

# HELLO, FPGA

Document my FPGA learning journey

APRIL 2023, 4 BY ACKYE

## Smart ZYNQ (SP&SL Version) Project II Design flow lights with ZYNQ's PL (FPGA).

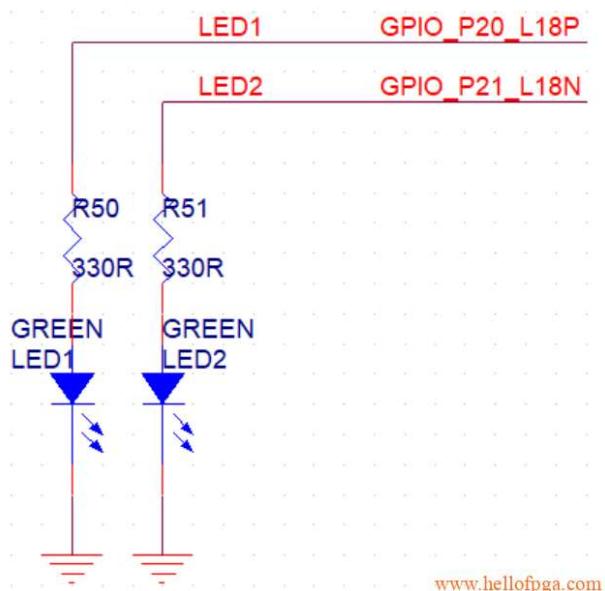
Project 1 completely introduces the whole process of a project from creation to compilation and synthesis, and finally to program operation, this project is further deepened on the basis of Project <>, and two LED lights are lit in the form of running lamps

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)**

### 1. Schematic analysis

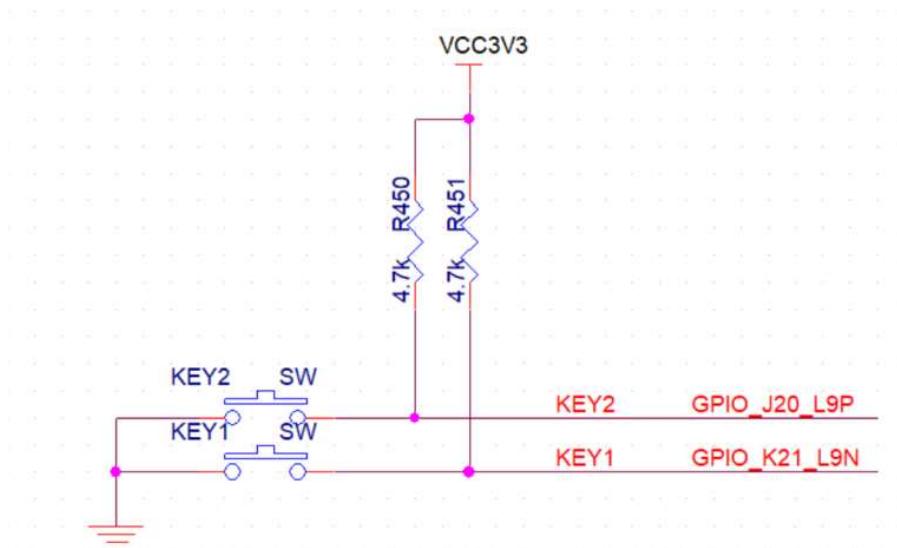
**LED light:** Two pieces of information can be seen from the circuit diagram below



[www.hellofpga.com](http://www.hellofpga.com)

- 1) The cathode of the two LED lights is connected to GND, then the pin output of the FPGA connected to the LED is high, the LED light is on, and the pin output of the FPGA connected to the LED is low-level LED off
- 2) The LED lights are connected to the P20 and P21 pins of the main chip (PL) part

**Buttons:** There are two buttons on the board, and we use one of the buttons connected to the board (KEY1) as a reset function (generally normal engineering requires a reset key, or there is no reset key like Project 1) KE21 is connected to the K<> pin



## 2. Programming

There are many ways to drive LEDs

## 1) Direct drive mode, relatively simple as follows

```
module LED (
    output LED1,
    output LED2
);
    assign LED1=1'b0;
    assign LED2=1'B1;

endmodule
```

This method is equivalent to connecting the output port of the LED directly to the VCC through the assign method, and GND, as in the above code, LED1 is connected to GND and is in the off state, and LED2 is connected to the internal VCC and is in the lit state

This code is relatively simple, so I won't expand it

## 2) Next, introduce a counter, use a counter to light two LED lights respectively to achieve the effect of running lights

