

# EBAZ4205 Use ZYNQ's PS IO resource to simulate the SPI protocol to light up the color LCD screen

## 1. Hardware introduction

Part of the hardware LCD interface of the rotary board is shown in the figure below

LCD backlight is often bright (purple rotor, the backlight can be controlled)

CS chip signal (the purple board is directly connected to the GND, and the black CS is connected)

SCL clock

SDA data

D/C data/command instructions

Res screen reset

## 2.Principle

This article uses Zynq's IO port to simulate the hardware SPI to drive the LCD screen, and the IO port is mapped to the PL interface by the PS side by EMIO

## 3. Creation project

1) Create a new project, chip model select XC7Z010CLG400-1

2) Create a block design and add Zynq7 Processing System module. The software automatically generates a zynq block as shown in the figure below.  
Next



### 3) Set the clock function in ZYNQ:

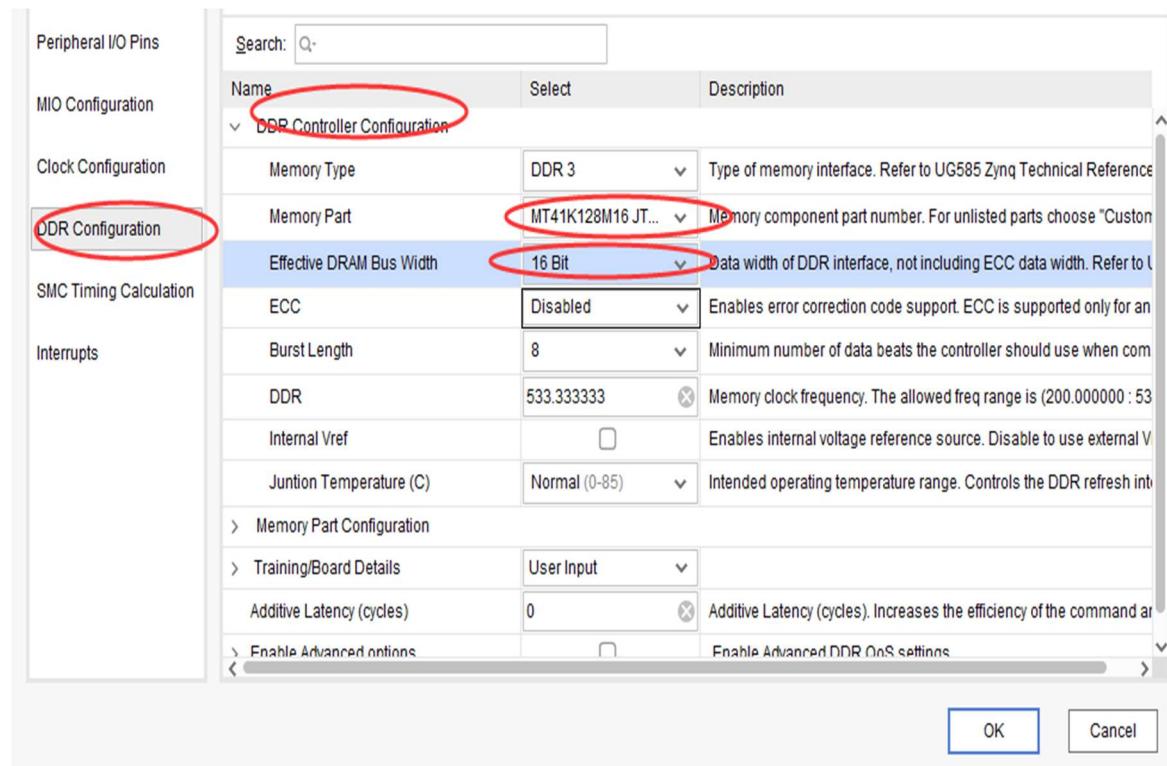
Find the CLOCK Configuration option in the project, set the clock frequency you need in PL Fabric Clocks. There are 4 frequencies here that can set our PLL function. Here we set up 50M clock

The screenshot shows the ZYNQ7 Processing System (5.5) software interface with the following details:

- Title Bar:** ZYNQ7 Processing System (5.5)
- Top Navigation:** Documentation, Presets, IP Location, Import XPS Settings, Summary Report
- Left Sidebar (Page Navigator):**
  - Zynq Block Design
  - PS-PL Configuration
  - Peripheral I/O Pins
  - MIO Configuration
  - Clock Configuration** (highlighted with a red circle)
  - DDR Configuration
  - SMC Timing Calculation
  - Interrupts
- Central Area (Clock Configuration):**
  - Basic Clocking** tab is active.
  - Input Frequency (MHz): 33.333333
  - CPU Clock Ratio: 6:2:1
  - Search bar: Q:
  - PL Fabric Clocks** section (highlighted with a red box):
    - FCLK\_CLK0** (selected): IO PLL, 50, 50.000000, 0.100000 : 250.000000
    - FCLK\_CLK1**: IO PLL, 50, 10.000000, 0.100000 : 250.000000
    - FCLK\_CLK2**: IO PLL, 50, 10.000000, 0.100000 : 250.000000
    - FCLK\_CLK3**: IO PLL, 50, 10.000000, 0.100000 : 250.000000
  - System Debug Clocks**
  - Timers**

4) Set DDR function in Zynq:

Find DDR Configuration → DDR Controller Configuration → DDR3 in the pop-up window, select the corresponding DDR3 according to the DDR on your own board in the Memory PART drop-down menu. Strike " OK ", as shown in the figure below.



5) In addition to the clock signal and the data signal of EMIO, there are 3 extra GPIO ports to control the backlight BL and the D/C signal, so there are 5 GPIO ports here. GPIO position width Width select 5 (because it is 5 IO ports)

Zynq Block Design

Bank 0 I/O Voltage LVC... ▾

Bank 1 I/O Voltage LVC...

PS-PL Configuration

Peripheral I/O Pins

MIO Configuration

Clock Configuration

DDR Configuration

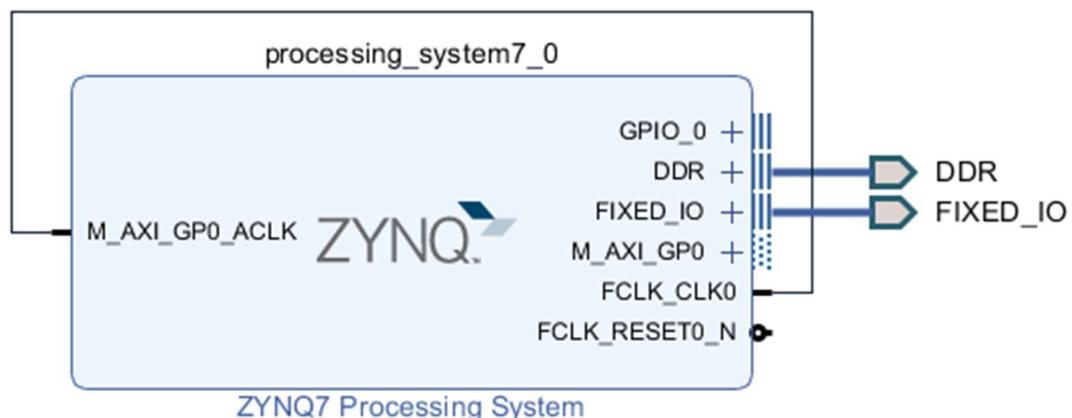
SMC Timing Calcula

Interrupts

Search:

