APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL Edition) Project XIX Demonstration of the GPIO (EMIO) input function based on the PS side

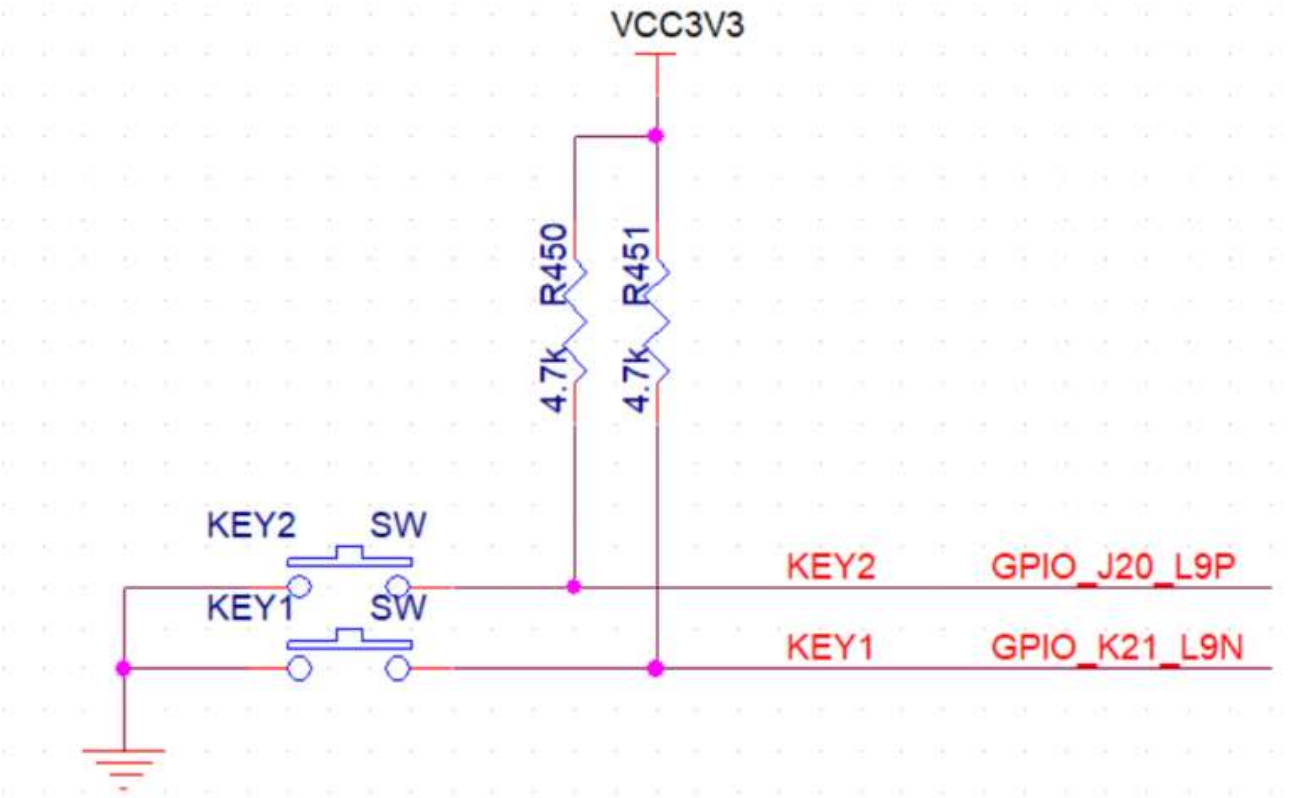
This experiment mainly demonstrates the GPIO reading function of EMIO

