PRX100 USER EXPERIMENTAL MANUAL

PRX100 EXPERIMENTAL INSTRUCTIONS

FRASER INNOVATION INC

September 16, 2019

Version Control

Version	Date	Descrption
V1.0	10/07/2019	Initial Release
V1.1	16/09/2019	Modify part of pin assignments and Ethernet description

Contents:

Part 1 FII-PRX100 Development System Introduction 5

1. System Design Objective 5
2. System Resource 5
3. Human-computer Interaction Interface..5
4. Software Development System..6
5. Supporting Resources..6

Part 2 FII-PRX100 Main Hardware Resources Usage and FPGA Development Experiment 6

Experiment 1 LED Shifting..6

1. Experiment Object 6
2. Create A New Project Under Vivado 6

Experiment 2 Switches and display..25

1.Experiment Objective..25
2.Start New Project 25
3.Verilog HDL Code..25
4.FPGA Pin Assignment 26
5.Program in Vivado..27
6.Download to the develop board to test and dial the DIP switch to see the corresponding LED light on and off. See Fig 2. 2. 27
7.Use of ILA 27

Experiment 3 Basic Digital Clock Experiment and Programming of FPGA Configuration Files31

1.Experiment Objective..31
2.Design of The Experiment..31
3.FPGA Pin Assignment 36
4.Configure the Serial Flash Programming..37

Experiment 4 Block/SCH Digital Clock Design 43

1.Experiment Objective..43
2.Experiment Procedure 43
3.More to Practice..51

Experiment 5 Button Debounce Design and Experimental Experiment 52

1.Experiment Objective..52
2.Experiment..52
3.Hardware Design..57

Experiment 6 Digital Clock Comprehensive Design Experiment 59

1.Experiment Objective..59
2.Design Procedure 59
3.Create an XDC File to Constrain the Clock..70

Experiment 7 Multiplier Use and ISIM Simulation 72

1.Experiment Objective..72

2.Experiment Design 72
3.The Top-level Entity Is as Follows: 74
4.ISIM Simulation Library Compilation and Call 74
Experiment 8 Hexadecimal Number to BCD Code Conversion and Application 79
1.Experiment Objective..79
2.Application of Hexadecimal Number to BCD Number Conversion 83
3.Experiment Reflection..89
Experiment 9 Use of ROM..90
1.Experiment Objective..90
2.Experiment Design 90
3.Design Procedure 90
Experiment 10 Use Dual_port RAM to Read and Write Frame Data 99
1.Experiment Objective..99
2.Experiment Implement 99
3.Program Design 100
4.Lock the Pins, Compile, and Download to The Board to Test...................................107
5.Use ILA to Observe Dual_port RAM Read and Write 109
6.Experiment Summary and Reflection 110
Experiment 11 Asynchronous Serial Port Design and Experiment.......................................111
1.Experiment Objective..111
2.Experiment Implement 111
3.Experiment Design 111
Experiment 12 IIC Protocol Transmission...143
1.Experiment Objective..143
2.Experiment Implement 143
3.Introduction to the IIC Agreement...143
4. Main Code 144
5.Downloading to The Board 153
6.More to Practice...154
Experiment 13 AD, DA Experiment 155
1.Experiment Objective..155
2.Experiment Implement 155
3.Experiment Design 155
4.Downloading to The Board 168
Experiment 14 HDMI Graphic Display Experiment 170
1.Experiment Objective..170
2.Experiment Implement 170
3.Program Design 171
4.Download the Program to the Board to Test 178
References...180

Part 1 FII-PRX100 Development System Introduction

1. System Design Objective

The main purpose of this system design is to complete FPGA learning, development and experiment with Xilin-Vivado. The main device uses the Xilinx-XC7A100T-2FGG676I and is currently the latest generation of FPGA devices from Xilinx. The main learning and development projects can be completed as follows:

(1) Basic FPGA design training
(2) Construction and training of the SOPC (Microblaze) system
(3) IC design and verification, the system provides hardware design, simulation and verification of RISC-V CPU
(4) Development and application based on RISC-V
(5) The system is specifically optimized for hardware design for RISC-V system applications

2. System Resource

(1) Extended memory
(2) Use two Super SRAMs in parallel to form a 32-bit data interface with a maximum access space of 1M bytes.
(3) IS61WV25616 (2 pieces) 256K x 16bit
(4) Serial flash
(5) Spi interface serial flash (128M bytes)
(6) Serial EEPROM
(7) Gigabit Ethernet: 100/1000 Mbps
(8) USB to serial interface: USB-UART bridge

3. Human-computer Interaction Interface

(1) 8 toggle switches
(2) 8 push buttons
(3) Definition of 7 push buttons: up, down, left, right, ok, menu, return
(4) 1 for reset: Reset button
(5) 8 LEDs
(6) 6 7-segment decoders
(7) I2C bus interface
(8) UART external interface
(9) Two JTAG programming interfaces: One is for downloading the FPGA debug interface, and the other one is the JTAG debug interface for the RISC-V CPU
(10) Built-in RISC-V CPU software debugger, no external RISC-V JTAG emulator required
(11) 4 12-pin GPIO connectors, in line with PMOD interface standards

4. Software Development System

(1) Vivado 18.1 and later version for FPGA development, Microblaze SOPC
(2) Freedom Studio-Win_x86_64 Software development for RISC-V CPU

5. Supporting Resources

RISC-V JTAG Debugger
xilinx Altera JTAG Download Debugger
FII-PRX100 Development Guide

Part 2 FII-PRX100 Main Hardware Resources Usage and FPGA Development Experiment

This part mainly guides the user to learn the development of FPGA program and the use of onboard hardware through the development example of FPGA. At the same time, the application system software Xilinx is introduced from the elementary to the profound. The development exercises covered in this section are as follows:

Experiment 1 LED Shifting

1. Experiment Object

(1) Practice how to use the development system software Vivado to establish a new project, call the system resource PLL to establish the clock.
(2) Write Verilog HDL program to achieve frequency division
(3) Write Verilog HDL program to implement LED shifting
(4) Combine hardware resources for FPGA pin configuration
(5) Compile
(6) Download the program to the develop board
(7) Observe the experimental result and debug the project

2. Create A New Project Under Vivado

(1) Start Vivado in the start Menu. See Fig 1. 1

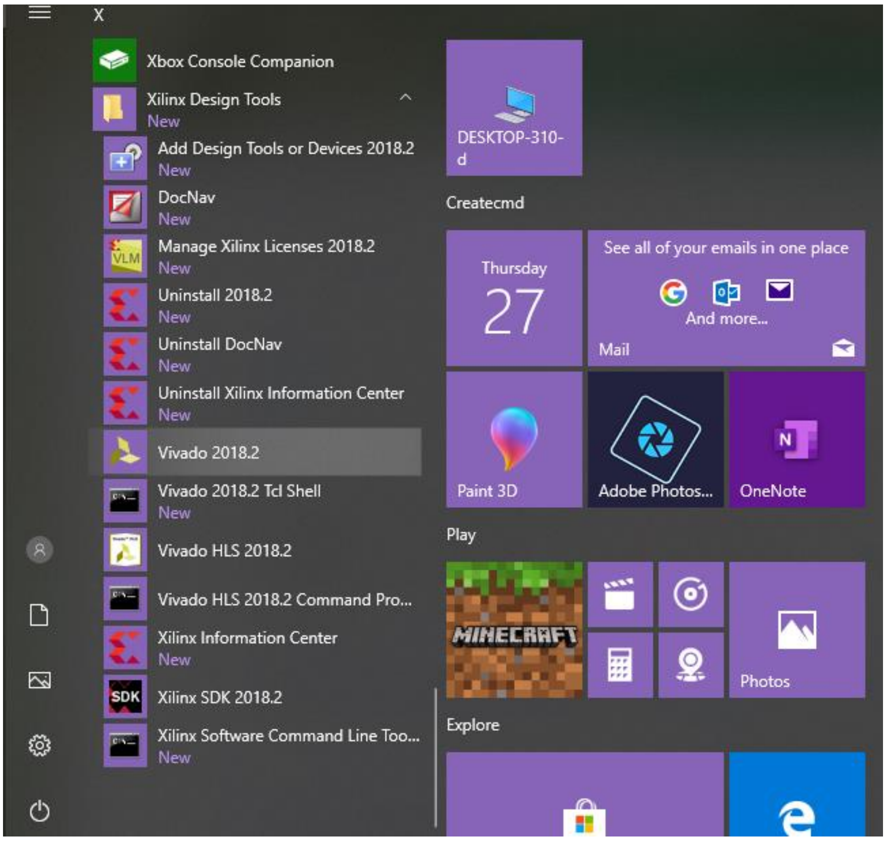


Fig 1. 1 Start Menu

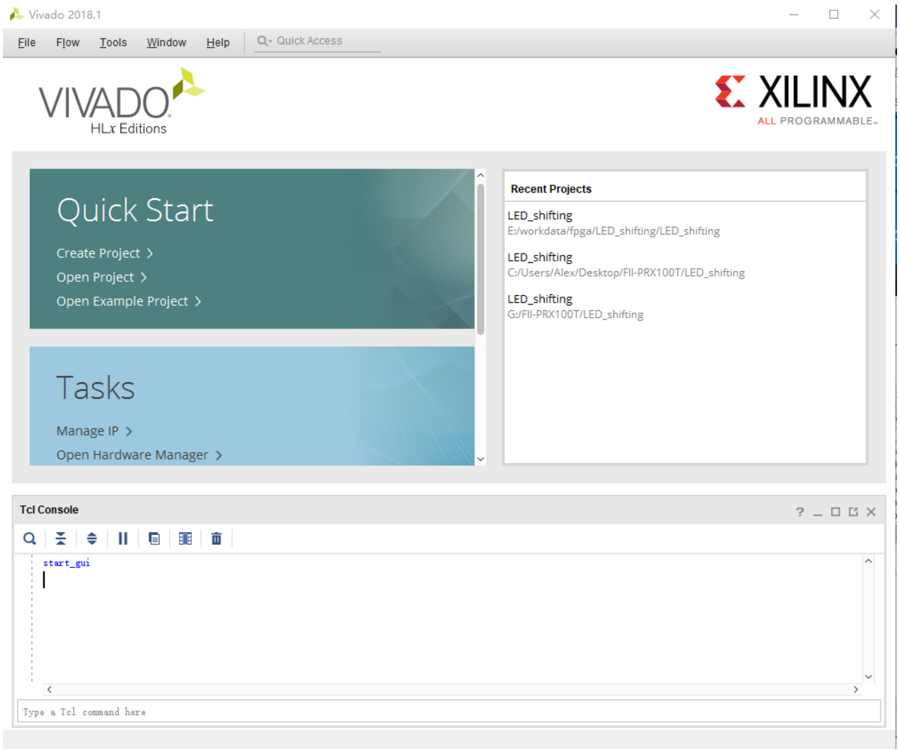


Fig 1. 2 Initial interface of Vivado

(2) **File -> Project -> NEW**

a. Click the **Next** option button in the pop-up dialog box. Then pop up the setup project interface of Fig 1. 3 and Fig 1. 4