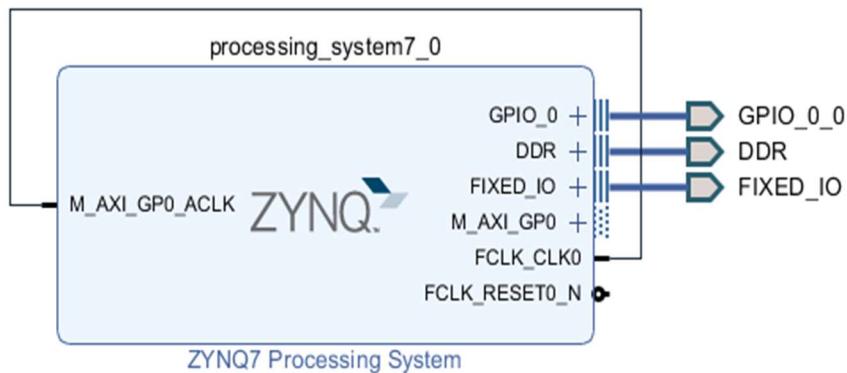
Peripheral	IO	Signal	IO Type	Sp
> SPI 0				
> SPI 1				
> CAN 0				
> CAN 1				
GPIO				
> GPIO MIO				
<input checked="" type="checkbox"/> EMIO GP...	5	▼		
> ENET R...				
> USB Reset				
> I2C Reset				

- 6) After completing the above operation, click "Run Block Automation" as shown in the figure below. Keep the default in the pop -up options, click "OK" to complete the configuration of Zynq7 Processing System, and connect to fclk\_clk and M\_AXI\_GP0\_ACLK with the mouse to get the figure below



Click the IO port in the figure above for the following operations:

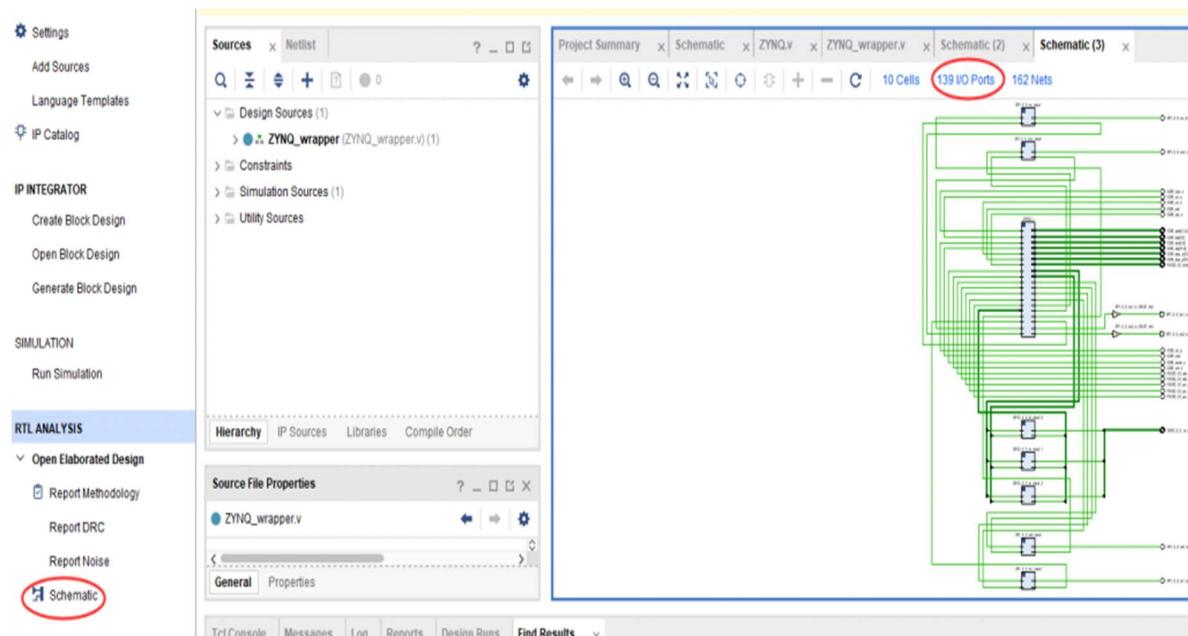
Right -click GPIO\_0 Select Make External



7) source→Design Source , Right -click the block project we created, click Create HDL Wrapper, package the block file and generate .v code

8) Click the green arrow RUN to compile the code

9) Click the schematic in RTL and select IO Ports on the right to increase the SPI's foot definition



Modify the definition of the GPIO tube foot and SPI tube foot. As shown in the figure below, save it after modification (as required by the pop -up window, enter the constraint file name in the window, and then save it)

BL T20

D/C R18

SCL R19

SDA P20

RES N17

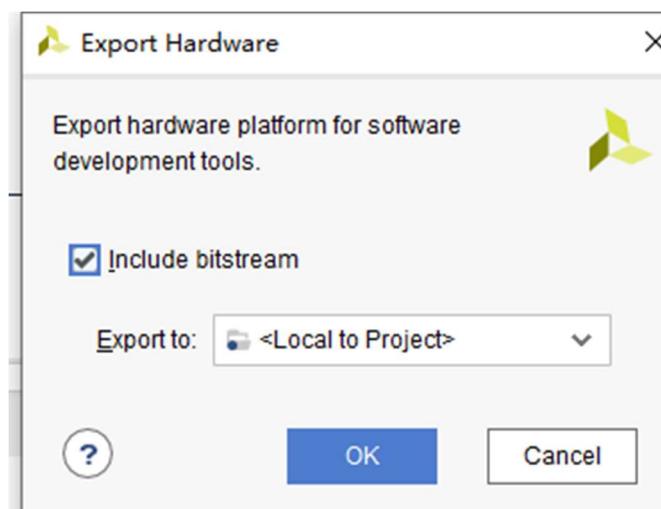
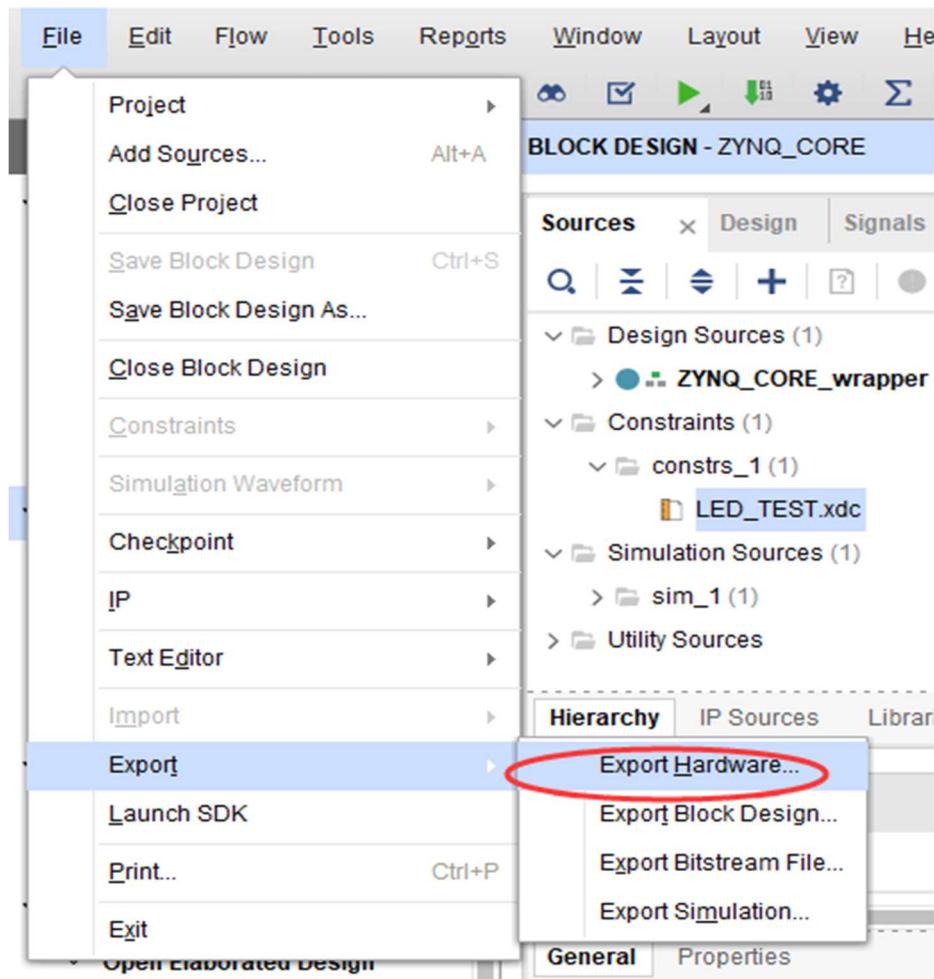
FIXED_IO_ps_srstb	INOUT	FIXED_IO_32035	B10	✓	501	LVCMOS33*
GPIO_0_0_tri_io[0]	INOUT	GPIO_0_0_32035	T20	✓	34	LVCMOS33*
GPIO_0_0_tri_io[1]	INOUT	GPIO_0_0_32035	R18	✓	34	LVCMOS33*
GPIO_0_0_tri_io[2]	INOUT	GPIO_0_0_32035	N17	✓	34	LVCMOS33*
GPIO_0_0_tri_io[3]	INOUT	GPIO_0_0_32035	R19	✓	34	LVCMOS33*
GPIO_0_0_tri_io[4]	INOUT	GPIO_0_0_32035	P20	✓	34	LVCMOS33*