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

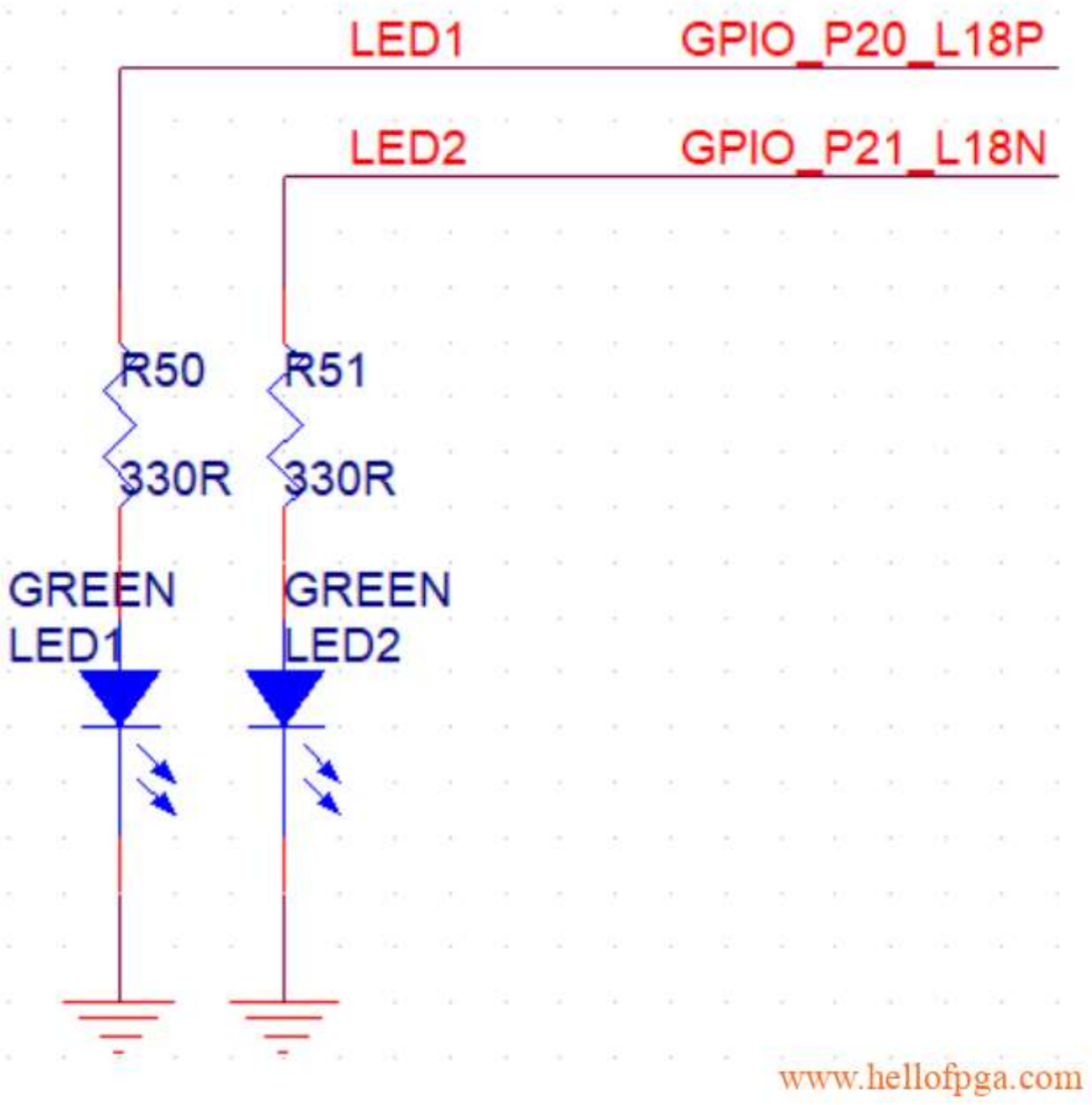
(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)

