21 pins of the FPGA chip



In order to demonstrate the function of the two buttons, two more indicator lights are introduced here, the indicator light has been tested in the previous 3 sections of the example, the drive pin of the LED is pulled high, the indicator light is on, the indicator light is pulled low, and the indicator light is connected to the P20 and P21 pins of the main chip respectively



工程创建

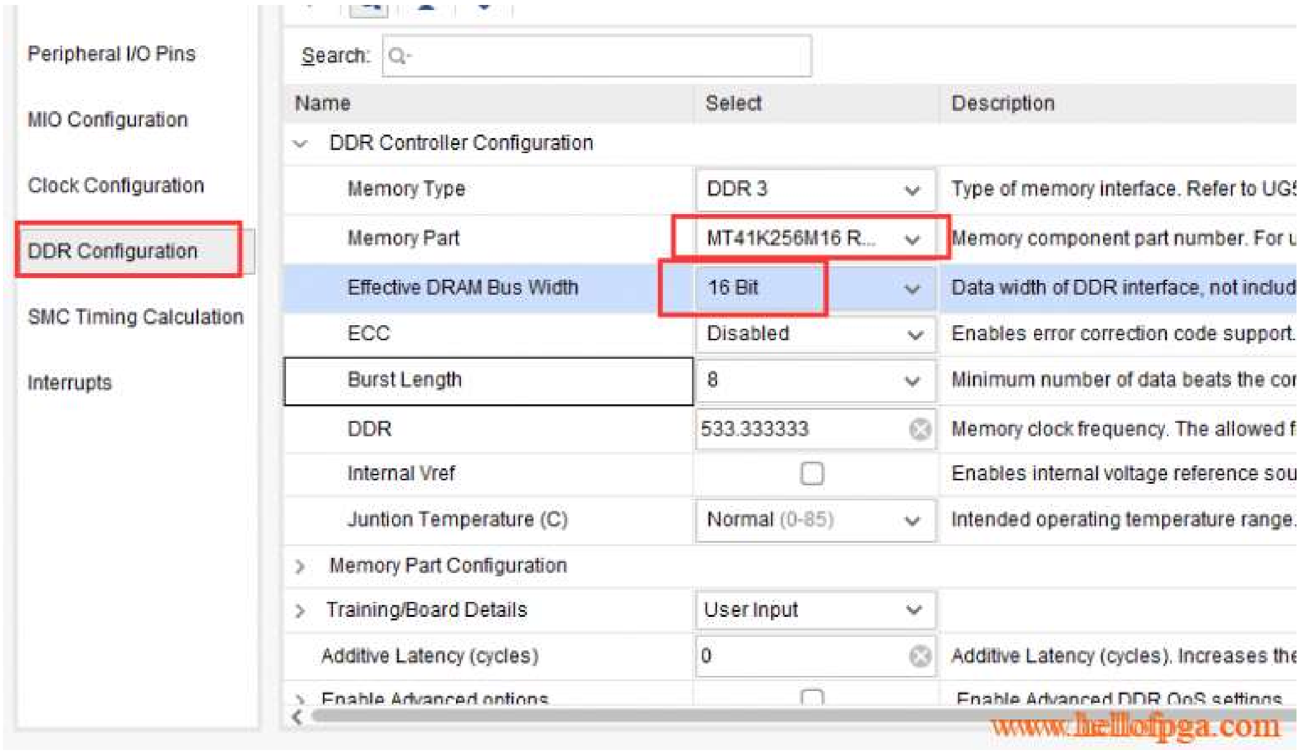
工程几乎和上一个实验完全一致，仅简单介绍。

完整的工程创建图文教程前面已经详细图文描写了，可以参考前面的工程，这里仅仅增加关键步骤。 详细过程可以参考Smart ZYNQ（SP&SL 版) 工程七 用ZYNQ的PS点亮连接到PL端的LED灯EMIO 方式 （推荐使用方式）

