APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL VERSION) Project 4 PL partial button function demonstration (IO input function)

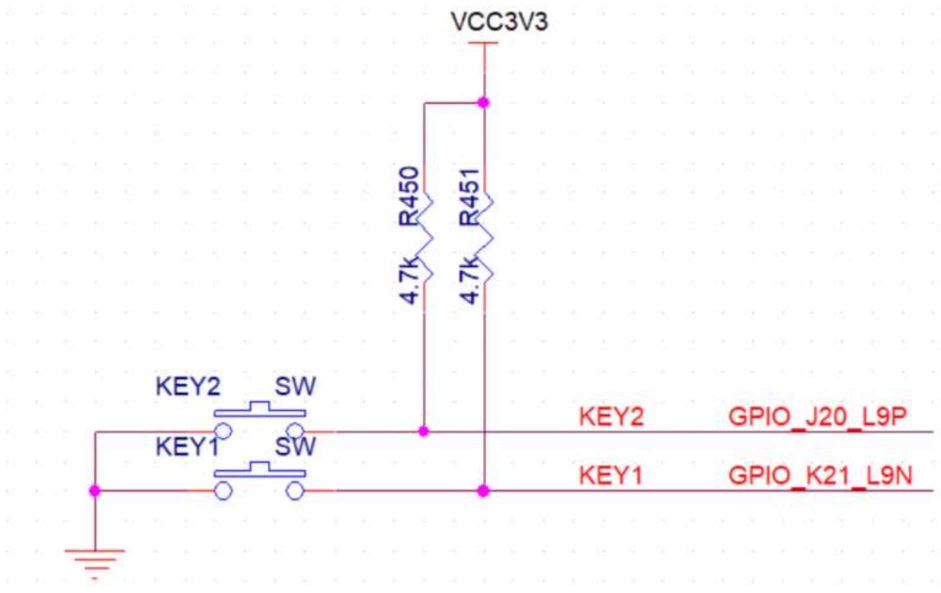
This chapter describes how to read the key operation to demonstrate the IO input function in the PL section

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

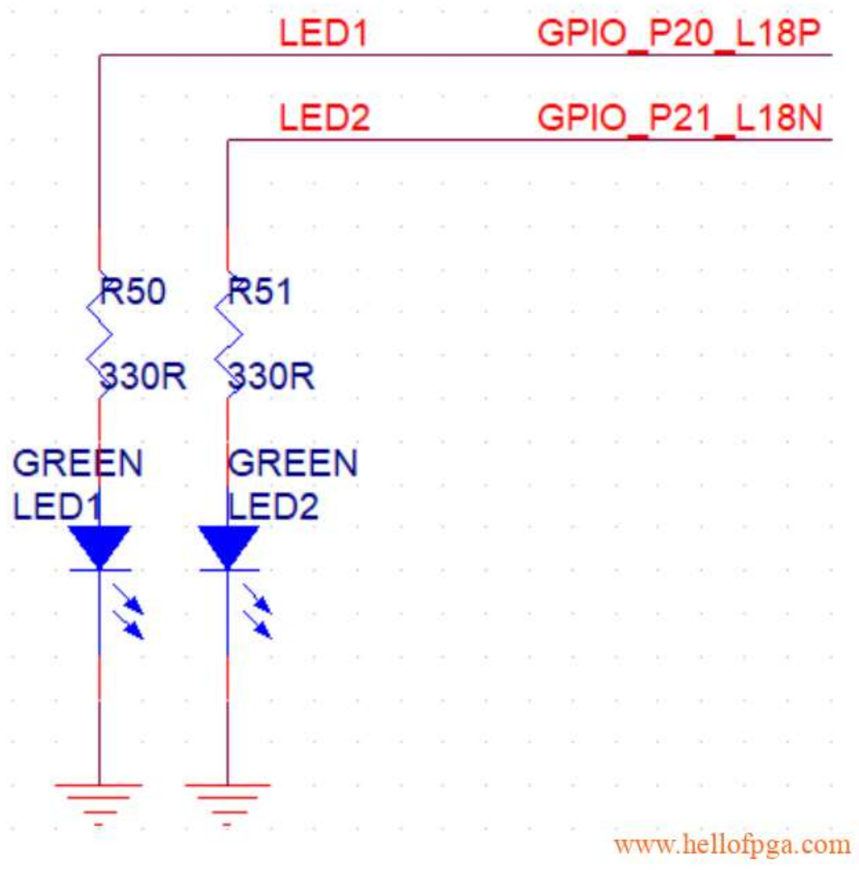
(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)