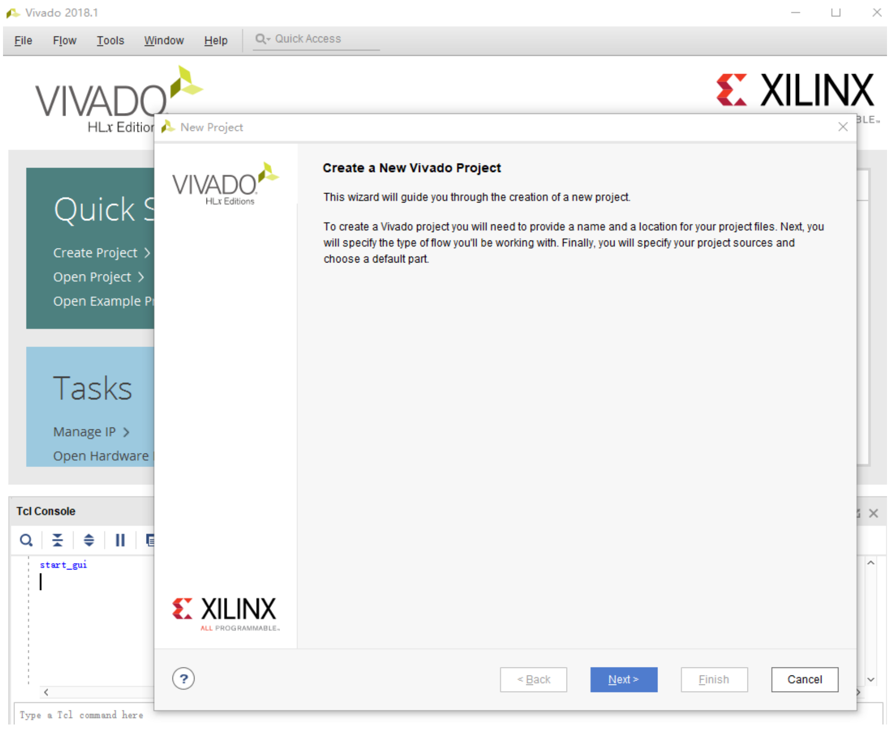


Fig 1. 3 Create a new project

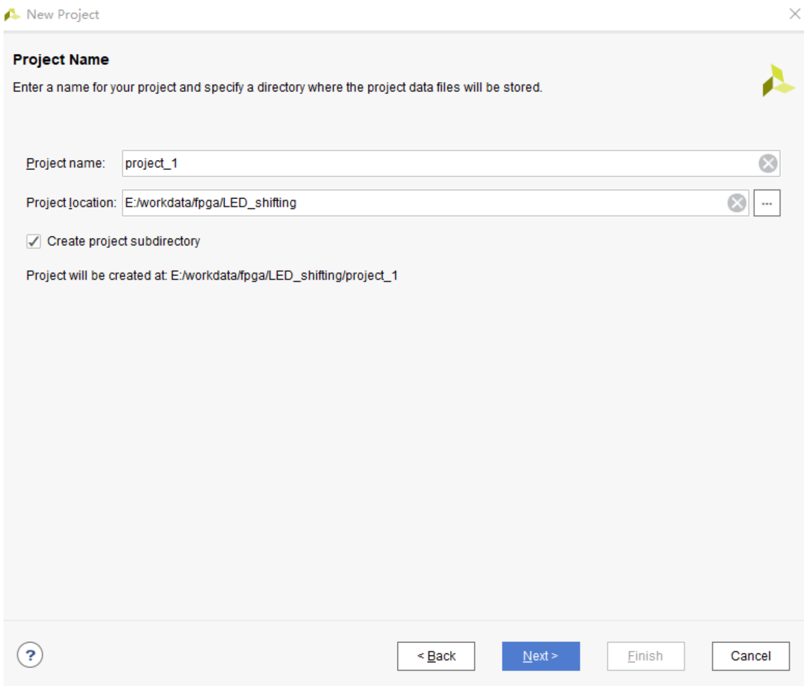


Fig 1. 4 Set the project path

Set the project name, project path. Note that the top-level file name must be consistent with the file name of the subsequent top-level file of Verilog. The top-level file name is case-sensitive.

b. Choose **RTL Project** to be the project type. See Fig 1. 5.

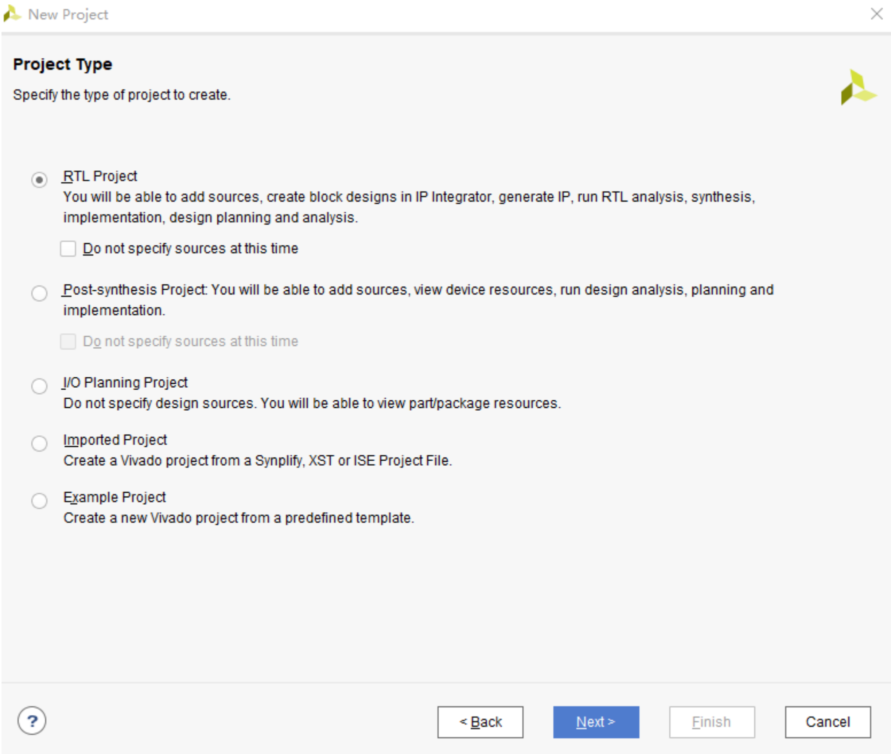


Fig 1. 5 Project type selecting

c. Click **Next** as shown in Fig 1. 6 (there is no source file that can be added since it is new)

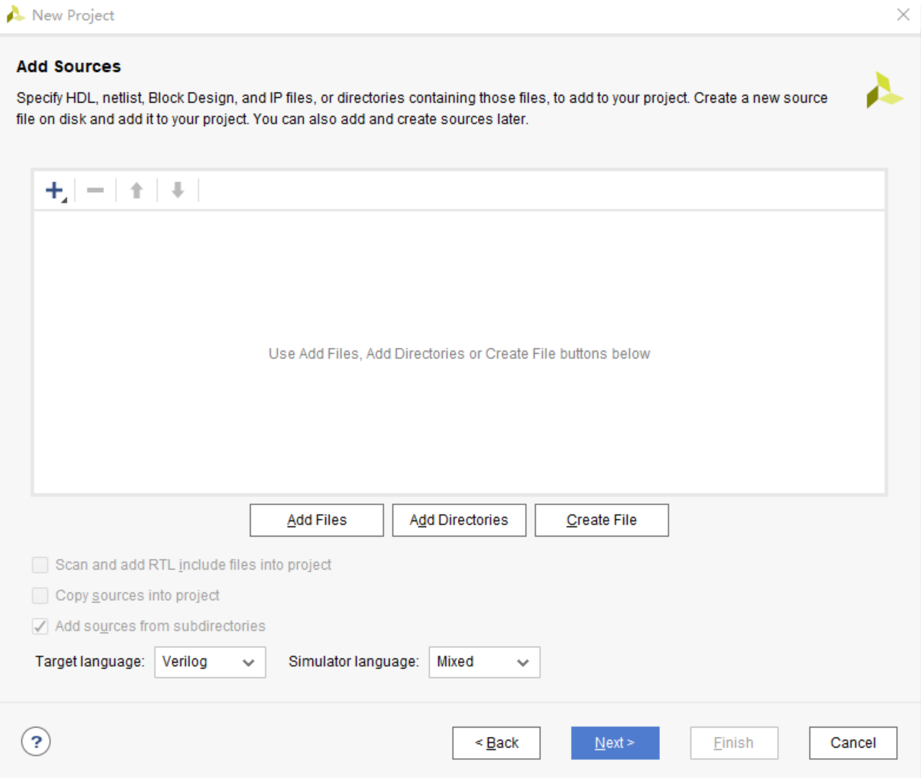


Fig 1. 6 Add source file

d. Click **Next** as shown in Fig 1. 7 (there are no files that can be added to constrain due to it is a new project)

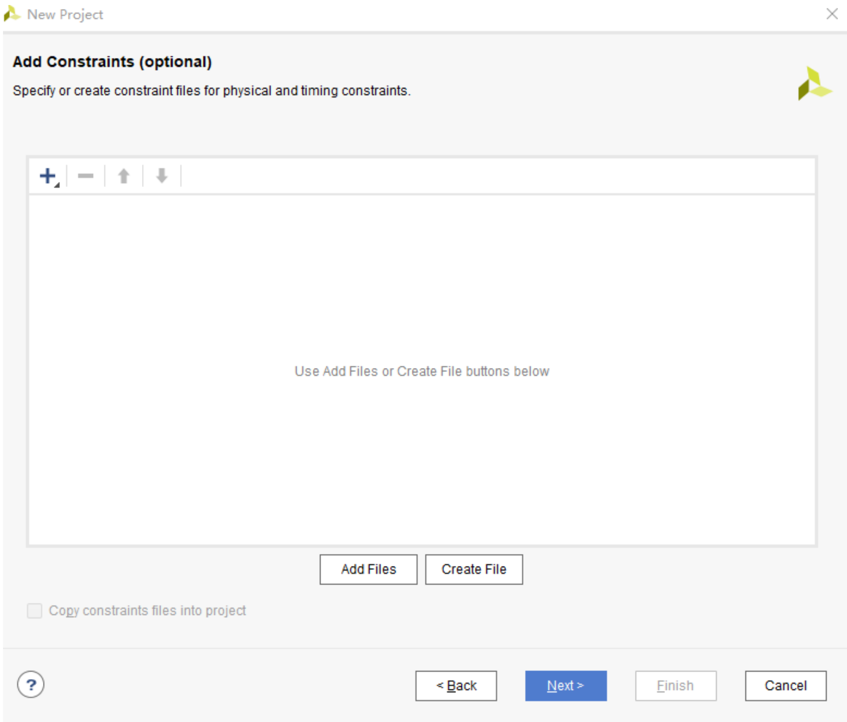


Fig 1. 7 Add constrains

e. Select **XC7A100TFGG676-2** in the selection dialog box. See Fig 1. 8, click **NEXT**, then **Finish** to complete the project building.

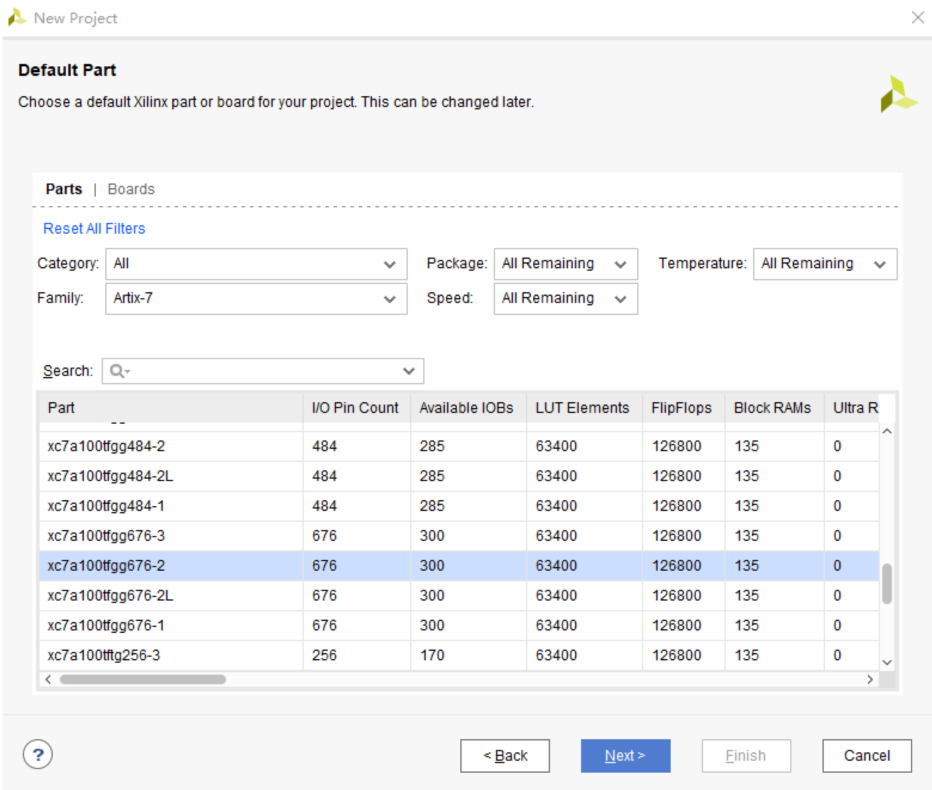


Fig 1. 8 Choose the default Xilinx part or board

(3) Create a Verilog HDL file, *LED_shifting.v*

a. Select **File -> Add Sources** or add the RTL file as shown in Fig 1. 9 or Fig 1. 10 below.