Since our board is soldered with a 50Mhz crystal oscillator, the crystal oscillates 50\_000\_000 times per second, and the calculator looks at 50000000 to convert to binary to occupy a total of 26 bits, so here we define a 26-bit register for counter



```
reg [25:0]time_count; // [25:0]代表26bit
```

## 1 second counter module

The next step is to write a finished 1-second counter module

```
parameter T1MS = 26'd50_000_000 ; //50M晶振时钟
reg [25:0]time_count;
always@ (posedge CLK or negedge RSTn)
    if(!RSTn)begin//当复位条件下，计数器赋初值0
        time_count<=26'd0;
    end
    else begin//当不在复位条件下
        if(time_count>=T1MS-1'b1)//如果计数器达到 50_000_000则代表一秒钟计数完成
            time_count<=26'd0;
        else time_count<=time_count+1'b1;//其他情况下计数器自增
    end
```

where parameter T1MS = 26'd50\_000\_000; To define a constant, the crystal oscillates 50\_1\_50 times in 000 second under the 000M crystal oscillator mentioned earlier

always@ (posedge CLK or negedge RSTn) represents the two conditions for the module to enter, one is the rising edge of the system clock (posedge), and the other is the falling edge of the reset signal (negedge), that is, when the system reset signal is pressed, each rising edge of the clock enters and executes the content in the always@ block

Under reset conditions if(! RSTn) The system initializes the counter module to 0

When the system is not in the initialization condition, each clock rising edge time\_count increments by 1, and when time\_count >= T1MS -1'b1, that is, when the count reaches 1 second, the time\_count clears again to zero

## LED light switching module

```
reg [1:0]led_state;
always@ (posedge CLK or negedge RSTn)
    if(!RSTn)begin//当复位条件下，计数器赋初值0
        led_state<=2'd0;
    end
    else begin
```

```

        if(time_count==T1MS-1'b1)begin//一秒钟
            if(led_state>=2'd1)led_state<=2'd0;//从0-2反复循环
            else led_state<=led_state+1'b1;//自增
        end
    end

    assign LED1= (led_state==2'd0)?1'b1:1'b0;
    assign LED2= (led_state==2'd1)?1'b1:1'b0;

```

Here defines a 2-bit led\_state register to control the display status of the current lamp, as can be seen from the program, whenever time\_count==T1MS-1'b1, led\_state repeatedly loops at 0-1 (if there are multiple lamps, you can modify it here, for example, 3 lamps can be modified to 0-1-2, and the led\_state width needs to be modified in the case of multiple lamps).

led\_state cycle repeatedly from 0-1, each state corresponds to the display of an LED light, such as led\_state = 0, LED1 is on, when led\_state = 1, LED2 is on, and so on.

The complete project code is as follows

```

`timescale 1ns / 1ps
module LED(
    input CLK,//时钟
    input RSTn,//复位
    output LED1,
    output LED2
);

parameter T1MS = 26'd50_000_000 ; //50M晶振时钟
reg [25:0]time_count;
always@ (posedge CLK or negedge RSTn)
    if(!RSTn)begin//当复位条件下，计数器赋初值0
        time_count<=26'd0;
    end
    else begin//当不在复位条件下
        if(time_count>=T1MS-1'b1)//如果计数器达到 50_000_000则代表一秒钟计数完成
            time_count<=26'd0;
        else time_count<=time_count+1'b1;//其他情况下计数器自增
    end
end

```

```

reg [1:0]led_state;
always@ (posedge CLK or negedge RSTn)
    if (!RSTn)begin//当复位条件下，计数器赋初值0
        led_state<=2'd0;
    end
    else begin
        if (time_count==T1MS-1'b1)begin//一秒钟
            if(led_state>=2'd1)led_state<=2'd0;//从0-2反复循环
            else led_state<=led_state+1'b1;//自增
        end
    end

assign LED1= (led_state==2'd0)?1'b1:1'b0;
assign LED2= (led_state==2'd1)?1'b1:1'b0;

endmodule