Introduction to hardware

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



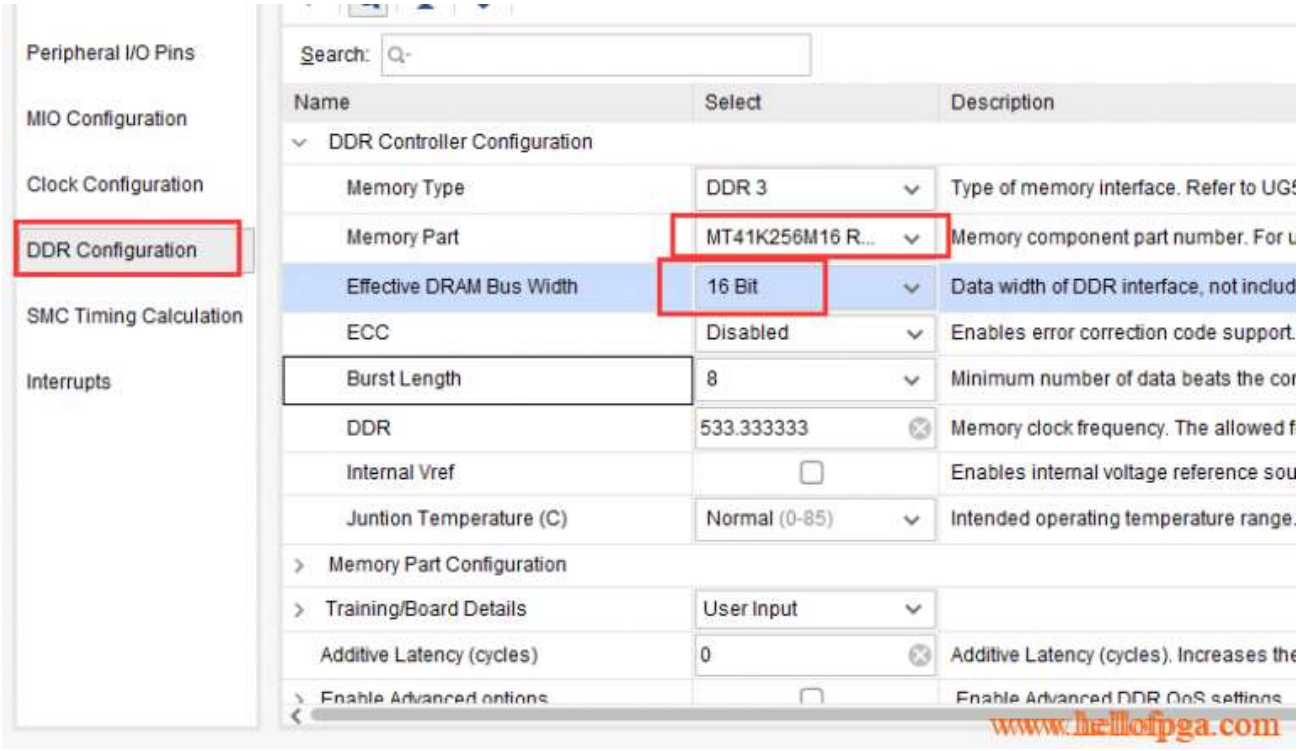
Project creation

The complete project creation graphic tutorial has been described in detail earlier, you can refer to the previous project, here only add key steps. For the detailed process, please refer to Smart ZYNQ (SP&SL version) Project 7 to use ZYNQ's PS to light the LED light EMIO method connected to the PL end

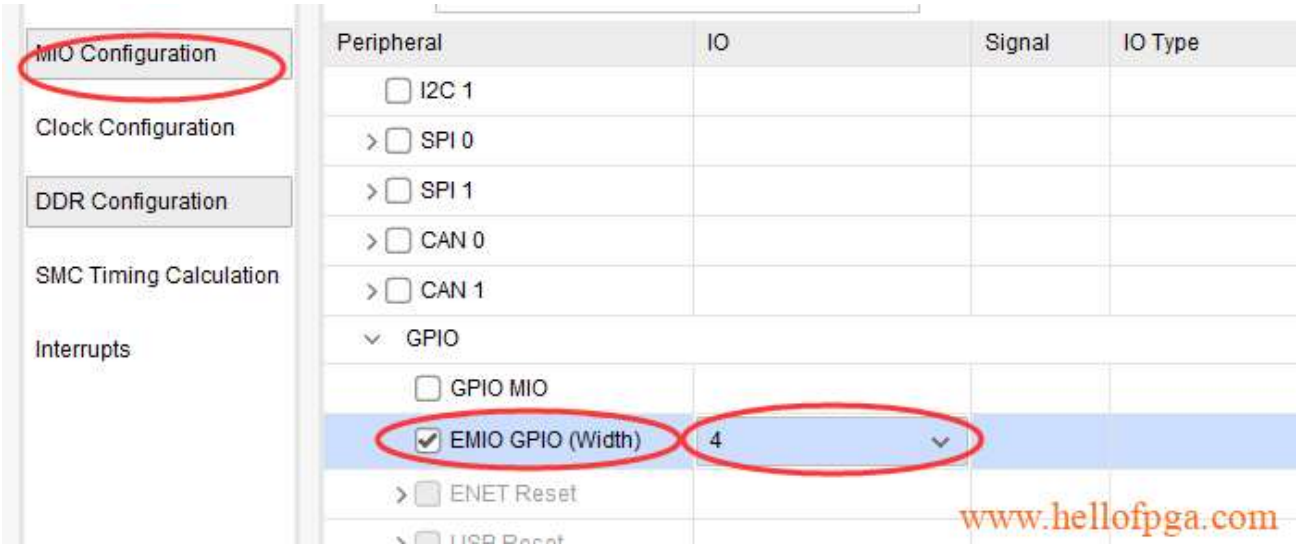
Select the XC7Z020CLG484-1 for the part name