器件名称选择 XC7Z020CLG484-1

在BLOCK DESIGN 中添加一个ZYNQ模块

在ZYNQ 设置界面，设置 DDR MT41K256M16RE-125的型号 和位宽16bit

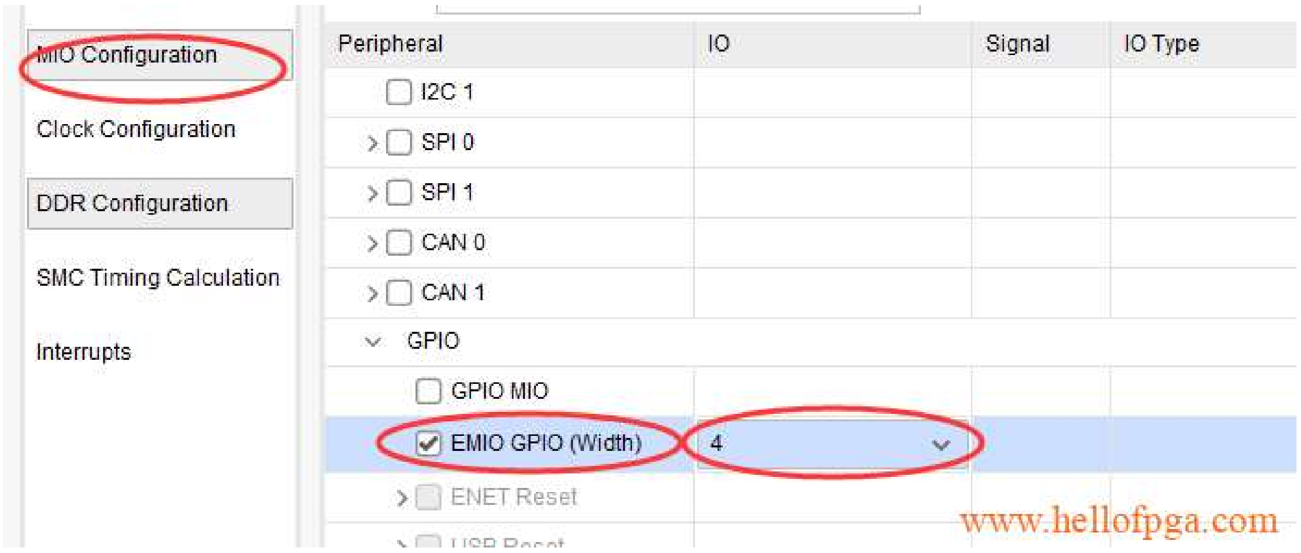


Peripheral I/O Pins
MIO Configuration
Clock Configuration
DDR Configuration
SMC Timing Calculation
Interrupts

Search: Q-

Name	Select	Description
DDR Controller Configuration		
Memory Type	DDR 3	Type of memory interface. Refer to UG5
Memory Part	MT41K256M16 R...	Memory component part number. For u
Effective DRAM Bus Width	16 Bit	Data width of DDR interface, not includ
ECC	Disabled	Enables error correction code support.
Burst Length	8	Minimum number of data beats the cor
DDR	533.333333	Memory clock frequency. The allowed f
Internal Vref	☐	Enables internal voltage reference sou
Juntion Temperature (C)	Normal (0-85)	Intended operating temperature range.
Memory Part Configuration		
Training/Board Details	User Input	
Additive Latency (cycles)	0	Additive Latency (cycles). Increases the
Enable Advanced options	☐	Enable Advanced DDR QoS settings