1. Hardware introduction

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



2. Writing code

The complete project creation has been described in detail before, which is skipped here and directly introduces the program.

Normal button operation, at the moment of pressing The level will produce jitter, although it is only a pressed operation, but the actual voltage may jump countless times, so the normal key code needs to be debounced, and debounce will be slightly troublesome Here is put at the end of this article

2.1 **In order to facilitate the understanding of the concept of input and output, here is a simple write of a button without debounce to control the program of the LED light** (generally the communication between chips does not need to do the debounce function, only the button needs to do debounce)

First of all, the original LED driver, we used output, output represents output, so here our button should be set to input, representing input

The complete code is relatively simple and will not be described in detail, see below for details

The program is run by timing **logic** synchronized by the clock signal, where the change of LED light and the change of KEY is only done on the rising edge of the clk and is affected by the clock

```
`timescale 1ns / 1ps
module KEY_TEST(
    input clk,
    output  LED1,
    output  LED2,
    input KEY1,
    input KEY2
);

reg LED1_r=0;
reg LED2_r=0;

always@(posedge clk)begin
    if(KEY1==0)LED1_r<=1'b1;
    else LED1_r<=1'B0;

    if(KEY2==0)LED2_r<=1'b1;
    else LED2_r<=1'B0;
end

assign LED1=LED1_r;
assign LED2=LED2_r;

endmodule
```

The constraint file is as follows

```
set_property PACKAGE_PIN P20 [get_ports LED1]
set_property PACKAGE_PIN P21 [get_ports LED2]
set_property PACKAGE_PIN K21 [get_ports KEY1]
set_property PACKAGE_PIN J20 [get_ports KEY2]
set_property PACKAGE_PIN M19 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports LED1]
set_property IOSTANDARD LVCMOS33 [get_ports LED2]
set_property IOSTANDARD LVCMOS33 [get_ports KEY1]
set_property IOSTANDARD LVCMOS33 [get_ports KEY2]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
```

After saving, compile and synthesize, and finally download to the board for running, you can see that when the left button is pressed, the left light is lit, and when the right button is pressed, the right light is lit

At this point, the simple input function is introduced.

Complete Project:

04_PL_KEY_XC7Z020 **Download**

Key control can also be combined logic **(not recommended),** the state of the LED light is only related to the button and clock and so on, this way is equivalent to the FPGA internal pull a wire and string a non-gate between the LED and the button

LED1=(KEY1==0)?1’b1:1’b0;

This method also does not require the creation of the corresponding REG register (LED_r) of the LED, and the disadvantage is that the individual signals are not synchronized and is not suitable for large systems

2.2 The above method does not have key debounce, suitable for transmission between different chips, and simple key applications, if you want to make the stability of the keys in the system greater, you need to add additional key debounce function in the system

Here's how to describe key shake

Reasons for key debounce:

Usually when a key switch is closed due to mechanical characteristics, the button will not be immediately turned on, and it will not be disconnected at once, but it will be accompanied by multiple jitter at the moment of closure and disconnection, and the generation of jitter will interfere with the signal of the button itself.

Generally, the normal key jitter process is less than 20ms, while the normal one-time key operation, even if it is a short press time, is more than 50ms. So our key debounce time can be bounded by 20ms.