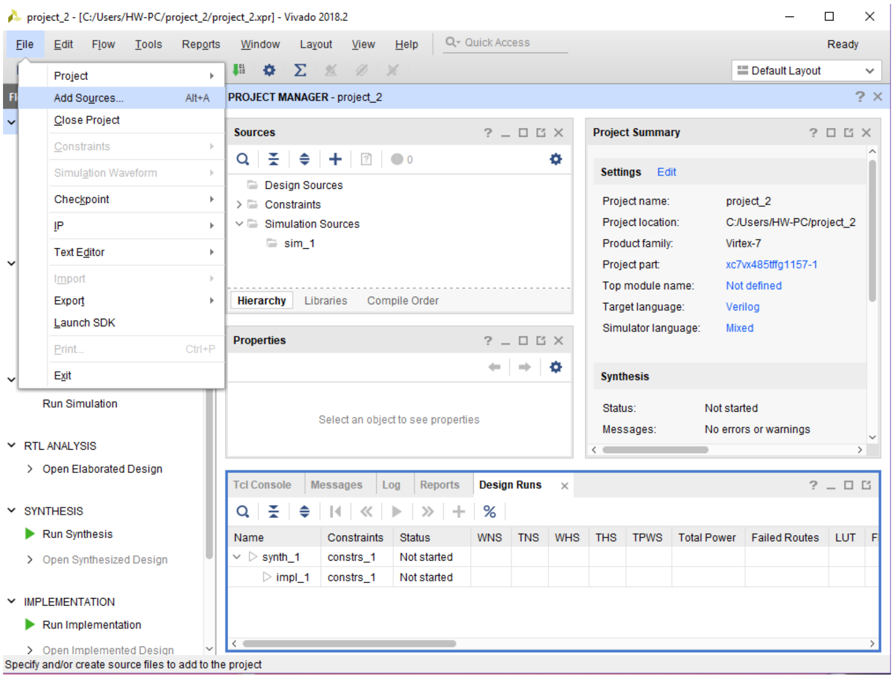


Fig 1. 9 Add source file

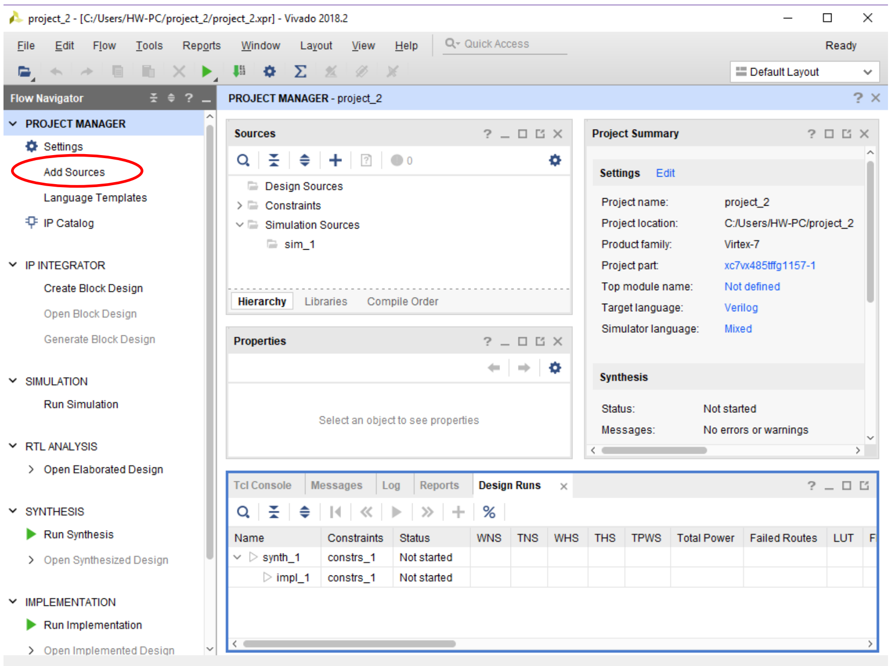


Fig 1. 10 Add source file

b. See Fig 1. 11, select **Add or create design sources** and then click **Next**.

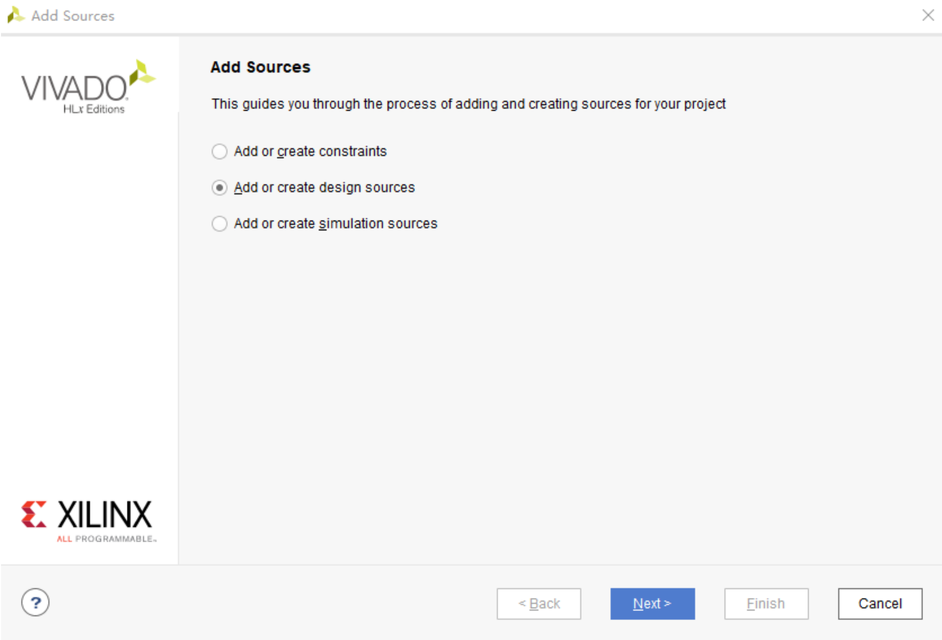


Fig 1. 11 Add source file 1

c. Click **Create File**. In the popup window, select the **Verilog** HDL for the file type. Fill in the file name and location **-> OK -> Finish**. See Fig 1. 12.

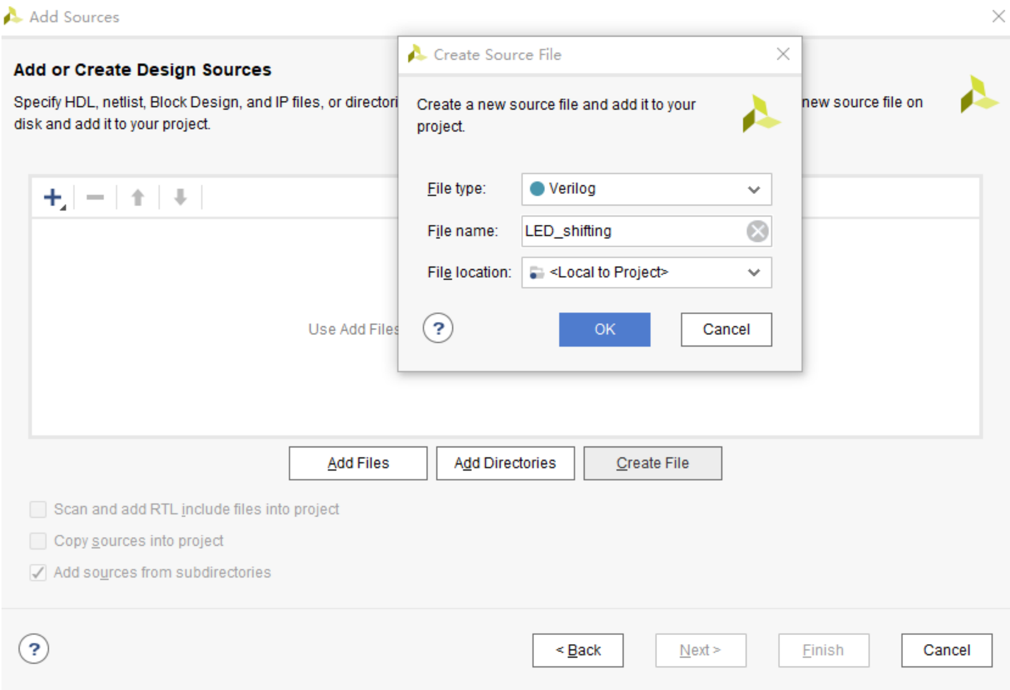


Fig 1. 12 Add source file 2

d. As shown in Fig 1. 13, if filling the module name wrongly in the previous step, the name can be modified here. Input and output pin configuration can also be directly set here through the I/O port definitions. (You can also write the generated pin information in the Verilog code later.) Then click **OK**.

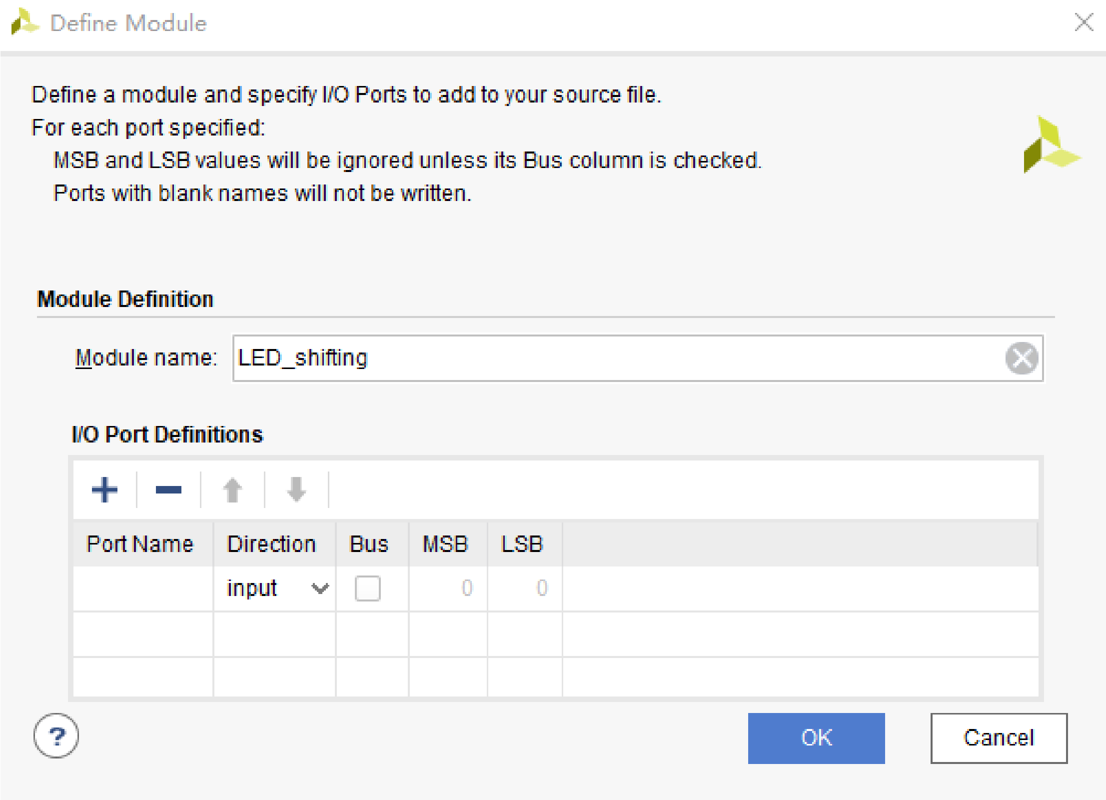


Fig 1. 13 Confirmation

e. Vivado's sources window generates an *LED_shifting* RTL file. Click on the file to edit the code. See Fig 1. 14.