SEARCH FOR AND ADD A ZYNQ MODULE IN BLOCK DESIGN

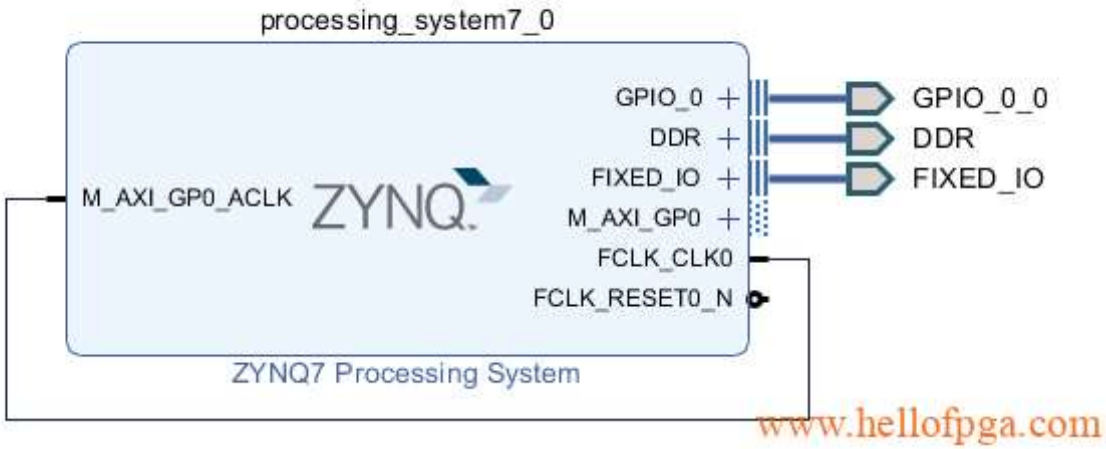
On the ZYNQ setting interface, set the model and bit width of DDR MT41K256M16RE-125 to 16bit



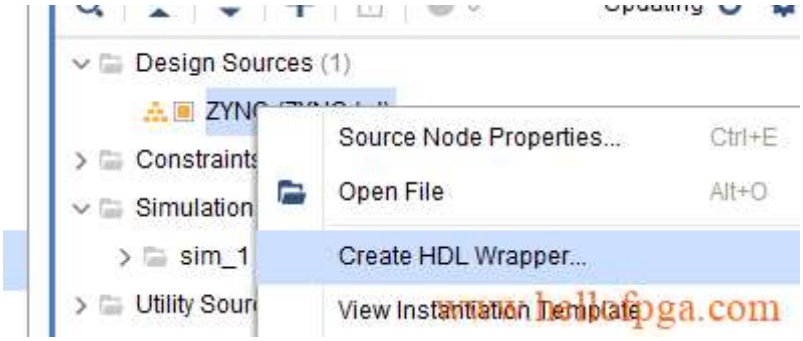
Add 4 EMIOs, two of which are used to drive LED lights and the other two are used to read the information of the key KEY



Make external for GPIO, and connect AXI's clock (AXI is enabled by default, AXI is not used here, you can also go to the settings to turn off the AXI function, there is no need to connect)