10) Generate bit file: Press the Generate Bitstream to complete the synthesis and generate the BIT file

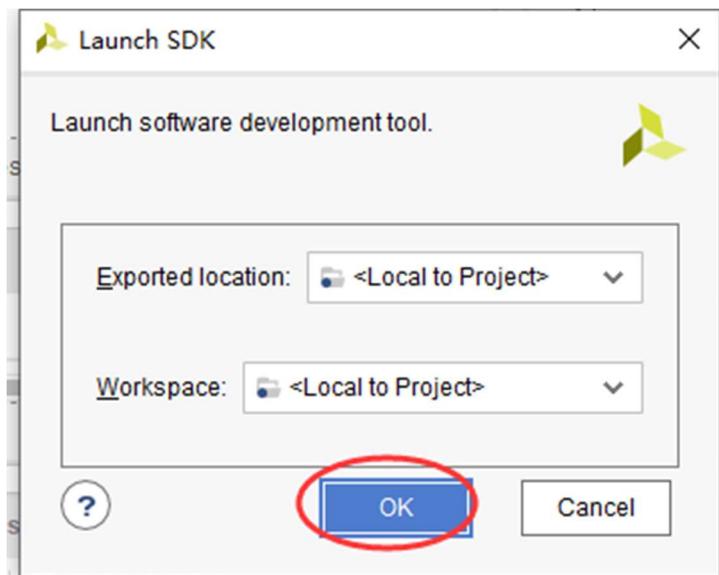


## 5. SDK program writing

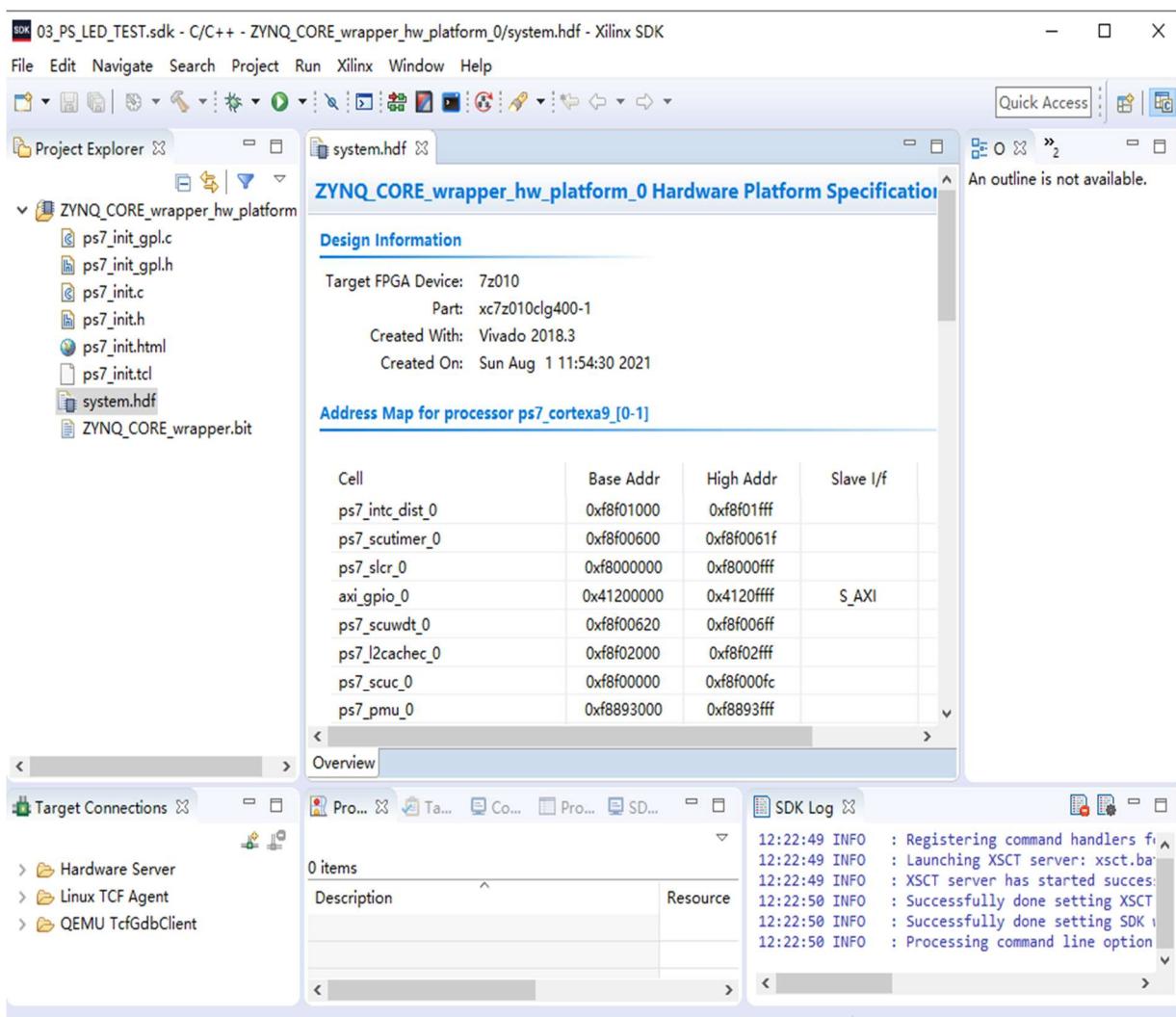
1) File→Export→Export hardware..., Select "Include Bitstream" in the pop - up dialog box and click "OK" to confirm, as shown in the figure below.



2) File→Launch SDK, In the pop -up dialog box, save the default and click "OK", as shown in the figure below.