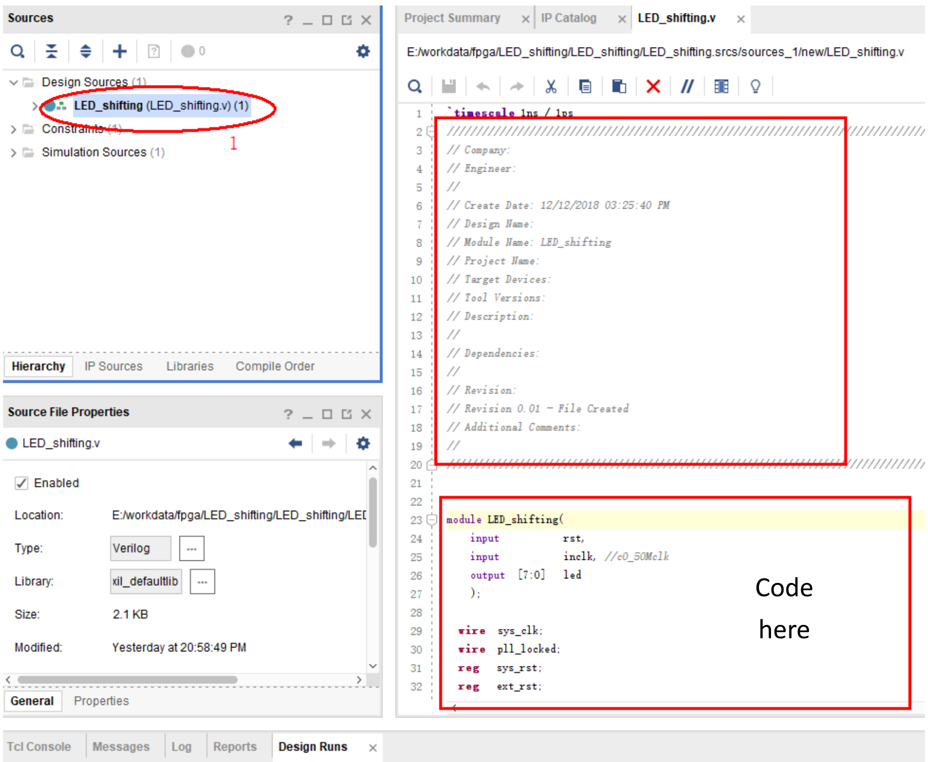


Fig 1. 14 Source file editing

f. Edit interface file

```
module Led_shifting(
    input           rst,
    input           inclk, //c0_50Mclk
    output  [7:0]   led
    );
endmodule
```

(4) Add clock module

See Fig 1. 15, click the **IP Catalog** option on the left side of the main interface to pop up the corresponding core supported by the engineering chip. Find the needed IP core by functions or names, or by fast searching. Entering clocking in step 1, then click **Clocking Wizard** shown in step 2. The clock IP configuration interface will appear after that.

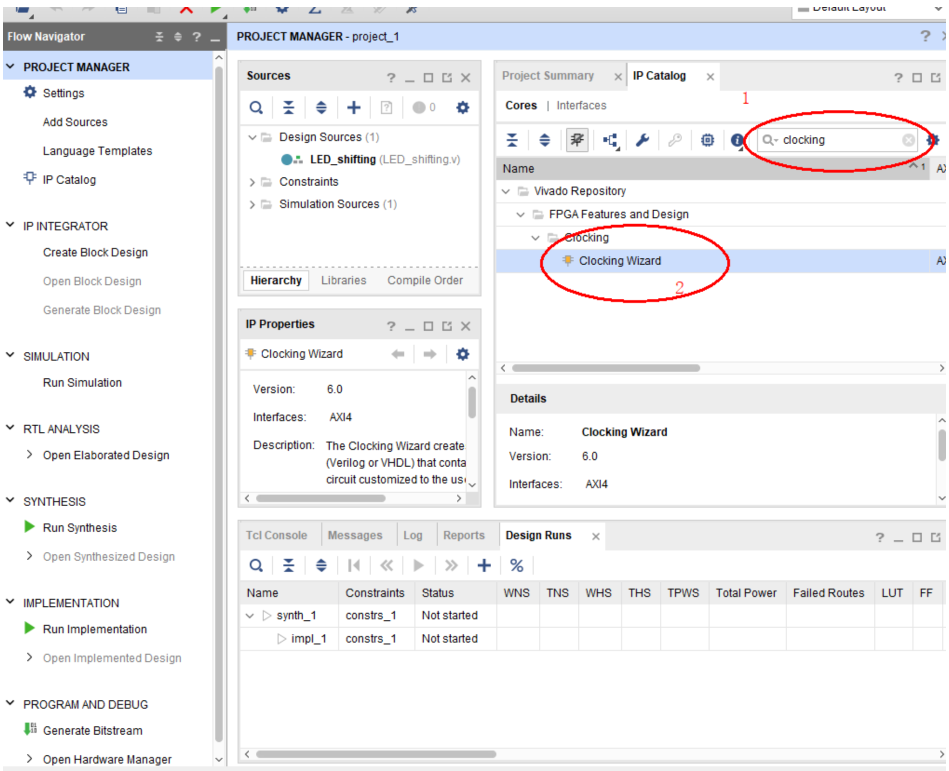


Fig 1. 15 PLL IP core setting

a. Enter the clock setting as shown below

1) Select either MMCM or PLL here. Here is an example of selecting a PLL core.
2) The path filled in Fig 1. 16 is the setting of the clock file path. Fig 1. 17 shows the name setting.
3) See Fig 1. 18, *clk_in1* (which is the input clock of the PLL, where there is only one input clock) is set to be 50 MHz, which is consistent with the clock provided by the hardware board.
4) Other PLL settings can be selected by default. If the required functions involve advanced features, use the official reference for more.
5) Click the **Output Clocks** tab to set the PLL compensation output clock to *clk_out1*.

6) For PLL asynchronous reset control and capture lock status settings, use the default mode shown in the figure.

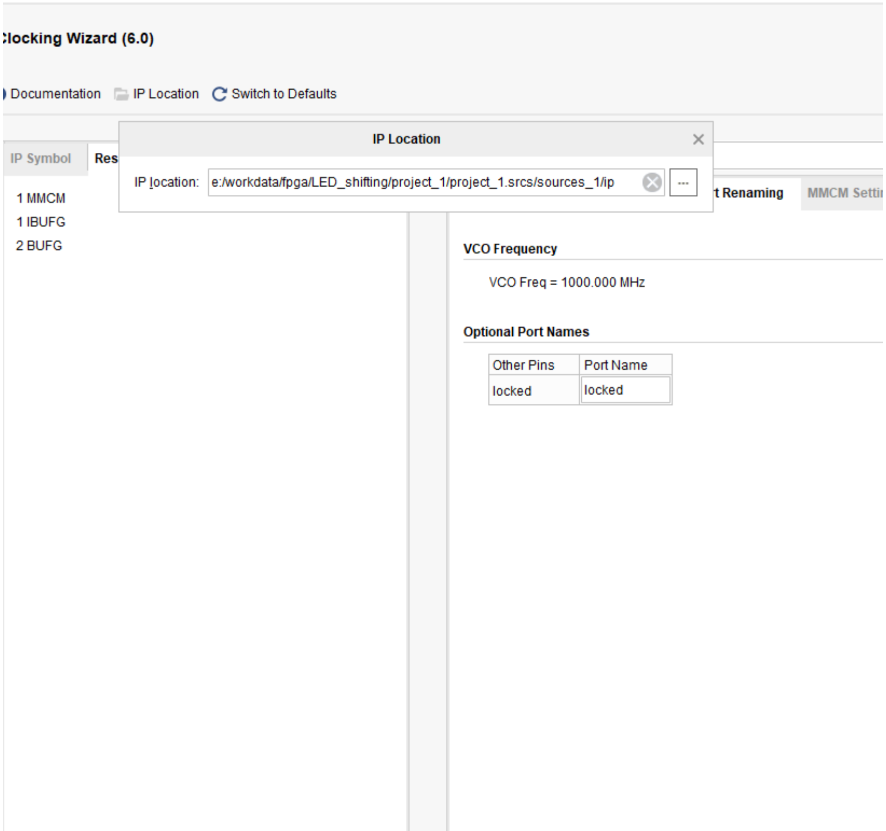


Fig 1. 16 IP location setting window

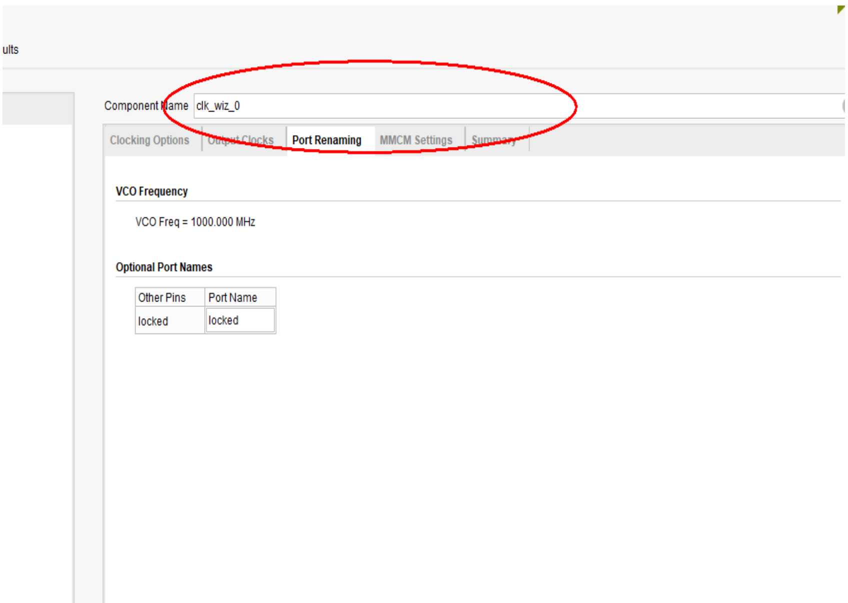


Fig 1. 17 IP core name setting

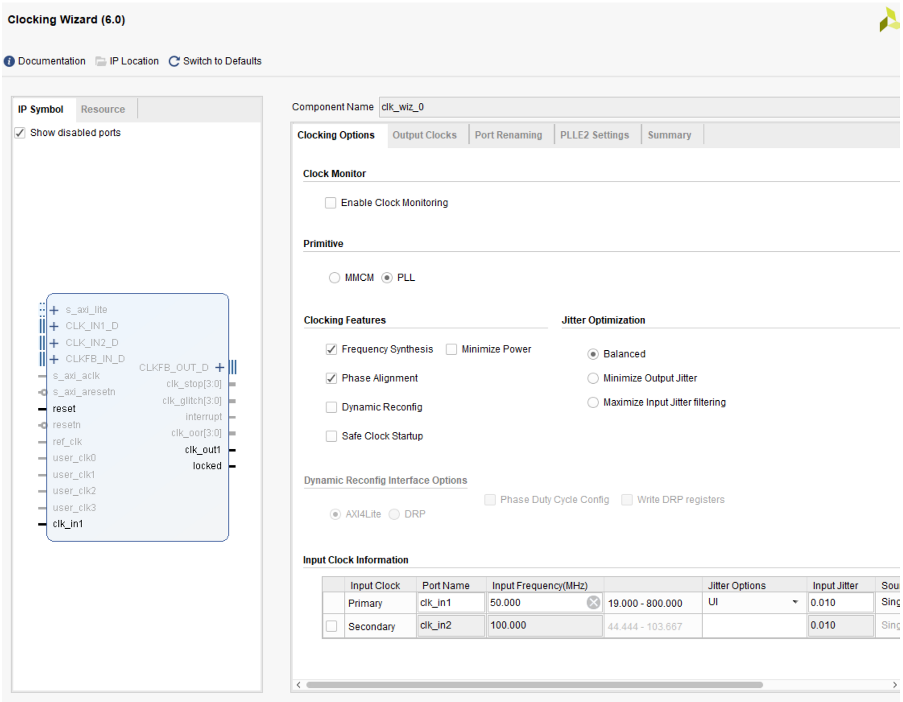


Fig 1. 18 PLL input clock setting

7) See Fig 1. 19, set the output frequency to 100 MHz, the phase offset to 0, and the duty cycle to 50%. Click **OK**.