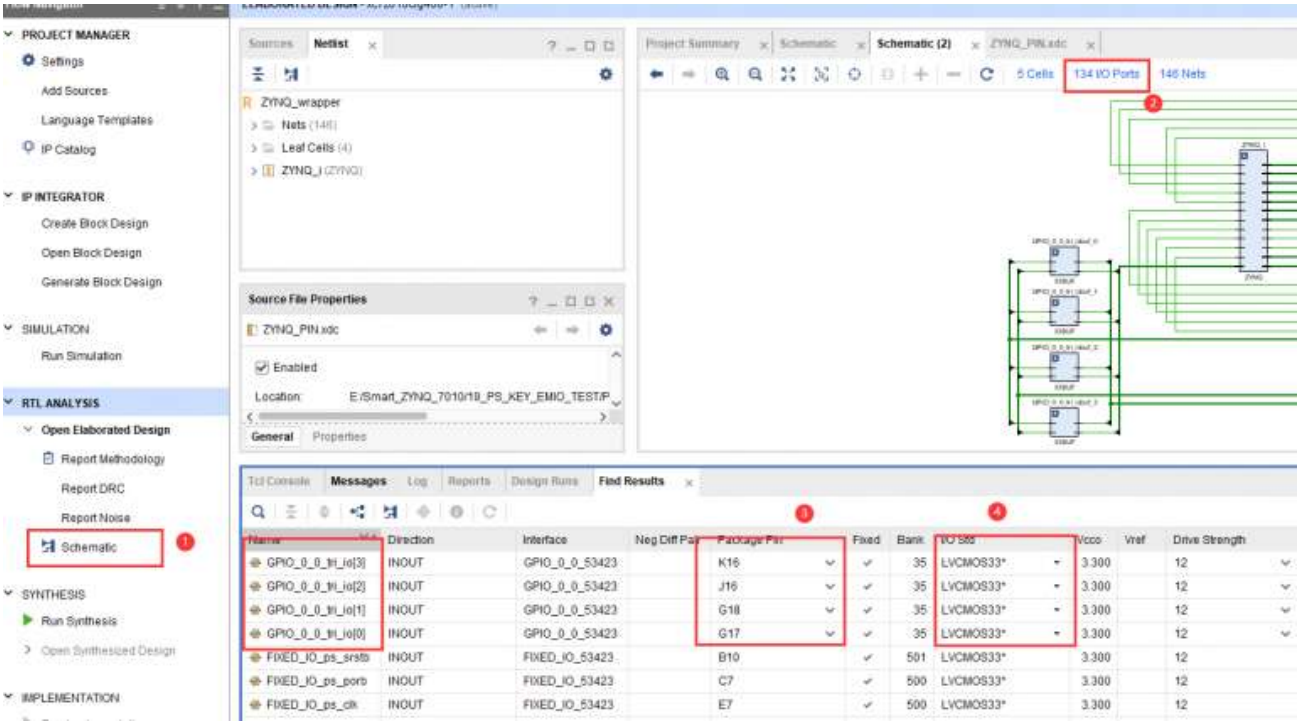


After that, click Create HDL Wrapper



Create and add constraint files afterwards:

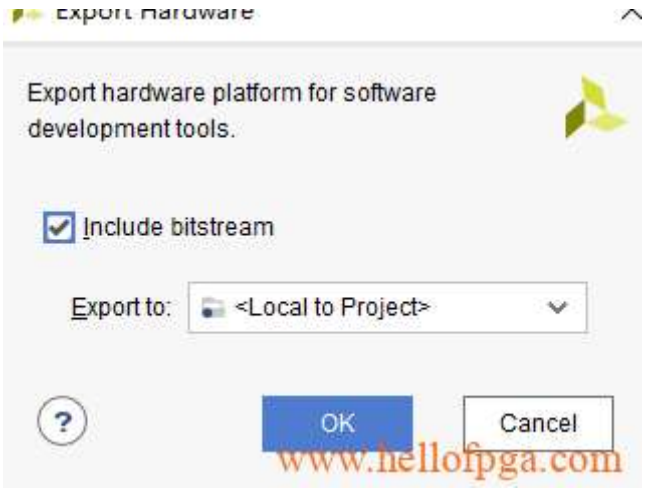
You can set the pin definition in the graphical interface



You can also add a description of the pin definition to the constraint file

```
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]}]
```

After that, the program is compiled and synthesized normally, and Export Hardware (check include bitstream) is exported for SDK loading



Programming:

Create a project named KEY_TEST Null and add main.c to add the following code:

```
#include "xparameters.h"
#include "xgpiops.h"
#include "xstatus.h"
#include "xplatform_info.h"
#include "xscutimer.h"
#include "Xscugic.h"
```

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

#define GPIO_DEVICE_ID          XPAR_XGPIOPS_0_DEVICE_ID
XGpioPs Gpio;


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);
}




int main(void)
{
        Gpio_Init();

        while(1){
                if(XGpioPs_ReadPin(&Gpio, KEY1)==0){
                        XGpioPs_WritePin(&Gpio, LED1, 1);
                        XGpioPs_WritePin(&Gpio, LED2, 0);
                }
                else if(XGpioPs_ReadPin(&Gpio, KEY2)==0){
                        XGpioPs_WritePin(&Gpio, LED1, 0);
```

```
                        XGpioPs_WritePin(&Gpio, LED2, 1);
                }
        }

        return 0;
}
```

The program is simple, Gpio_Init is to initialize the 4 GPIOs, because EMIO starts with the pin number 54, so according to the original VIVADO pin constraints, the two buttons should correspond to 54 and 55, and the two LED lights should correspond to 56 and 57

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

XGpioPs_SetDirectionPin(&Gpio, KEY1, 0); The last 0 represents the input, XGpioPs_WritePin (&Gpio, LED1, 1); Like this last 1 represents the output.

The main program is also very simple, after initializing the GPIO, start to cycle through the key status, when button 1 is pressed, LED1 is on, LED2 is off, when button 2 is pressed, LED2 is on, LED1 is off (both lights are on by default after power-on)

The program is only for demonstration, and in the actual project, it is necessary to add a key destabilization program to the key, otherwise the program code will be repeatedly triggered when pressed

The following is the complete project (for learning reference only)

19_PS_KEY_EMIO_TEST_XC7Z020 **Download**

SMART ZYNQ SP & SL