Key debounce ideas:

A. Initial State counter is 0, jump to state B

b. Detects that the key press (i.e. the level is 0) and the counter starts timing and jumps to state C

c. The counter counts to 20ms before continuous detection, detects that the voltage value is not 0, that is, it is considered to be a jitter signal, then the system returns to the initial state of A, if the voltage value clock is 20 within 0ms, the button is considered valid and jumps to state D

d. The active signal of the key is set, when the key level is detected that the key level is not 0, the active signal of the button is cleared to 0

(The above is to judge the debounce processing idea of the key press, if the release of the button does not participate in the decision, there is no need to do the key release debounce, otherwise the key release also needs to do the debounce treatment).

Next, start writing the debounced program:

```
`timescale 1ns / 1ps
module KEY_TEST(
    input clk,
    output LED,
    input KEY
);

reg [1:0]debounce_mode=2'd0;
reg [19:0]debounce_count=20'd0;

reg [1:0]KEY_r=0;
always@(posedge clk)begin
    KEY_r[1]<= KEY_r[0];
    KEY_r[0]<= KEY;
end
wire KEY_NEGEDGE=(KEY_r[1]&~KEY_r[0])?1'b1:1'b0;


reg KEY_value=0;
always@(posedge clk)begin
    if(debounce_mode==0)begin
        debounce_count<=20'd0;
        debounce_mode<=2'd1;
        KEY_value<=0;
    end
    else if(debounce_mode==1)begin
        if(KEY_NEGEDGE==1)begin
            debounce_count<=20'd0;
            debounce_mode<=2'd2;
        end
    end
    else if(debounce_mode==2)begin
        if(debounce_count>=20'd1_000_000)debounce_mode<=2'd3;
        else begin
            if(KEY==1)debounce_mode<=2'd0;
            debounce_count<=debounce_count+1'b1;
        end
    end
    else begin
        if(KEY==1)debounce_mode<=2'd0;
```

```
            KEY_value<=1'b1;
        end
    end
```

Simple interpretation of the code

The code uses two registers to shift the key state of the KEY, and then find the falling edge of the key by comparing the key changes before and after (the previous moment is high, the next moment is low, representing the falling edge, that is, (KEY_r[1]&~KEY_r[0])), when the system is the falling edge KEY_NEGEDGE output high

```
reg [1:0]KEY_r=0;
always@(posedge clk)begin
    KEY_r[1]<= KEY_r[0];
    KEY_r[0]<= KEY;
end
wire KEY_NEGEDGE=(KEY_r[1]&~KEY_r[0])?1'b1:1'b0;
```

debounce_mode represents the state in which the current state machine is located, where 0, 1, 2, and 3 represent the 4 process steps of A, B, C, and D, respectively, of the key debounce idea written earlier, which can be seen accordingly. 0 represents initialization, 1 mode detects the falling edge, if the falling edge enters mode 2, 2 starts counting and detects whether the key state becomes 1, if it is, it is considered that the trigger is jitter interference, then return to mode 0 to wait for the next falling edge, if the counter exceeds 20ms The key clock does not change, then the key is considered to be effectively entered into mode 3, and the output value of the key is changed under mode 3 KEY_value

debounce_count is a 20-bit counter that counts from the falling edge of the key until it stops at 20ms (20ms corresponds to the 50,1000000,<>th pulse of a <>MHz clock)

KEY_value represents the value of the key after final debounce, and KEY_value=1 indicates that the key is valid and lasts. Here the result of the key is assigned by LED=KEY_value; By mapping the value of the KEY_value register directly to the LED indicator, you can better see the result of the key press.

To add a constraint file:

```
set_property PACKAGE_PIN D18 [get_ports LED]
set_property PACKAGE_PIN G15 [get_ports KEY]
set_property PACKAGE_PIN K18 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports LED]
set_property IOSTANDARD LVCMOS33 [get_ports KEY]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk_IBUF}]
```

Only one button is demonstrated here, if you want to do multiple buttons, you can use the module instantiation way to design multiple buttons, please try it yourself

If you need to increase the key release debounce, the principle is the same, you can study it yourself

Here is the complete project with debounce:

04_PL_KEY_DEBOUNCE_XC7Z020 **Download**

SMART ZYNQ SP & SL