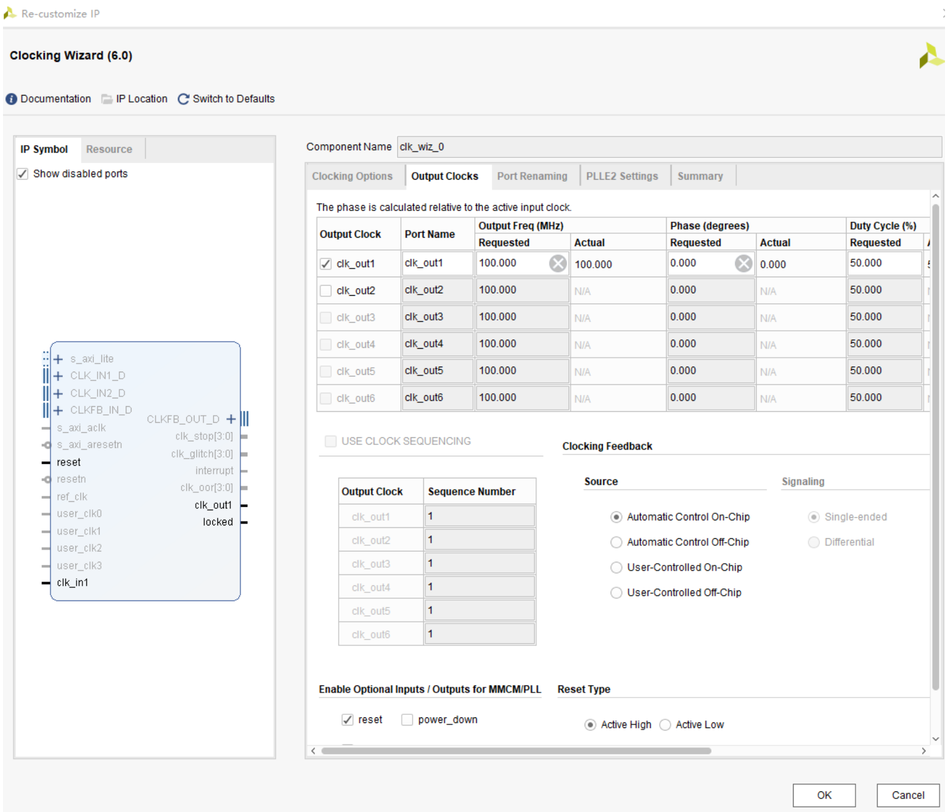


Fig 1. 19 Output frequency and duty cycle setting

8) Click **Generate** to finish the IP core setting. See Fig 1. 20.

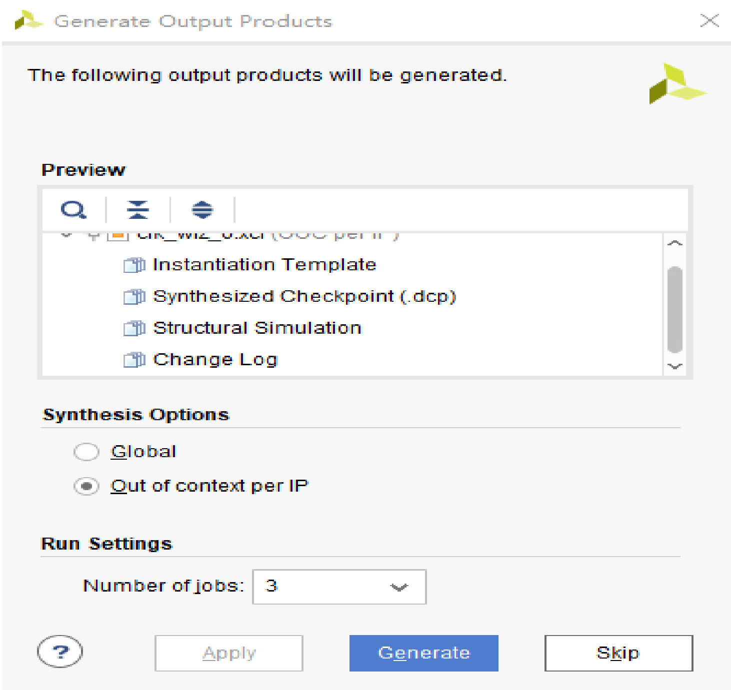


Fig 1. 20 Generate IP core

9) After the clock module is generated, select the **IP Sources** sub-tab in the Labels box of the **Sources** of the project interface, that is, the IP core file can be found just after the generation. See Fig 1. 21.

10) Instantiate the module to the top-level entity

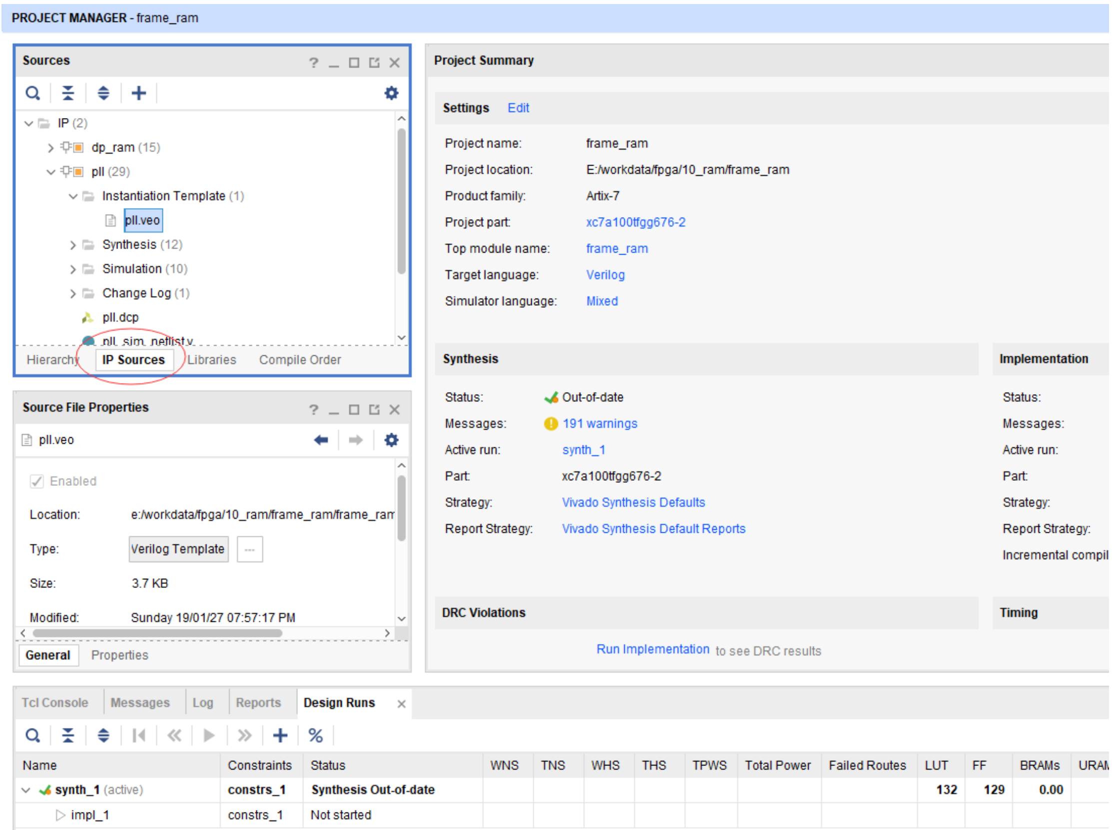


Fig 1. 21 Instantiate to the top-level entity

The code is as follows:

11) Top-level entity instance
12) Key signal description

sys_rst, the value before the PLL lock is '1' as a reset signal for the entire system. After the system is locked (*pll_locked == 1'b1*), the value of *sys_rst* becomes '0'. At the same time, it is driven by the rising edge of sys_clk, so it is a synchronous reset signal.

```
module Led_shifting(
        input               inclk, //c0_50Mclk
        output   [7:0]      led
        );

        wire   sys_clk;
        wire   pll_locked;
        reg    sys_rst;

        always@(posedge sys_clk) begin
          sys_rst<=!pll_locked;
        end

        clk_wiz_0    clk_wiz_0_inst(
                .clk_out1     (sys_clk),
                .reset        (1'b0),
                .locked       (pll_locked),
                .clk_in1      (inclk)
        );
endmodule
```

Note that the user is already familiar with the Verilog syntax by default, so the Verilog syntax is not exhaustive here.

(5) Frequency division design

a. The system clock is 100 MHz, while the speed of the LED blinking is set to be 1 second, so frequency division is needed.
b. Microsecond frequency division

The Verilog HDL code is as follows：

```
reg [7:0] us_reg;
reg          us_f;

always@(posedge sys_clk)
        if(sys_rst) begin
                us_reg<=0;
                us_f<=1'b0;
```

```
        end
        else begin
        us_f<=1'b0;
        if(us_reg==99)begin
            us_reg<=0;
            us_f<=1'b1; //Microsecond pulse, outputs a sys_clk pulse every //1 us
        end
        else begin
            us_reg<=us_reg+1;
        end
    end
```

c. Millisecond frequency division

```
        reg [9:0] ms_reg;
        reg       ms_f;

        always@(posedge sys_clk)
            if(sys_rst) begin
                ms_reg<=0;
                ms_f<=1'b0;
            end
            else begin
                ms_f<=1'b0;
            if(us_f) begin
                if(ms_reg==999)begin    //Every 1000 microseconds, ms_f //produces a
    sys_clk pulse
                    ms_reg<=0;
                    ms_f<=1'b1;
                end
            else//Counter adds 1 every microsecond
             ms_reg<=ms_reg+1;
            end
        end
```

d. Second frequency division

```
always@(posedge sys_clk)
    if(sys_rst) begin
      s_reg<=0;
      s_f<=1'b0;
    end
    else begin
        s_f<=1'b0;
```

```
        if(ms_f) begin
            if(s_reg==999)begin
            s_reg<=0;
            s_f<=1'b1;
            end
        else
            s_reg<=s_reg+1;
        end
    end
```

e. LED shifting design

```
    always@(posedge sys_clk)
        if(sys_rst) begin
          s_reg<=0;
          s_f<=1'b0;
        end
        else begin
            s_f<=1'b0;
            if(ms_f) begin
                if(s_reg==999)begin
                s_reg<=0;
                s_f<=1'b1;
                end
            else
                s_reg<=s_reg+1;
            end
      end
```

Because the schematics design uses FPGA I/O sink current mode, it must be inverted bitwise before output. Otherwise, it will show that each time 7 LEDs are lit, only one LED is left in the non-lighting state.

Assign led=~led_r; //Bitwise inverse

The pin assignment table of the program is as follows:

Signal Name	Port Description	Network Label	FPGA Pin
inclk	System clock 50 MHz	C10_50MCLK	U22
rst	Reset, high by default	KEY1	M4
led0	LED 0	LED0	N17
led1	LED 1	LED1	M19
led2	LED 2	LED2	P16
led3	LED 3	LED3	N16
led4	LED 4	LED4	N19
led5	LED 5	LED5	P19
led6	LED 6	LED6	N24
led7	LED 7	LED7	N23

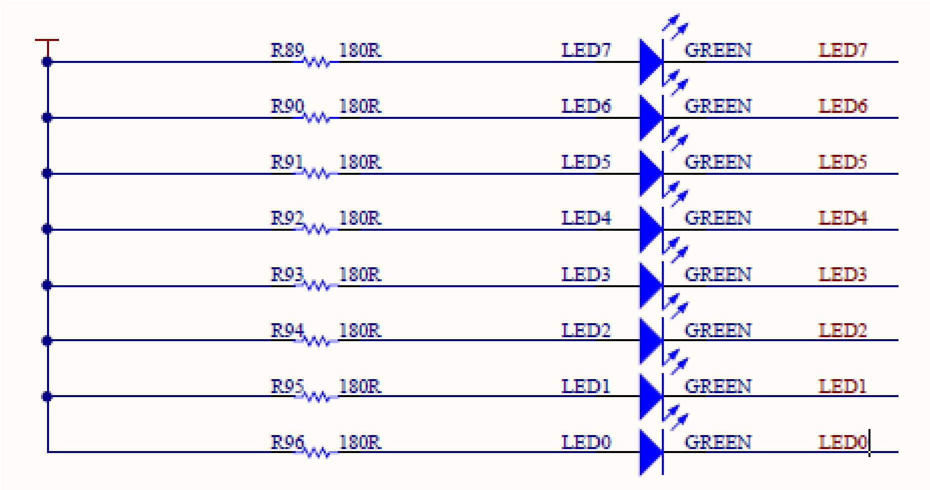


Fig 1. 22 Schematics for LED

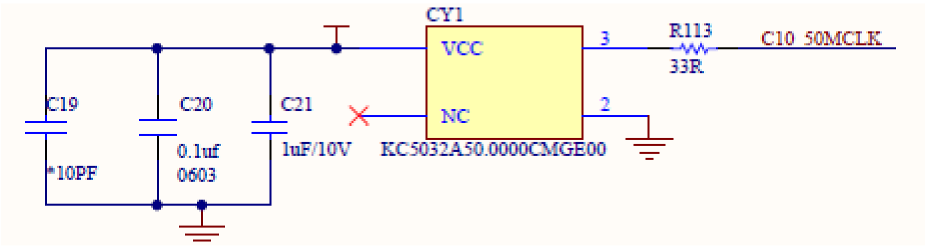


Fig 1. 23 FPGA input clock

(6) After the code is integrated, there are two ways to add the constraint file. One is to use the I/O planning function in Vivado, and the other is to directly create a constraint file for the XDC and manually enter the constraint command. Here the first method is adopted for now, I/O planning function. The procedure is as follows

a. Go to **Flow Navigator -> Synthesis -> Run Synthesis**, integrate the project first. See Fig 1. 24. The purpose is:

1) Check the syntax error
2) Form the tree hierarchy of the project

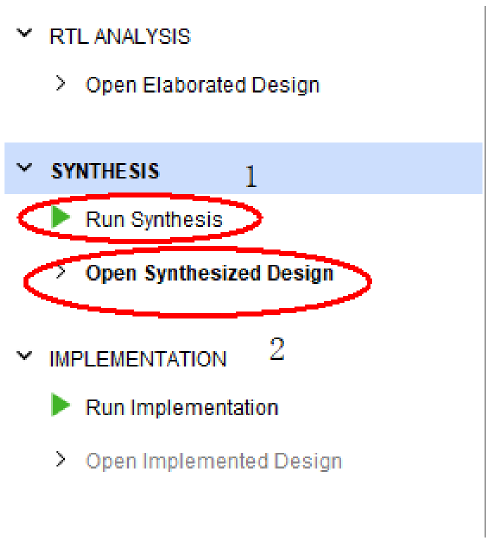


Fig 1. 24 Check the syntax, compilation synthesis

After the integration is complete, select **Open Synthesized Design**, open the comprehensive results, select I/O Planning under layout, and assign the pins in the I/O port section in the figure below.

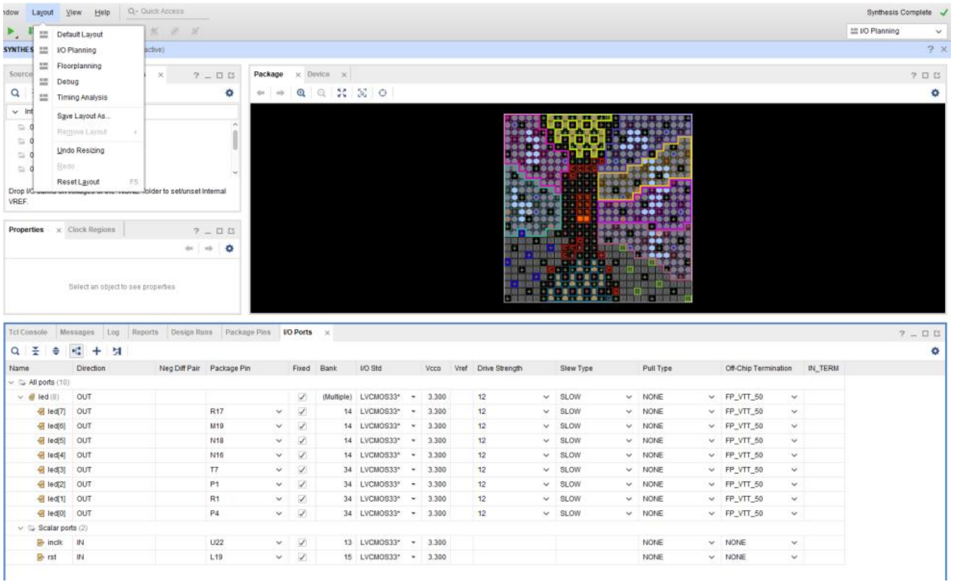
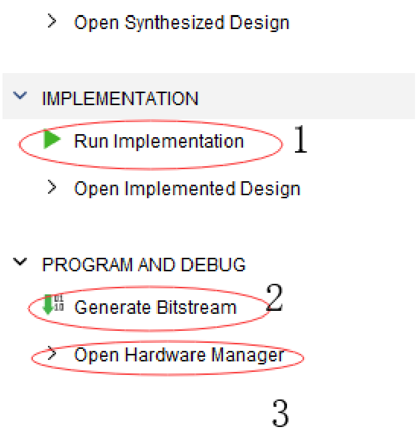


Fig 1. 25 Pin assignment

b. After the pin assignment is completed, click **Run Implementation** as shown in Fig 1. 26. After the completion of the **Generate Bitstream**, generate a downloadable bit file. Click **Open Hardware Manager** to link to the device. See Fig 1. 27.



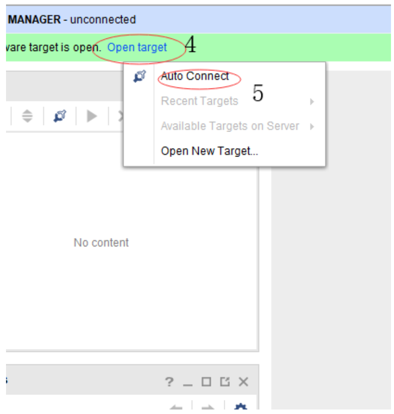


Fig 1. 26 Generate bit files Fig 1. 27 Connect with the experiment board

c. As shown in Fig 1. 28 below, select the correct bit file and download the bit file settings.

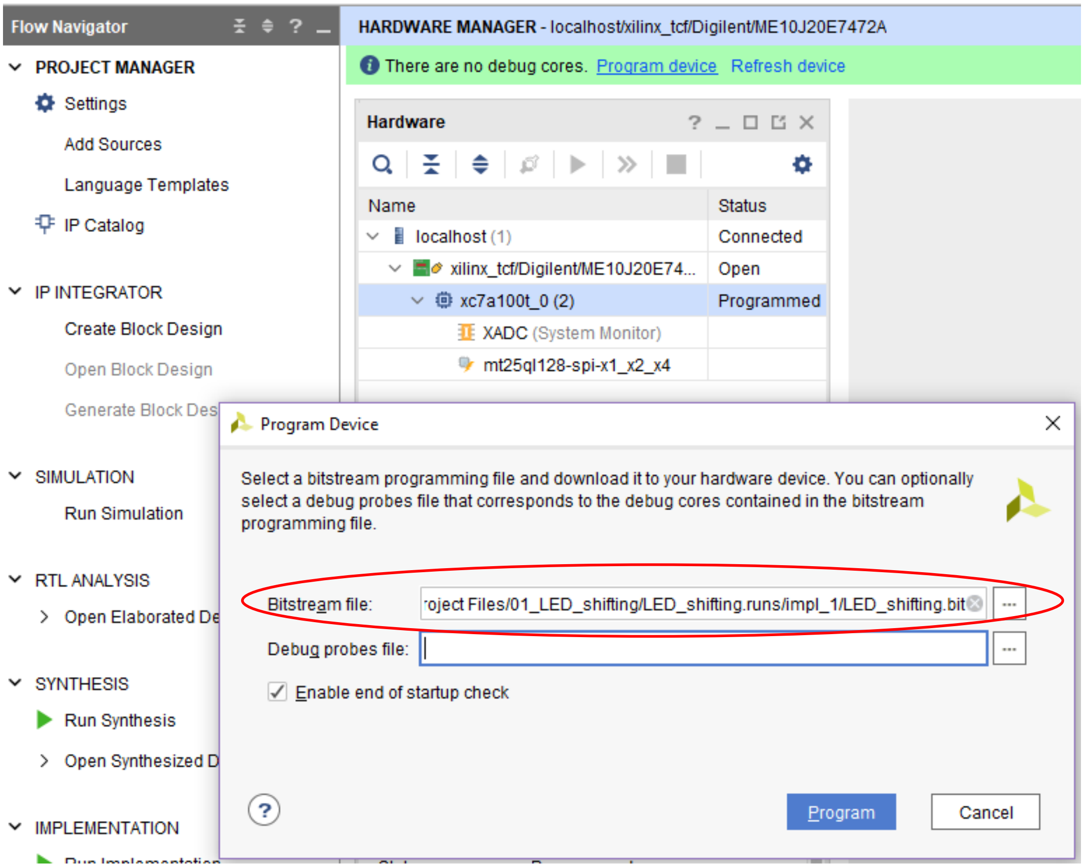


Fig 1. 28 Download the bit file configuration

d. Click **Program** to download the program to the board to test

1) The hardware connection is shown as follows, the 8 LEDs blink one by one.

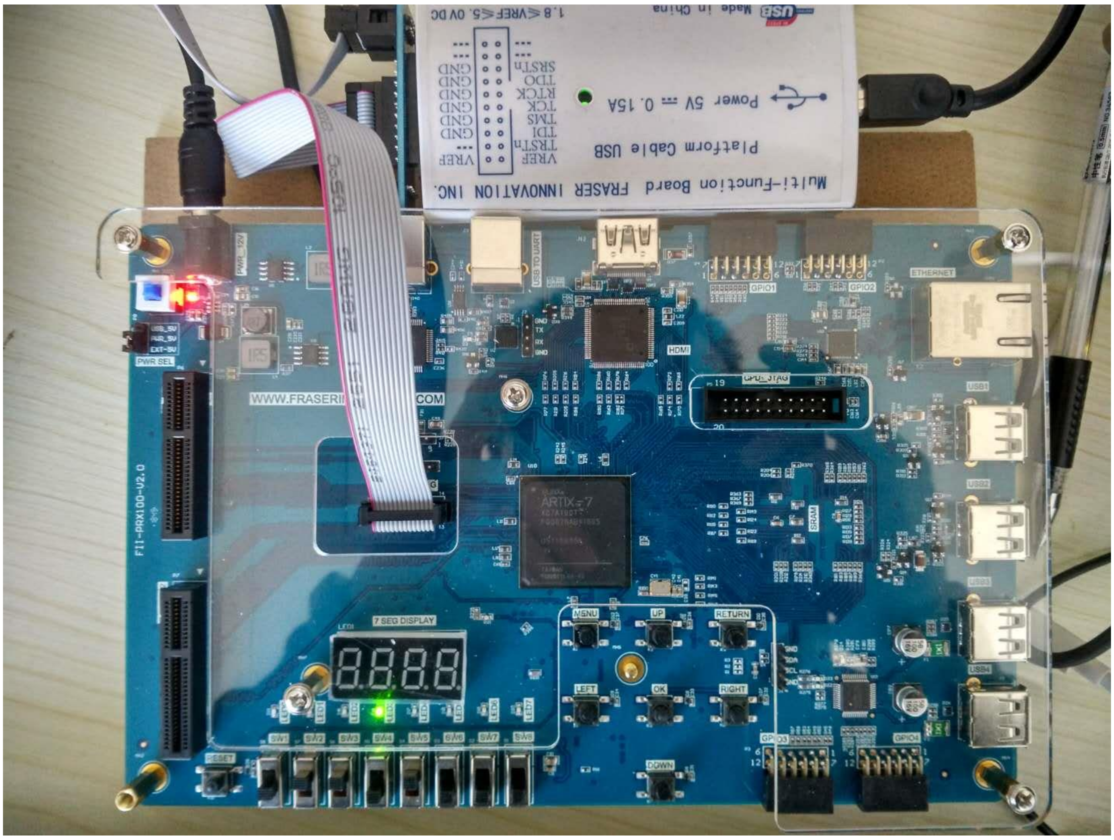


Fig 1. 29 Develop board

2) Review the above steps to be proficient in each process

Experiment 2 Switches and display

1.Experiment Objective

(1) Continue to practice using develop board
(2) Learn to use ILA (Integrated Logic Analyzer) in Vivado
(3) Learn to use the FPGA configuration memory for programming

2.Start New Project

(1) Refer to Experiment 1
(2) Select the same chip in Experiment 1
(3) Add PLL1 (Here PLL1 is optional, external input clock can be used directly)

3.Verilog HDL Code

```
module SW_LED(
	input			inclk,
	input		[7:0]	sw,
	output reg[7:0]		led
	);
	wire sys_clk;
	wire pll_locked;
	reg sys_rst;
	always@(posedge sys_clk)
	sys_rst<=!pll_locked;

	always @(posedge inclk)
		if(sys_rst)
			led<=8'hff;
		else
			led<=~sw;
	PLL1 PLL1_INST(
		.reset	(1'b0),
		.clk_in1	(inclk),
		.clk_out1 (sys_clk),
		.locked	(pll_locked)
		);
endmodule
```

Schematics of develop board

(1) See Fig 2. 1. the diodes D19-D26 are mainly used to eliminate the damage of the FPGA pin caused by human body contact static electricity.