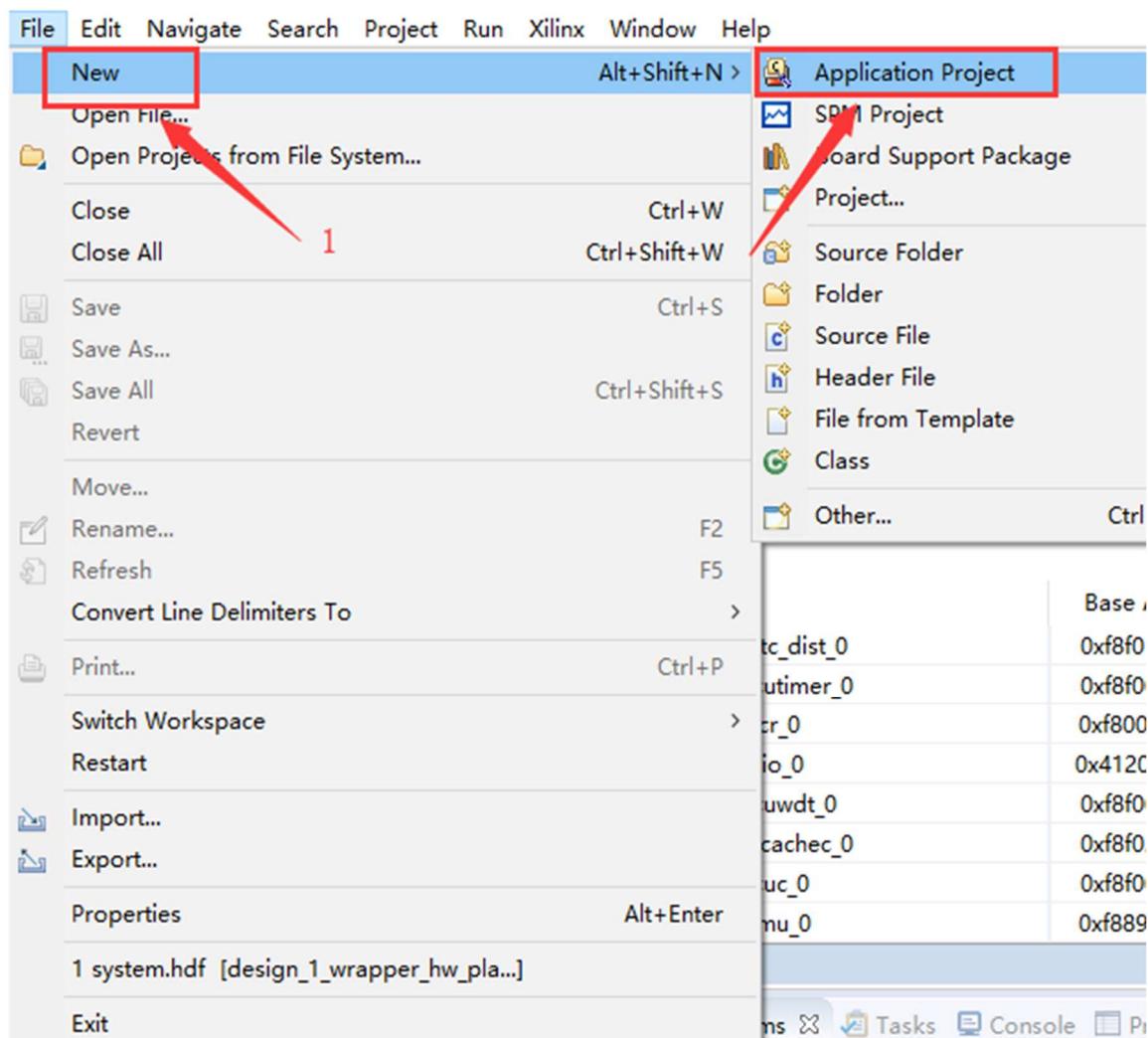


The system will automatically open the SDK development environment

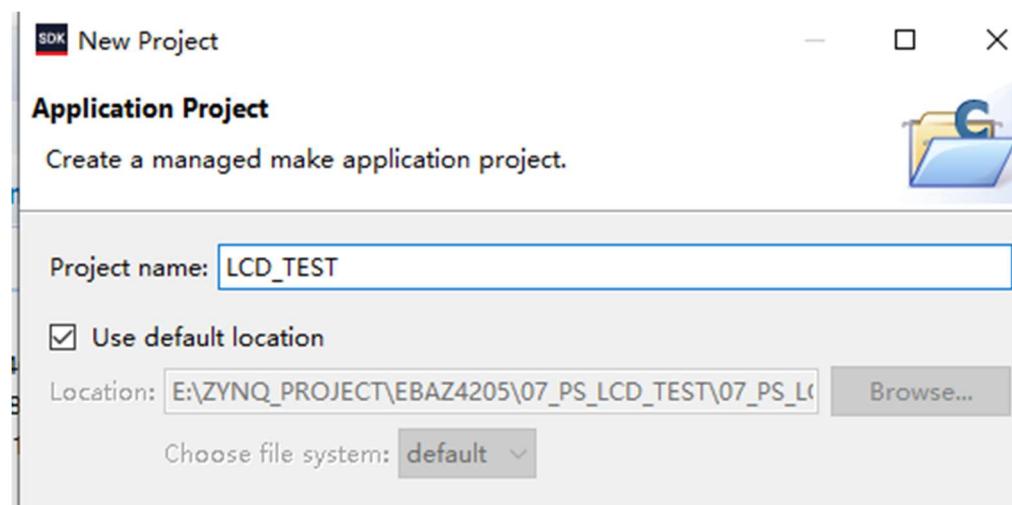


Cell	Base Addr	High Addr	Slave I/f
ps7_intc_dist_0	0xf8f01000	0xf8f01fff	
ps7_scutimer_0	0xf8f00600	0xf8f0061f	
ps7_slcr_0	0xf8000000	0xf8000fff	
axi_gpio_0	0x41200000	0x4120ffff	S_AXI
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	
ps7_scuc_0	0xf8f00000	0xf8f000fc	
ps7_pmu_0	0xf8893000	0xf8893fff	

3 Create a new project File → New → Application Project to create a new "Application Project", as shown in the figure below.



4) Enter your own project name in the new project name, click Next



5) Choose an Empty project, click to complete

## Templates

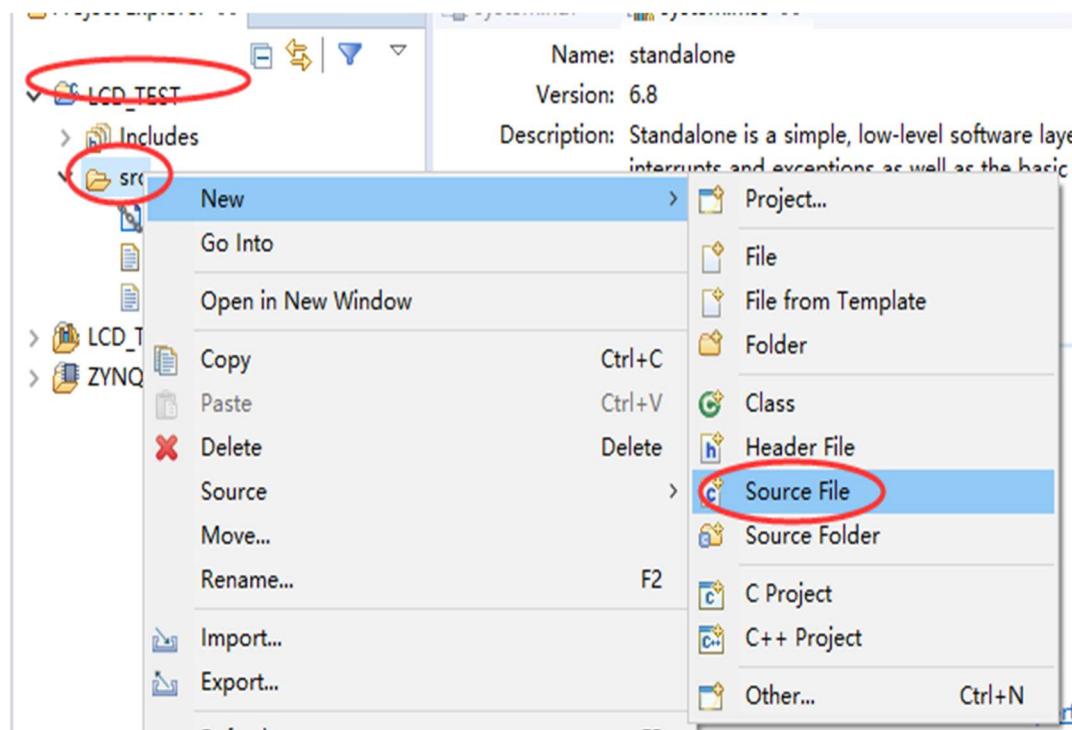
Create one of the available templates to generate a fully-functioning application project.