```

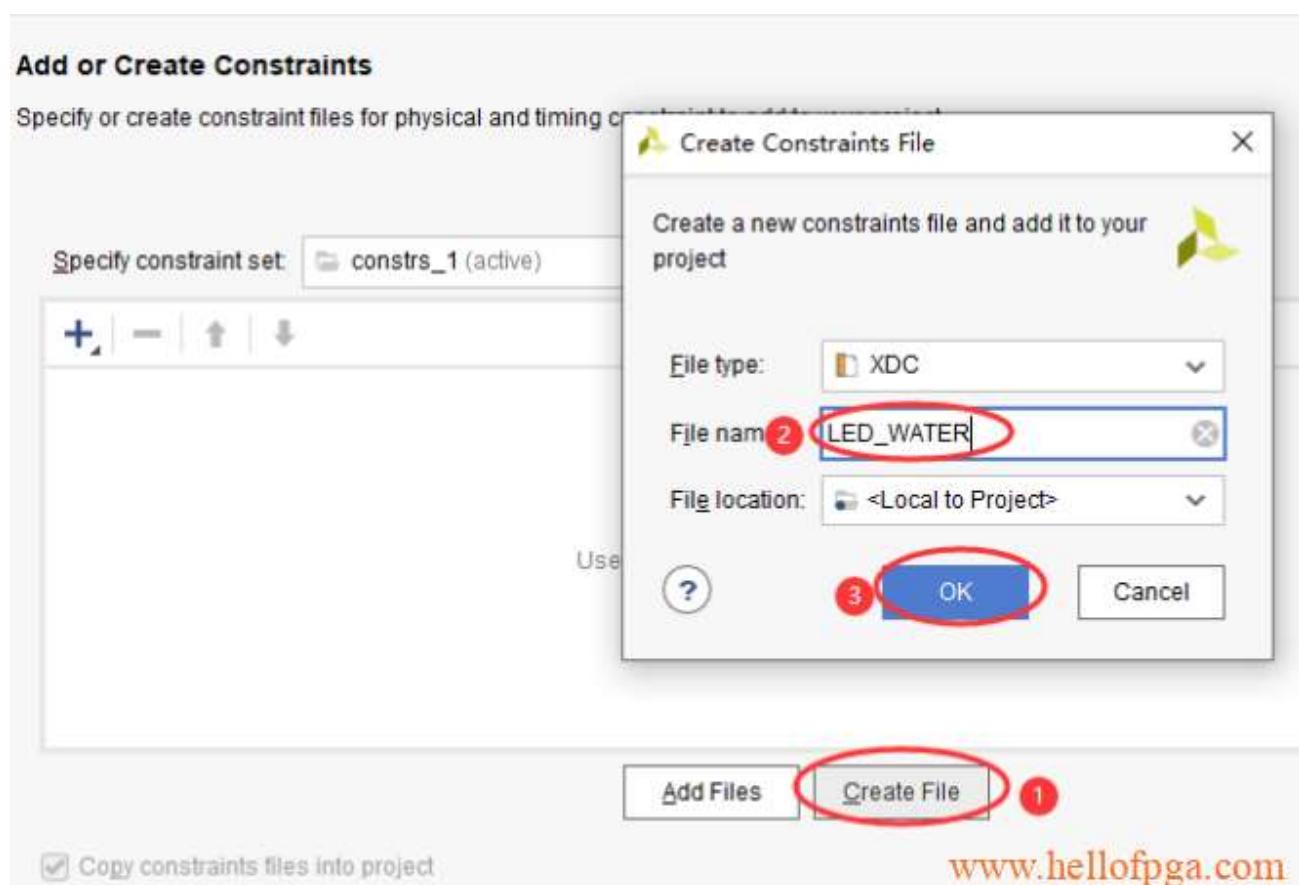
### 3. Pin constraint file creation

The previous part and the creation of the first project are roughly the same, you can refer to the project has, constraint file part of the project one through the graphical interface settings, automatically generated, this project we use different ways to create constraint files, the specific operation is as follows:

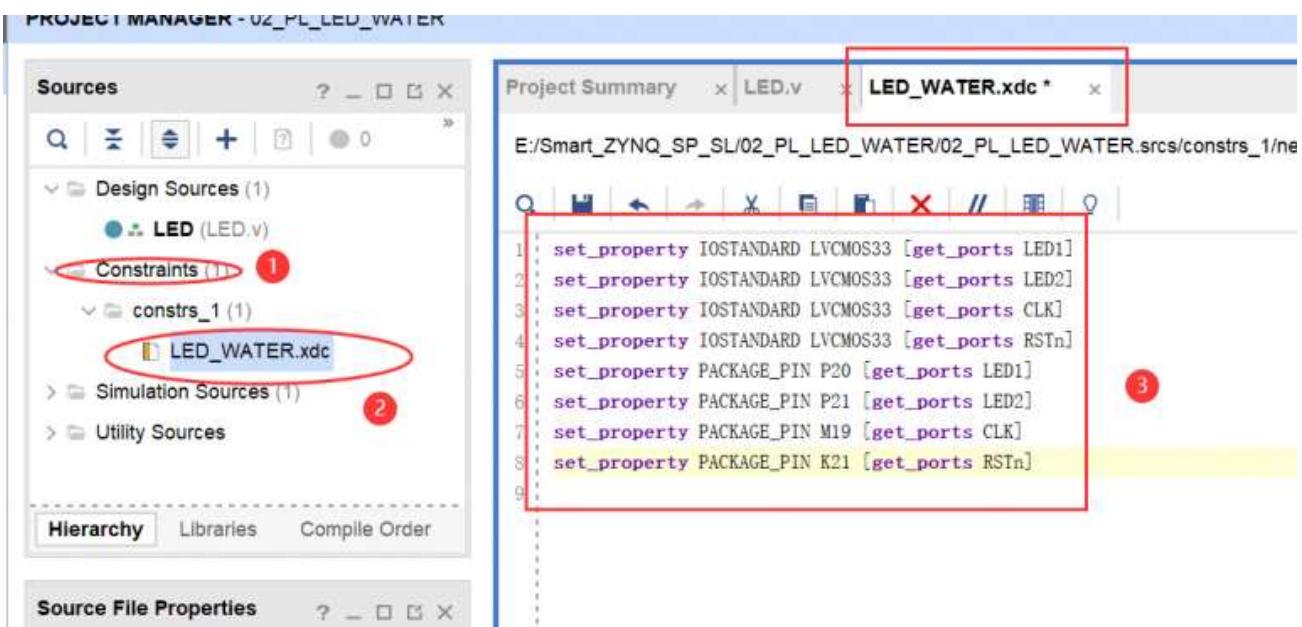
- Create a new constraint file



- b. In the pop-up window, click Create File and fill in the file name of the constraint file, then click OK and finish



- c. Open the constraint file as shown below



d. Fill in the constraint file with constraint information (pin definition, pin voltage, etc.)

```

set_property IOSTANDARD LVCMOS33 [get_ports LED1]
set_property IOSTANDARD LVCMOS33 [get_ports LED2]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports RSTn]
set_property PACKAGE_PIN P20 [get_ports LED1]
set_property PACKAGE_PIN P21 [get_ports LED2]
set_property PACKAGE_PIN M19 [get_ports CLK]
set_property PACKAGE_PIN K21 [get_ports RSTn]

```

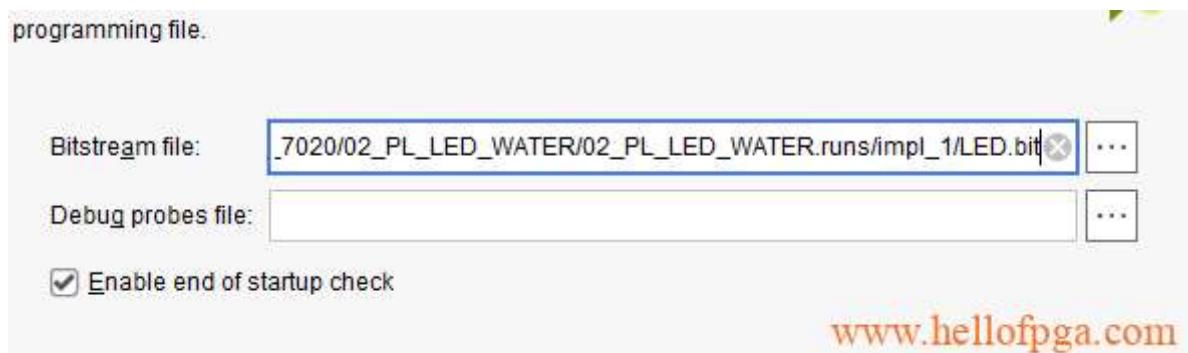
### 3. Project compilation, synthesis and download

Finally, the program is compiled and synthesized,



And use the TYPE C cable to connect the JTAG port to download the program to the board (the steps can refer to project 1), you can see that the two lights on the adapter board flash at an interval frequency of one second, and press KE<> (program-defined reset button) The light returns to the initial state

Note: If the bit file does not appear when downloading, you can add the bit file manually



Complete project download (**there are multiple versions of the board, please download and test according to the chip model of the corresponding board, corresponding version VIVADO2018.3**).

[02\\_PL\\_LED\\_WATER\\_XC7Z020](#) **Download**

SMART ZYNQ SP & SL