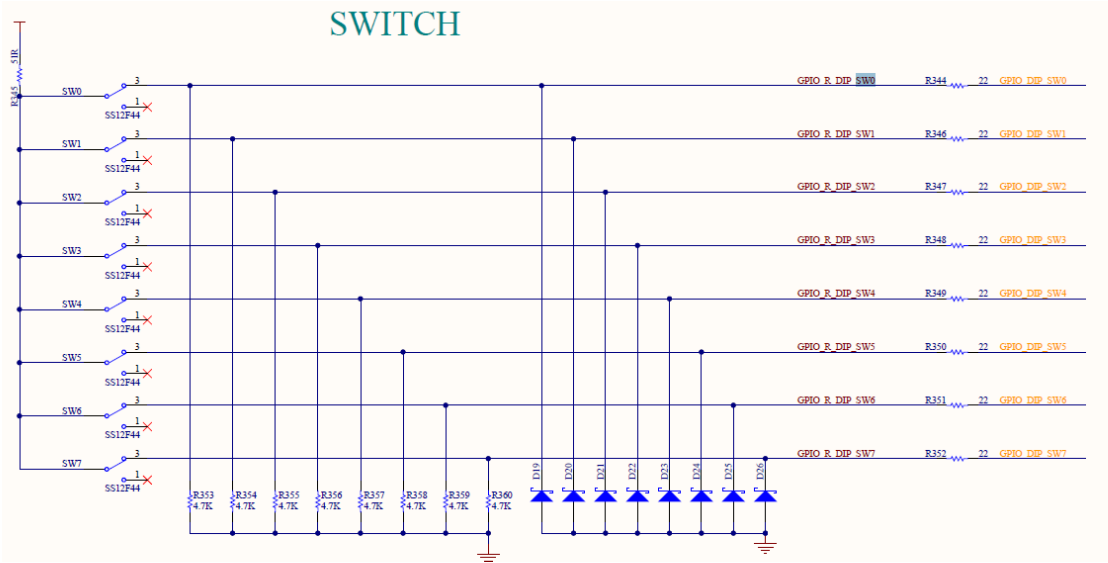


Fig 2. 1 Switches drive the circuit

4.FPGA Pin Assignment

Signal Name	Port Description	Network Label	FPGA Pin
inclk	System Clock 50 MHz	C10_50MCLK	U22
rst	Reset, high by default	KEY1	M4
led0	LED 0	LED0	N17
led1	LED 1	LED1	M19
led2	LED 2	LED2	P16
led3	LED 3	LED3	N16
led4	LED 4	LED4	N19
led5	LED 5	LED5	P19
led6	LED 6	LED6	N24
led7	LED 7	LED7	N23
SW0	SW 0	GPIO_DIP_SW0	N8
SW1	SW 1	GPIO_DIP_SW1	M5
SW2	SW 2	GPIO_DIP_SW2	P4
SW3	SW 3	GPIO_DIP_SW3	N4
SW4	SW 4	GPIO_DIP_SW4	U6
SW5	SW 5	GPIO_DIP_SW5	U5
SW6	SW 6	GPIO_DIP_SW6	R8
SW7	SW 7	GPIO_DIP_SW7	P8

5.Program in Vivado

6.Download to the develop board to test and dial the DIP switch to see the corresponding LED light on and off. See Fig 2. 2.

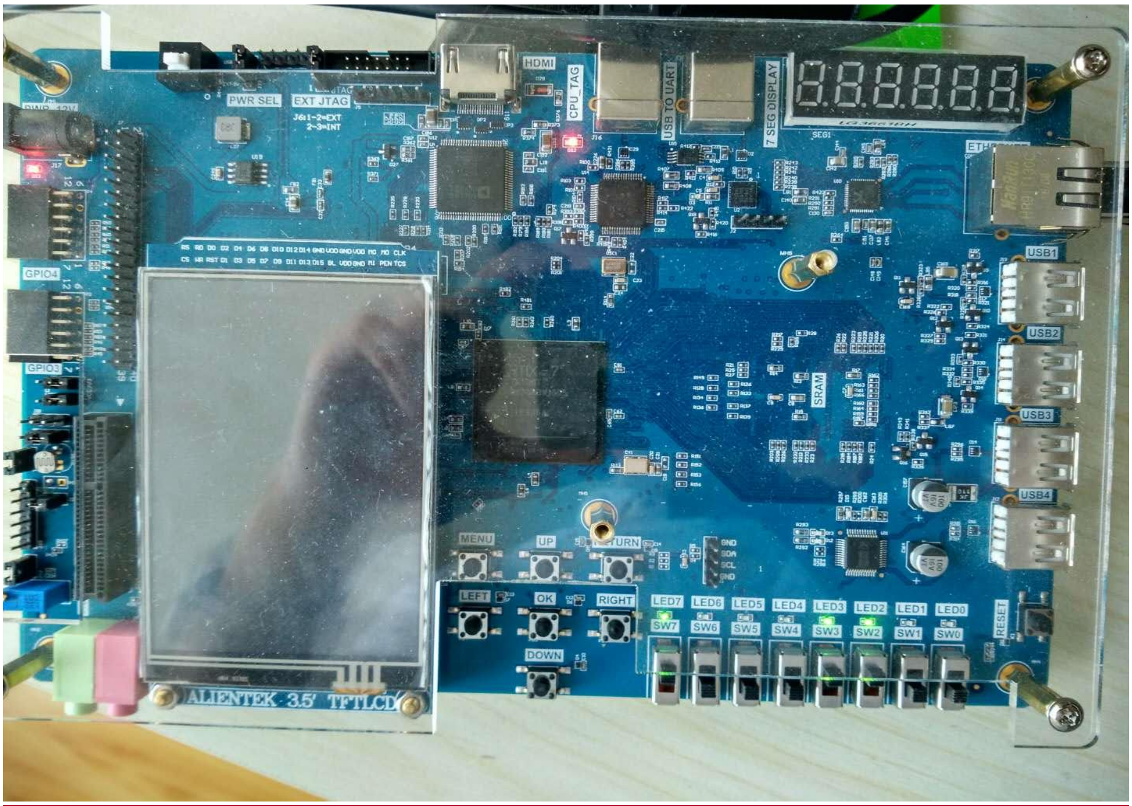


Fig 2. 2 Experiment result

7.Use of ILA

(1) Choose top-level entity *SW_LED.v* file to **Run Synthesis**

a. After the integration is complete, under the **Netlist** window, all network nodes present in the current design are listed. Debug the network nodes. See Fig 2. 3.

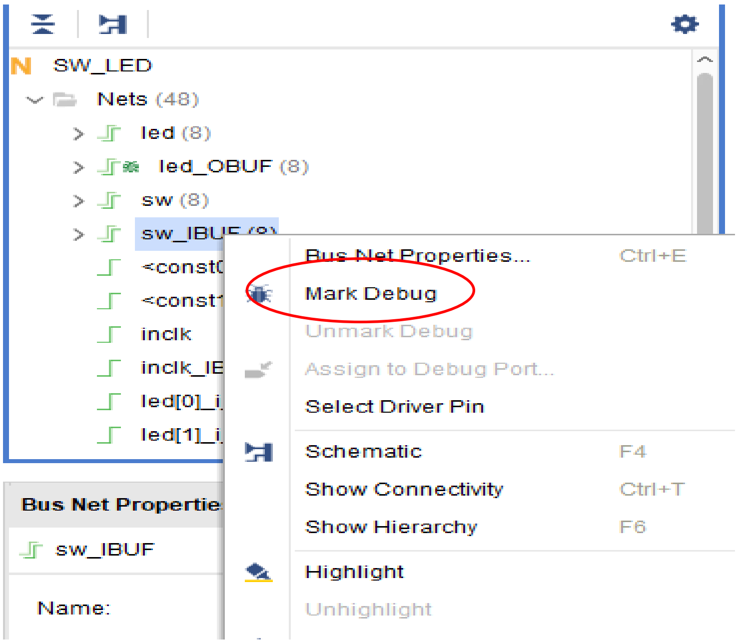


Fig 2. 3 Mark debugged network nodes

b. In the Vivado main interface menu, execute the menu command **Tool -> Set up** Debug. In the popup window, there is clock domain of the selected debug signal. The clock domain of *sw_IBUF* is red. See Fig 2. 4.

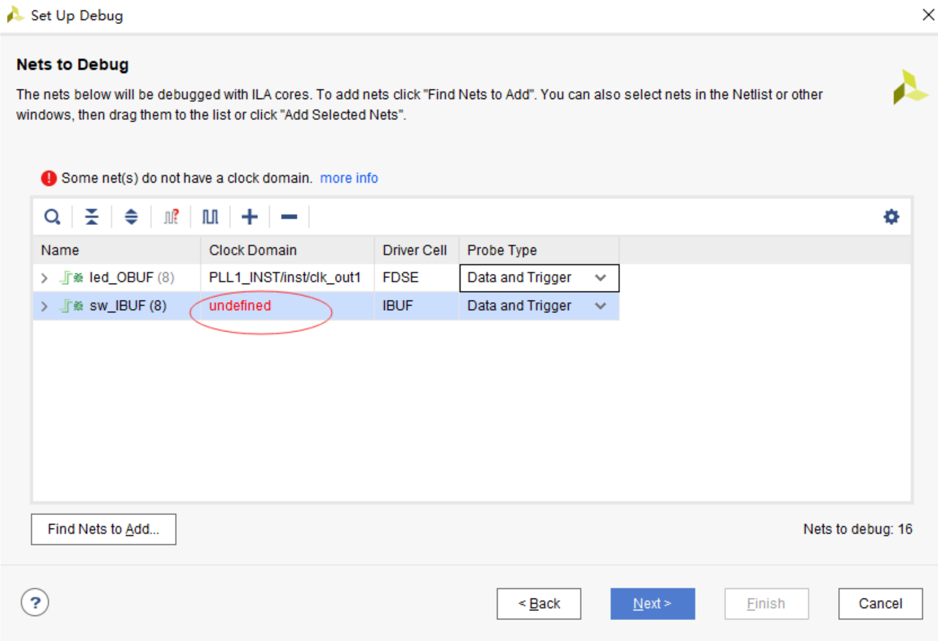


Fig 2. 4 Debugged network node clock domain setting

c. In the red circle shown in Fig 2. 4, right click to set the clock domain.

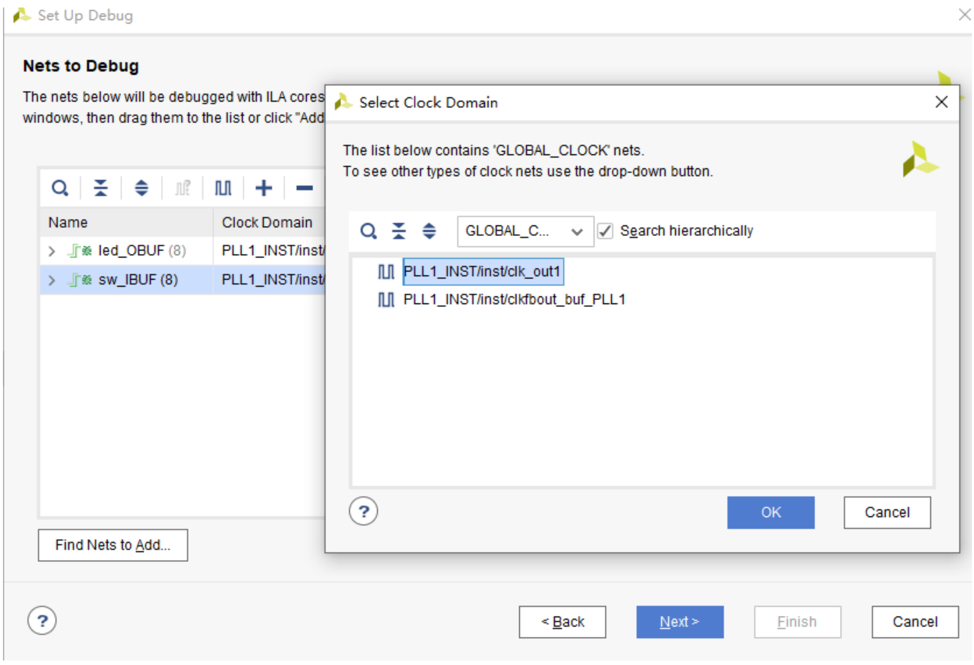


Fig 2. 5 Modify the debugged network node clock domain

d. After the setting is completed, click **Next**. The popup window is shown in Fig 2. 6. Set the data collection depth and select the check box in front of **Capture control** and **Advance trigger**. Then keep clicking **Next** until the end.