### Available Templates:

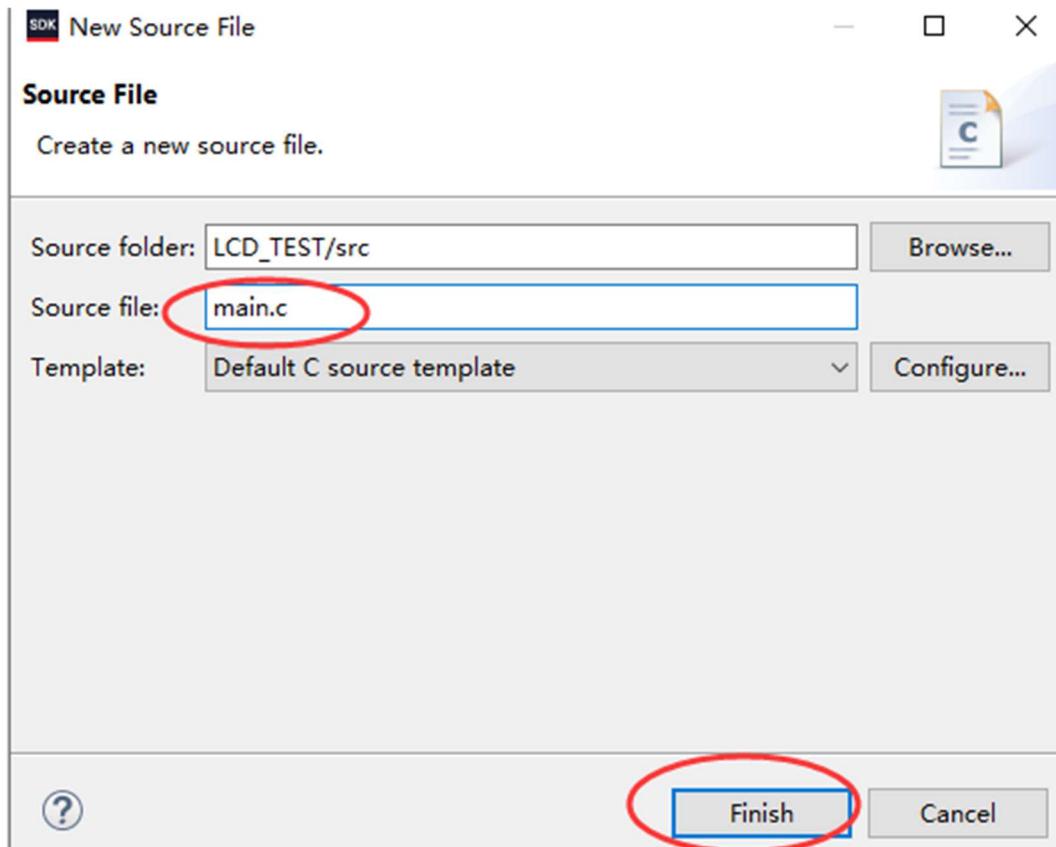
- Dhrystone
- Empty Application**
- Hello World
- IwIP Echo Server
- IwIP TCP Perf Client

6) Create our own code in the air engineering

Expand the project we created, right-click on the SRC directory, select New-> Source File, as shown in the figure below:



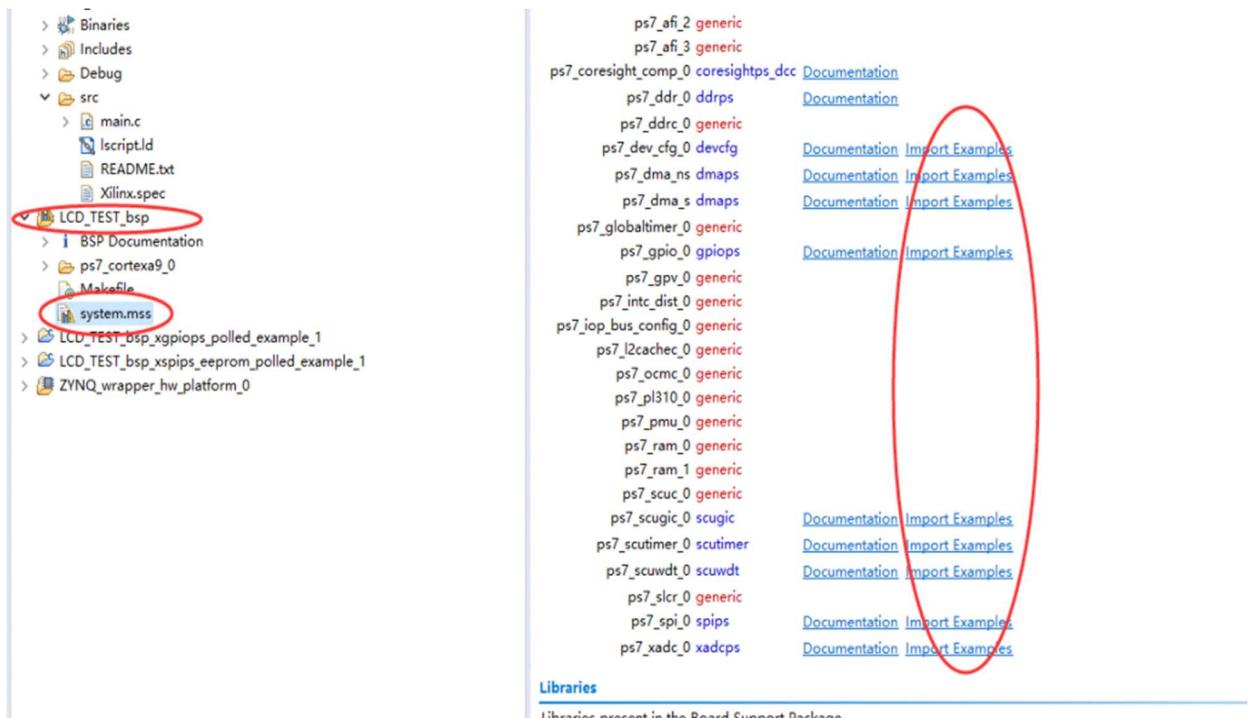
Create a main.c file in the pop -up window



6) Write your own code

#### 6.1 Tips:

When the code of the PS section is written for the first time, if the GPIO code does not know how to write, in fact, you can actually open the System.MSS file under the BSP project, and then import the required reference routine on the right. Part of the part (such as GPIO initialization, or SPI's writing and reading code Copy to the main function of your project)