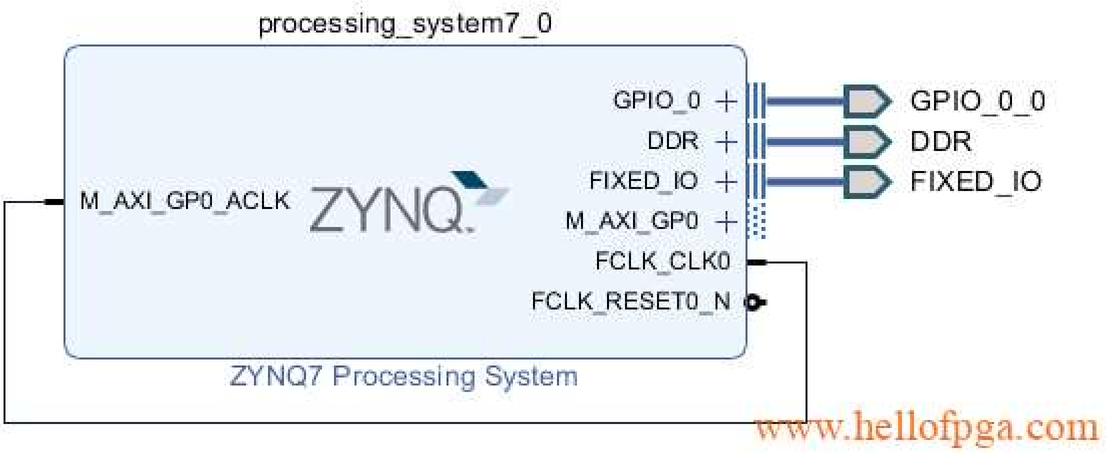
添加4个EMIO ，其中两个用来驱动LED灯，另两个用来读取按键KEY的信息



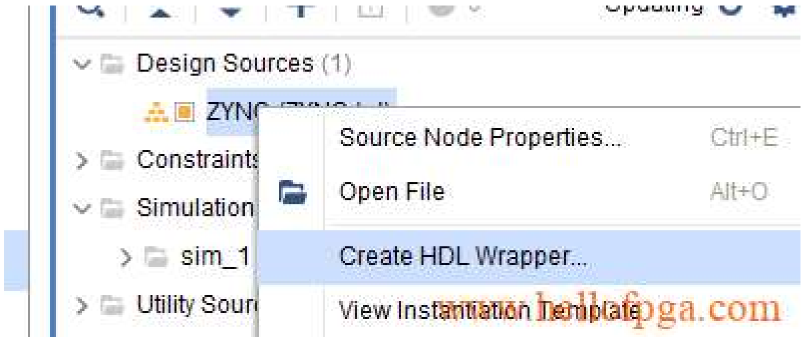
MIO Configuration
Clock Configuration
DDR Configuration
SMC Timing Calculation
Interrupts

Peripheral	IO	Signal	IO Type
☐ I2C 1			
☐ SPI 0			
☐ SPI 1			
☐ CAN 0			
☐ CAN 1			
GPIO			
☐ GPIO MIO			
✓ EMIO GPIO (Width)	4		
☐ ENET Reset			
☐ USB Reset			

对GPIO make external，并且连接AXI的时钟（AXI默认开启的，这里用不到AXI，也可以去设置里关闭AXI功能，就不需要连接了）

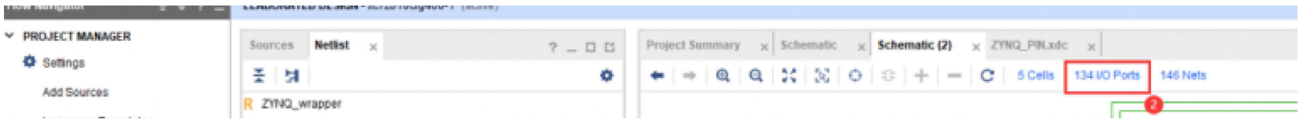


之后 点选Create HDL Wrapper



之后创建和添加约束文件:

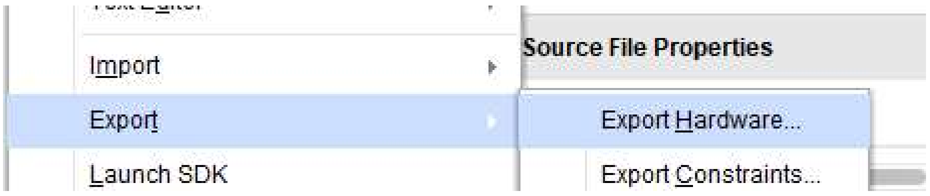
可以在图文界面里设置管脚定义

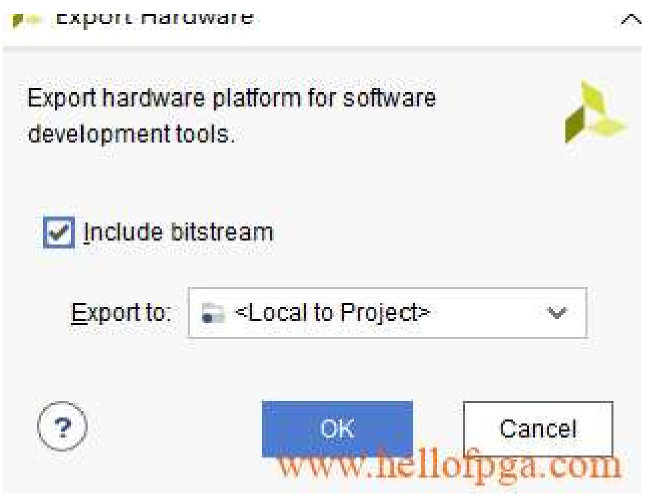


也可以在约束文件里添加管脚定义的描述

```
set_property PACKAGE_PIN P20 [get_ports {GPIO_0_0_tri_io[3]}]
set_property PACKAGE_PIN P21 [get_ports {GPIO_0_0_tri_io[2]}]
set_property PACKAGE_PIN K21 [get_ports {GPIO_0_0_tri_io[1]}]
set_property PACKAGE_PIN J20 [get_ports {GPIO_0_0_tri_io[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[0]}]
```

之后对程序进行正常的编译综合，并且导出 Export Hardware (勾选 include bitstream)供SDK加载




Export Hardware

Export hardware platform for software development tools.

[✓] Include bitstream

Export to: <Local to Project>

程序设计：

建立一个名为GPIO_INTERRUPT空的工程，并且添加main.c 添加如下代码：

```
#include "xparameters.h"
#include "xgpiops.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xplatform_info.h"
#include <xil_printf.h>
```

```
#define LED2            57
#define LED1            56
#define KEY2            55
#define KEY1            54

#define GPIO_DEVICE_ID          XPAR_XGPIOPS_0_DEVICE_ID
XGpioPs Gpio;
#define GPIO_BANK       XGPIOPS_BANK0  /* Bank 0 of the GPIO
Device */

#define GPIO_DEVICE_ID          XPAR_XGPIOPS_0_DEVICE_ID
#define INTC_DEVICE_ID          XPAR_SCUGIC_SINGLE_DEVICE_ID
#define GPIO_INTERRUPT_ID       XPAR_XGPIOPS_0_INTR

static XScuGic Intc; /* The Instance of the Interrupt Controller
Driver */


static void IntrHandler(void *CallBackRef, u32 Bank, u32 Status)
{
        XGpioPs *Gpio_cb = (XGpioPs *)CallBackRef;
        if (XGpioPs_IntrGetStatusPin(Gpio_cb, KEY1)){
                XGpioPs_WritePin(&Gpio, LED1, 1);
                XGpioPs_WritePin(&Gpio, LED2, 0);
                XGpioPs_IntrClearPin(Gpio_cb, KEY1);
        }
        else if (XGpioPs_IntrGetStatusPin(Gpio_cb, KEY2)){
                XGpioPs_WritePin(&Gpio, LED1, 0);
                XGpioPs_WritePin(&Gpio, LED2, 1);
                XGpioPs_IntrClearPin(Gpio_cb, KEY2);
        }
}



void SetupInterruptSystem(XScuGic *GicInstancePtr, XGpioPs *Gpio,
                                u16 GpioIntrId){

        XScuGic_Config *IntcConfig;
        Xil_ExceptionInit();

        IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
        XScuGic_CfgInitialize(GicInstancePtr, IntcConfig,
                                        IntcConfig-
>CpuBaseAddress);

        Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,

(Xil_ExceptionHandler)XScuGic_InterruptHandler,
                                GicInstancePtr);
        XScuGic_Connect(GicInstancePtr, GpioIntrId,
                                (Xil_ExceptionHandler)IntrHandler,
                                (void *)Gpio);


        XScuGic_Enable(GicInstancePtr, GpioIntrId);

        XGpioPs_SetIntrTypePin(Gpio, KEY1,
XGPIOPS_IRQ_TYPE_EDGE_FALLING);
        XGpioPs_SetIntrTypePin(Gpio, KEY2,
XGPIOPS_IRQ_TYPE_EDGE_FALLING);

        XGpioPs_IntrEnablePin(Gpio, KEY1);
        XGpioPs_IntrEnablePin(Gpio, KEY2);

        Xil_ExceptionEnableMask(XIL_EXCEPTION_IRQ);
}


void Gpio_Init(void){
        XGpioPs_Config *ConfigPtr;

        ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
        XGpioPs_CfgInitialize(&Gpio, ConfigPtr,ConfigPtr-
>BaseAddr);

        XGpioPs_SetDirectionPin(&Gpio, LED1, 1);
        XGpioPs_SetOutputEnablePin(&Gpio, LED1, 1);
        XGpioPs_WritePin(&Gpio, LED1, 1);

        XGpioPs_SetDirectionPin(&Gpio, LED2, 1);
        XGpioPs_SetOutputEnablePin(&Gpio, LED2, 1);
        XGpioPs_WritePin(&Gpio, LED2, 1);
        XGpioPs_SetDirectionPin(&Gpio, KEY1, 0);
        XGpioPs_SetDirectionPin(&Gpio, KEY2, 0);

        SetupInterruptSystem(&Intc, &Gpio, GPIO_INTERRUPT_ID);
}
```

```
int main(void)
{
        Gpio_Init();

        while(1){

        }

        return 0;
}
```