## 6.2 Define the GPIO part

Let's enter the topic. We need to light up the screen. The screens of the screen use 5 GPIOs. They all use the EMIO resource on the Zynq PS side. Corresponding to EMIO's 54-55-56-57-58, respectively, as shown in the figure below define GPIO

```
#define EMIO_LCD_CS      54
#define EMIO_LCD_CD      55
#define EMIO_LCD_RES      56
#define EMIO_LCD_SCL      57
#define EMIO_LCD_SDA      58
```

## 6.3 Initialize the GPIO part

```
void Lcd_Gpio_Init(void) {
    XGpioPs_Config *ConfigPtr;
```

```
ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);

XGpioPs_CfgInitialize(&Gpio,
ConfigPtr, ConfigPtr->BaseAddr);

XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_CS, 1);

XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_CS, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_CS, 0);

XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_CD, 1);

XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_CD, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_CD, 0);

XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_RES, 1);

XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_RES, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_RES, 0);

XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_SCL, 1);

XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_SCL, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_RES, 0);
```

```

    XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_SDA, 1);

    XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_SDA, 1);

    XGpioPs_WritePin(&Gpio, EMIO_LCD_RES, 0);

}

```

First define the structure pointer of an XGPIOPS. The content of this pointer includes the ID and base address assigned by GPIO. This information is usually in the XPAMETER.H header file. information

ConfigPtr=XGpioPs\_LookupConfig(GPIO\_DEVICE\_ID); it is equivalent to giving the obtained information to the configptr structure defined before

Eventually use XGPIOPS\_CFGINITIALIZE function to initialize GPIO

GPIO 的使用 (EMIO\_LCD\_BL 为之前定义的 EMIO 54 的管脚)

XGpioPs\_SetDirectionPin(&Gpio, EMIO\_LCD\_CS, 1);// Set to output mode

XGpioPs\_SetOutputEnablePin(&Gpio, EMIO\_LCD\_CS, 1);// Open the output enable

XGpioPs\_WritePin(&Gpio, EMIO\_LCD\_BL, 0);// 0 SET LOW

XGpioPs\_WritePin(&Gpio, EMIO\_LCD\_BL, 1);// 1 SET HIGH

**In order to simplify the writing of the code, use Define to simplify the operation of pull -up and pull low**

```
#define LCD_SDA_HIGH XGpioPs_WritePin(&Gpio, EMIO_LCD_SDA, 1)
```

```
#define LCD_SDA_LOW XGpioPs_WritePin(&Gpio, EMIO_LCD_SDA, 0)
```

```
#define LCD_SCL_HIGH XGpioPs_WritePin(&Gpio, EMIO_LCD_SCL, 1)
```

```
#define LCD_SCL_LOW XGpioPs_WritePin(&Gpio, EMIO_LCD_SCL, 0)
```

```
#define LCD_BLK_HIGH XGpioPs_WritePin(&Gpio, EMIO_LCD_BLK, 1)
```

```
#define LCD_BLK_LOW XGpioPs_WritePin(&Gpio, EMIO_LCD_BLK, 0)
```

```
#define LCD_CD_HIGH XGpioPs_WritePin(&Gpio, EMIO_LCD_CD, 1)
```

```
#define LCD_CD_LOW XGpioPs_WritePin(&Gpio, EMIO_LCD_CD, 0)
```

```
#define LCD_RES_HIGH XGpioPs_WritePin(&Gpio, EMIO_LCD_RES, 1)
```

```
#define LCD_RES_LOW XGpioPs_WritePin(&Gpio, EMIO_LCD_RES, 0)
```

IO simulation SPI code compilation (CS signal pin increases here)

```
void delay_spi_nop(){
```

```
    volatile int Delay;
```

```
    for (Delay = 0; Delay < 1; Delay++);
```

```
}
```

```
void spi_send(unsigned char dat){
```

```
    unsigned char i;
```

```
    LCD_CS_LOW;
```

```
    for(i=0;i<8;i++){
```

```
        LCD_SCL_LOW;
```

```

delay_spi_nop();

if(dat&0x80)LCD_SDA_HIGH;

else LCD_SDA_LOW;

delay_spi_nop();

LCD_SCL_HIGH;

delay_spi_nop();

dat=dat<<1;

}

LCD_CS_HIGH;

}

```

A loop in the delay\_spi\_nop () function is just to generate a short -term delay, similar to the \_Nop\_ () of a single -chip microcomputer;

7) After integrating the code as follows, the following is the complete screen drive code, copy the code to the main.c

```

#include "xparameters.h"

#include "xgpiops.h"

#include "xstatus.h"

#include "xplatform_info.h"

#include <xil_printf.h>