程序很简单， Gpio_Init是对4个GPIO进行初始化， 因为EMIO 是从54的管脚号开始的，所以根据原先VIVADO 的管脚约束来看，两个按键应该对应54和55，两个LED灯对应56和57

SetupInterruptSystem(&Intc, &Gpio, GPIO_INTERRUPT_ID); 是对中断进行初始化，这里我们分别对KEY1和KEY2都增加了中断初始化，这两两个按键中任何一个按键被按下 都会调用中断XGpioPs_SetIntrTypePin(Gpio, KEY1, XGPIOPS_IRQ_TYPE_EDGE_FALLING); 代表对KEY1设置下降沿中断。

IntrHandler 是中断服务函数，在中断事件产生后对中断进行响应，一般我们中断后需要的动作代码都放在这个函数里面。 （因为我们同时增加了两个外部中断，所以中断回调函数进来第一件事是区分哪个GPIO产生了中断，所以这里用XGpioPs_IntrGetStatusPin命令去读取CallBackRef的值，来判断哪个按键产生了中断，并根据结果点亮不同的LED灯）

```
static void IntrHandler(void *CallBackRef, u32 Bank, u32 Status)
{
        XGpioPs *Gpio_cb = (XGpioPs *)CallBackRef;
        if (XGpioPs_IntrGetStatusPin(Gpio_cb, KEY1)){
                XGpioPs_WritePin(&Gpio, LED1, 1);
                XGpioPs_WritePin(&Gpio, LED2, 0);
                XGpioPs_IntrClearPin(Gpio_cb, KEY1);
        }
        else if (XGpioPs_IntrGetStatusPin(Gpio_cb, KEY2)){
                XGpioPs_WritePin(&Gpio, LED1, 0);
                XGpioPs_WritePin(&Gpio, LED2, 1);
                XGpioPs_IntrClearPin(Gpio_cb, KEY2);
        }
}
```

For the main function function, we only need to complete the GPIO initialization (including interrupt initialization), and then do nothing directly

```
int main(void)
{
        Gpio_Init();
        while(1){

        }

        return 0;
}
```

We downloaded the program into the board for DEBUG, and found that either of the two buttons was pressed and the corresponding light was lit, proving that our external interrupt had run successfully (because the main program is not responsible for lighting the light, only in the interrupt is responsible for changing the state of the lamp)

Interrupts are relatively simple, but they are essential for systematic programming.

Here is the full code (FYI):

20_PS_GPIO_INTERRUPT_XC7Z020 **Download**

SMART ZYNQ SP & SL