```

```
#define WHITE          0xFFFFF

#define BLACK           0x0000

#define BLUE            0x001F

#define BRED             0XF81F

#define GRED             0XFFE0

#define GBLUE            0X07FF

#define RED              0xF800

#define MAGENTA          0xF81F

#define GREEN             0x07E0

#define CYAN              0x7FFF

#define YELLOW            0xFFE0

#define BROWN             0XBC40

#define BRRED             0XFC07

#define GRAY              0X8430

#define EMIO_LCD_CS      54

#define EMIO_LCD_CD      55

#define EMIO_LCD_RES     56

#define EMIO_LCD_SCL     57

#define EMIO_LCD_SDA     58
```

```
#define GPIO_DEVICE_ID      XPAR_XGPIOPS_0_DEVICE_ID

#define SPI_DEVICE_ID          XPAR_XSPIPS_0_DEVICE_ID

XGpioPs Gpio;           /* The driver instance for GPIO
Device. */

#define LCD_SDA_HIGH XGpioPs_WritePin(&Gpio,
EMIO_LCD_SDA, 1)

#define LCD_SDA_LOW  XGpioPs_WritePin(&Gpio, EMIO_LCD_SDA,
0)

#define LCD_SCL_HIGH XGpioPs_WritePin(&Gpio,
EMIO_LCD_SCL, 1)

#define LCD_SCL_LOW  XGpioPs_WritePin(&Gpio, EMIO_LCD_SCL,
0)

#define LCD_CS_HIGH XGpioPs_WritePin(&Gpio, EMIO_LCD_CS,
1)

#define LCD_CS_LOW  XGpioPs_WritePin(&Gpio, EMIO_LCD_CS,
0)

#define LCD_CD_HIGH XGpioPs_WritePin(&Gpio, EMIO_LCD_CD,
1)
```

```
#define LCD_CD_LOW XGpioPs_WritePin(&Gpio, EMIO_LCD_CD,  
0)  
  
#define LCD_RES_HIGH XGpioPs_WritePin(&Gpio,  
EMIO_LCD_RES, 1)  
  
#define LCD_RES_LOW XGpioPs_WritePin(&Gpio, EMIO_LCD_RES,  
0)  
  
  
  
  
void delay_spi_nop() {  
  
    volatile int Delay;  
  
    for (Delay = 0; Delay < 1; Delay++);  
  
}  
  
  
  
  
void spi_send(unsigned char dat) {  
  
    unsigned char i;  
  
    LCD_CS_LOW;  
  
    for(i=0;i<8;i++) {  
  
        LCD_SCL_LOW;  
  
        delay_spi_nop();  
  
        if(dat&0x80) LCD_SDA_HIGH;  
  
        else LCD_SDA_LOW;  
    }  
}
```

```
    delay_spi_nop();

    LCD_SCL_HIGH;

    delay_spi_nop();

    dat=dat<<1;

}

LCD_CS_HIGH;

}

void Lcd_Gpio_Init(void) {

    XGpioPs_Config *ConfigPtr;

    ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);

    XGpioPs_CfgInitialize(&Gpio,
ConfigPtr,ConfigPtr->BaseAddr);

    XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_CS, 1);

    XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_CS, 1);

    XGpioPs_WritePin(&Gpio, EMIO_LCD_CS, 0);

    XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_CD, 1);
```

```
XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_CD, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_CD, 1);

XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_RES, 1);

XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_RES, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_RES, 1);

XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_SCL, 1);

XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_SCL, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_SCL, 1);

XGpioPs_SetDirectionPin(&Gpio, EMIO_LCD_SDA, 1);

XGpioPs_SetOutputEnablePin(&Gpio, EMIO_LCD_SDA, 1);

XGpioPs_WritePin(&Gpio, EMIO_LCD_SDA, 1);

}
```

```
void delay(unsigned int i){  
    volatile int Delay;  
  
    volatile int k;  
  
    for(k=0;k<i;k++)  
  
        for (Delay = 0; Delay < 10000; Delay++);  
  
}  
  
  
  
  
void LCD_WR_DATA8(u8 dat){  
  
    LCD_CD_HIGH;  
  
    spi_send(dat);  
  
}  
  
  
  
  
void LCD_WR_REG(u8 dat){  
  
    LCD_CD_LOW;  
  
    spi_send(dat);  
  
}  
  
  
  
  
void Lcd_Init(void){  
  
    LCD_RES_HIGH;  
  
    delay(500);
```

```
LCD_RES_LOW;

delay(500);

LCD_RES_HIGH;

delay(500);

LCD_WR_REG(0x36);

LCD_WR_DATA8(0x00);

LCD_WR_REG(0x3A);

LCD_WR_DATA8(0x05);

LCD_WR_REG(0xB2);

LCD_WR_DATA8(0x0C);

LCD_WR_DATA8(0x0C);

LCD_WR_DATA8(0x00);

LCD_WR_DATA8(0x33);

LCD_WR_DATA8(0x33);

LCD_WR_REG(0xB7);

LCD_WR_DATA8(0x35);

LCD_WR_REG(0xBB);

LCD_WR_DATA8(0x19);

LCD_WR_REG(0xC0);

LCD_WR_DATA8(0x2C);
```

```
LCD_WR_REG(0xC2);  
  
LCD_WR_DATA8(0x01);  
  
LCD_WR_REG(0xC3);  
  
LCD_WR_DATA8(0x12);  
  
LCD_WR_REG(0xC4);  
  
LCD_WR_DATA8(0x20);  
  
LCD_WR_REG(0xC6);  
  
LCD_WR_DATA8(0x0F);  
  
LCD_WR_REG(0xD0);  
  
LCD_WR_DATA8(0xA4);  
  
LCD_WR_DATA8(0xA1);  
  
LCD_WR_REG(0xE0);  
  
LCD_WR_DATA8(0xD0);  
  
LCD_WR_DATA8(0x04);  
  
LCD_WR_DATA8(0x0D);  
  
LCD_WR_DATA8(0x11);  
  
LCD_WR_DATA8(0x13);  
  
LCD_WR_DATA8(0x2B);  
  
LCD_WR_DATA8(0x3F);  
  
LCD_WR_DATA8(0x54);
```

```
LCD_WR_DATA8 (0x4C);  
  
LCD_WR_DATA8 (0x18);  
  
LCD_WR_DATA8 (0x0D);  
  
LCD_WR_DATA8 (0x0B);  
  
LCD_WR_DATA8 (0x1F);  
  
LCD_WR_DATA8 (0x23);  
  
LCD_WR_REG (0xE1);  
  
LCD_WR_DATA8 (0xD0);  
  
LCD_WR_DATA8 (0x04);  
  
LCD_WR_DATA8 (0x0C);  
  
LCD_WR_DATA8 (0x11);  
  
LCD_WR_DATA8 (0x13);  
  
LCD_WR_DATA8 (0x2C);  
  
LCD_WR_DATA8 (0x3F);  
  
LCD_WR_DATA8 (0x44);  
  
LCD_WR_DATA8 (0x51);  
  
LCD_WR_DATA8 (0x2F);  
  
LCD_WR_DATA8 (0x1F);  
  
LCD_WR_DATA8 (0x1F);  
  
LCD_WR_DATA8 (0x20);
```

```
LCD_WR_DATA8(0x23);

LCD_WR_REG(0x21);

LCD_WR_REG(0x11);

LCD_WR_REG(0x29);

}

void LCD_WR_DATA(u16 dat)

{

    u8 spi_dat;

    LCD_CD_HIGH;

    spi_dat=dat>>8;

    spi_send(spi_dat);

    spi_dat=dat;

    spi_send(spi_dat);

}

void Address_set(unsigned int x1,unsigned int y1,unsigned
int x2,unsigned int y2)
```

```
{  
  
LCD_WR_REG(0x2a);  
  
LCD_WR_DATA8(x1>>8);  
  
LCD_WR_DATA8(x1);  
  
LCD_WR_DATA8(x2>>8);  
  
LCD_WR_DATA8(x2);  
  
LCD_WR_REG(0x2b);  
  
LCD_WR_DATA8(y1>>8);  
  
LCD_WR_DATA8(y1);  
  
LCD_WR_DATA8(y2>>8);  
  
LCD_WR_DATA8(y2);  
  
LCD_WR_REG(0x2C);  
  
}  
  
void LCD_Test()  
{  
    unsigned int i,j;  
  
    Address_set(0,0,240-1,240-1);
```

```
for(i=0;i<240;i++) {  
  
    if(i>=0&&i<60)  
  
        for (j=0;j<240;j++) LCD_WR_DATA(WHITE);  
  
    else if(i>=60&&i<120)  
  
        for (j=0;j<240;j++) LCD_WR_DATA(RED);  
  
    else if(i>=120&&i<180)  
  
        for (j=0;j<240;j++) LCD_WR_DATA(GREEN);  
  
    else if(i>=180&&i<240)  
  
        for (j=0;j<240;j++) LCD_WR_DATA(BLUE);  
  
    }  
  
}  
  
unsigned char k=0;  
  
void LCD_Test2()  
  
{
```

```
unsigned int i,j;

Address_set(0,0,240-1,240-1);

if(k==0) {

    for(i=0;i<240;i++) {

        for (j=0;j<240;j++) LCD_WR_DATA(WHITE);

    }

    k=1;

}

else {

    for(i=0;i<240;i++) {

        for (j=0;j<240;j++) LCD_WR_DATA(BLUE);

    }

    k=0;

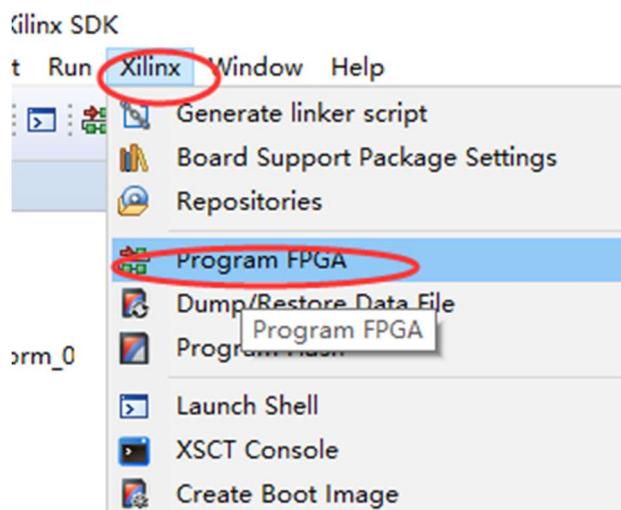
}

int main(void)
```

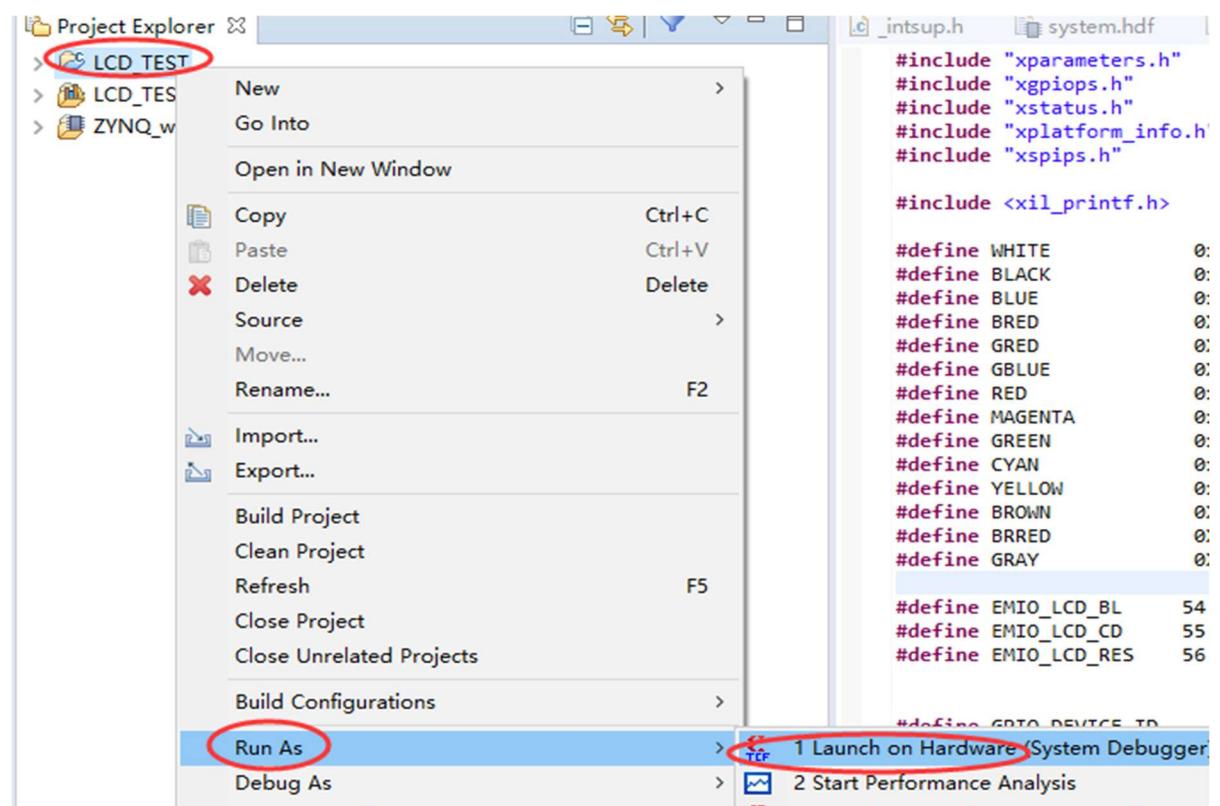
```
{  
  
    Lcd_Gpio_Init();  
  
    Lcd_Init();  
  
    LCD_Test();  
  
  
  
  
    while(1) {  
  
        //LCD_Test2();  
  
    };  
  
  
  
  
    return XST_SUCCESS;  
  
}  

```

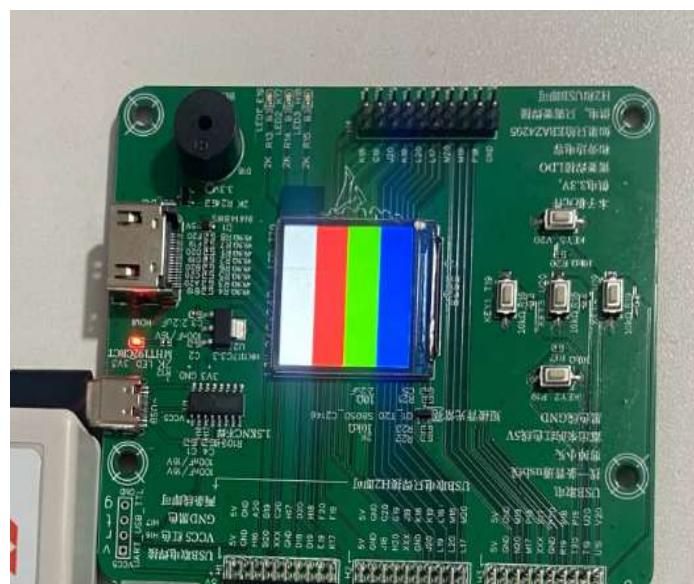
- 8) Use the binary files generated before, programming FPGA, Xilinx Tools-> Program FPGA and then click "Program"



9) When FPGA programming is successful, we need to initialize the processor in Zynq. Right -click the air engineering just created, select Run As-> Launch on Hardware (System Debugger) or Launch On Hardware (GDB).



After the above operations, the screen was successfully lit, and the 4 -color stripes were displayed normally, as shown in the figure below



## Code explanation

The above is the complete engineering graphic creation process. In addition to the GPIO code mentioned above, the following briefly introduces some of the rest of the functional code in the project

### 1) About color

The screen itself is 16 colors, that is, the color driving method of the RGB565, so you can use 16 bytes to define different colors, such as red, green, blue and white in the program, and other colors marked below. To synthesize the color you need

```
#define WHITE          0xFFFFF  
  
#define BLACK         0x0000  
  
#define BLUE          0x001F  
  
#define BRED          0XF81F  
  
#define GRED          0XFFE0  
  
#define GBLUE         0X07FF  
  
#define RED           0xF800  
  
#define MAGENTA       0XF81F  
  
#define GREEN         0x07E0  
  
#define CYAN          0x7FFF  
  
#define YELLOW        0xFFE0  
  
#define BROWN         0XBC40  
  
#define BRRED         0XFC07
```

```
#define GRAY 0X8430
```

2)Regarding the color stripes, as follows, the following code is displayed at 0-60 lines, 60-120 shows red, 120-180 displays green, 180-240 shows blueAddress\_Set (0,0,240-1,240-1); equivalent to defining a rectangular area, 0, 0, 240, 240 is the four coordinate points of this rectangular (here cover the space of the entire screen, and you can customize some areas. Look at the screen manual in detail)

```
void LCD_Test()

{

    unsigned int i,j;

    Address_set(0,0,240-1,240-1);

    for(i=0;i<240;i++) {

        if(i>=0&&i<60)

            for (j=0;j<240;j++) LCD_WR_DATA(WHITE);

        else if(i>=60&&i<120)

            for (j=0;j<240;j++) LCD_WR_DATA(RED);

        else if(i>=120&&i<180)

            for (j=0;j<240;j++) LCD_WR_DATA(GREEN);

    }

}
```

```

        else if(i>=180&&i<240)

            for (j=0;j<240;j++) LCD_WR_DATA(BLUE);

    }

}

```

Finally, I introduce the main function, which is particularly simple, including the initialization of GPIO, and the initialization of the screen (the configuration information of the internal register inside the screen is directly provided by the manufacturer), and the call of the screen test function call

```

int main(void)

{

    Lcd_Gpio_Init();

    Lcd_Init();

    LCD_Test();

    while(1);

    return XST_SUCCESS;

}

```

PS introduced the complete method of simulating SPI with the IO port to light up the LCD screen. The hardware SPI does not consume the CPU resources during the transmission. The software SPI transmission will consume the CPU system resources