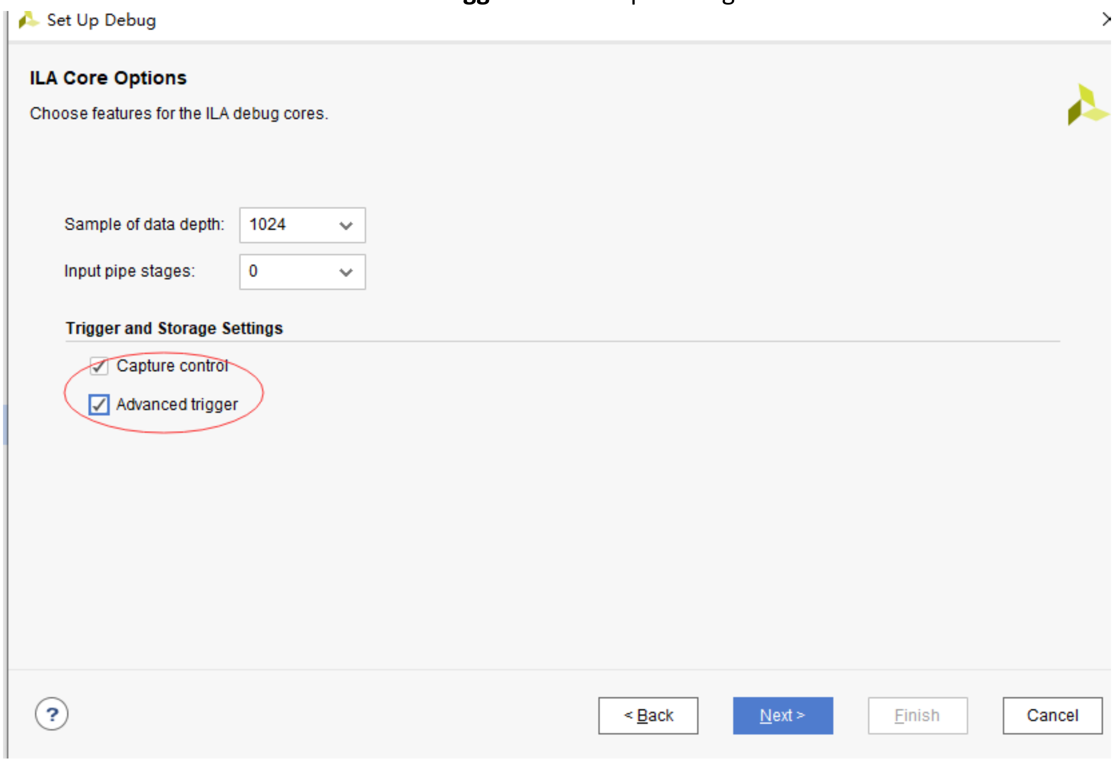


Fig 2. 6 Set the data collection depth

e. Add I/O pin constraint information for implementation. Then generate a bit file and download it to FPGA. The debugging interface is automatically popped up. Click the icon button to see the following results. The test results in the debug diagram below Fig 2. 7 indicate that the design results are correct.

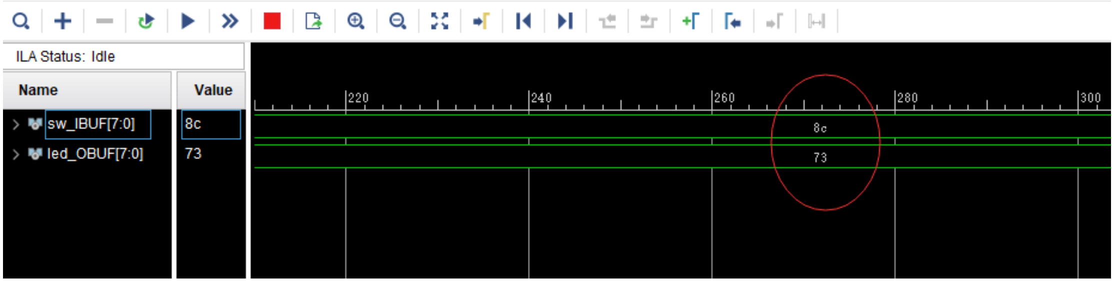


Fig 2. 7 Debug

When the input of switch is high, the input LED pin is controlled to be low, and the LED is lit. The figure for the experiment result on board from above shows that the input *sw* is 10001100 and the LED light is 01110011. The hexadecimal is 8c and 73 respectively. It is consistent with the ILA test results in the figure above.

(2) Modify the trigger condition to test the output under different trigger conditions

Experiment 3 Basic Digital Clock Experiment and Programming of FPGA Configuration Files

1.Experiment Objective

(1) Review the contents of experiment 1 and experiment 2, master the configuration of PLL, the design of frequency divider, the principle of schematics and the pin assignment of FPGA.
(2) Study BCD decoder
(3) Display design of 4-digit hexadecimal to 7 segment display decoders
(4) Generate a programmable configuration file and program it to the serial FLASH of the development board through the JTAG interface.

2.Design of The Experiment

(1) Refer experiment 1 for building new projects, chip selection

```
module BCD_counter(

        input                   rst,
        input                   inclk, //c0_50Mclk
        output reg   [7:0]    seven_seg,
        output reg   [3:0]    scan
          );
          wire    sys_clk;
        wire    pll_locked;
        reg      sys_rst;
        reg      ext_rst;

        always@(posedge sys_clk) begin
        sys_rst<=!pll_locked;
        ext_rst<=rst;
        end
```

(2) Add PLL, the input clock is 50 MHz, and the output clock is 100 MHz. Refer experiment 1 for more information

```
BCD_counterPLL1    BCD_counterPLL1_inst
(
.areset(1'b0),
.inclk0(inclk),
.c0(sys_clk),
```

```
                    .locked(pll_locked)
                    );
```

(3) Add microsecond, millisecond, and second frequency dividers. Refer to experiment 1.

```
                   reg [7:0] us_reg;
                    reg [9:0] ms_reg;
                    reg [9:0] s_reg;
                    reg         us_f,ms_f,s_f,min_f;

                    always@(posedge sys_clk)    //Microsecond frequency division
                    if(sys_rst) begin
                       us_reg<=0;
                       us_f<=1'b0;
                    end
                    else begin
                       us_f<=1'b0;
                       if(us_reg==99)begin
                       us_reg<=0;
                       us_f<=1'b1;
                    end
                    else begin
                    us_reg<=us_reg+1'b1;
                    end
                    end

                      always@(posedge sys_clk)
                    if(sys_rst) begin
                       ms_reg<=0;
                       ms_f<=1'b0;
                    end
                    else begin
                       ms_f<=1'b0;
                       if(us_f)begin
                          if(ms_reg==999)begin
                             ms_reg<=0;
                             ms_f<=1'b1;
                         end
                         else
                          ms_reg<=ms_reg+1'b1;
                      end
              End
```

```
always@(posedge sys_clk)
if(sys_rst) begin
s_reg<=0;
s_f<=1'b0;
end
else begin
  s_f<=1'b0;
  if(ms_f)begin
    if(s_reg==999)begin
      s_reg<=0;
      s_f<=1'b1;
        end
    else
      s_reg<=s_reg+1'b1;

 end
end
```

(4) Minute and second frequency divider

```
always@(posedge sys_clk)      //Second frequency division
        if(!ext_rst)begin
          counta<=0;
          countb<=0;
        min_f <=1'b0;
        end
else begin
        min_f <=1'b0;
        if(s_f) begin
    if(counta==4'd9) begin
          counta<=4'd0;
              if(countb==5)begin
                countb<=0;
                min_f<=1'b1;
              end
            else
              countb<=countb+1'b1;
    end
    else begin
          counta<=counta+1'b1;
    end
  end
end

always@(posedge sys_clk)    //Minute frequency division
if(!ext_rst)begin
```

```
            countc<=4'd0;
            countd<=4'd0;
          end
          else begin
               if(min_f) begin
            if(countc==4'd9) begin
               countc<=4'd0;
                    if(countd==5)begin
                      countd<=0;
                     end
              else
                     countd<=countd+1'b1;
            end
           else begin
            countc<=countc+1'b1;
           end
         end
          end
```

(5) Learn the schematics of the common anode segement decoder and the connection between the scanning circuit and the FPGA.

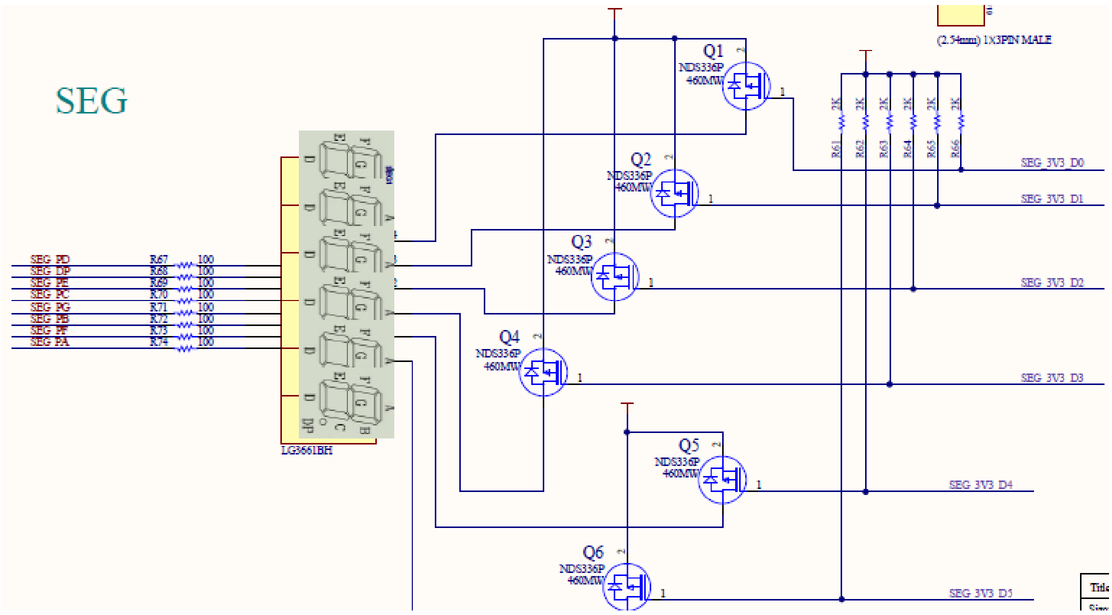


Fig 3. 1 Common anode segment decoder schematics

a. The pins of segment display decoder are shown in Fig 3. 1. This is a schematic diagram of the six decoders combined. The pin names A, B, C, D, E, F, and G (corresponding connections are SEG_PA, SEG_PB, SEG_PC, SEG_PD, SEG_PE, SEG_PF, SEG_PG) correspond to the 7 segments of the decoder, and the DP (corresponding connection is SEG_PD)corresponds to the 8th segment, which is commonly used as a decimal point display.

A, B, C, D, E, F, G, D, P select which segment of the decoder will lit. The segment to be lit corresponds to the low point.
Illumination of segment decoders is controlled by the bit selection lines SEG_3V3_D0, SEG_3V3_D1, SEG_3V3_D2, SEG_3V3_D3, SEG_3V3_D4, SEG_3V3_D5.

b. Code for the segment display decoder

```
always@(*)
case(count_sel)
0:seven_seg_r<=7'b100_0000;
1:seven_seg_r<=7'b111_1001;
2:seven_seg_r<=7'b010_0100;
3:seven_seg_r<=7'b011_0000;
4:seven_seg_r<=7'b001_1001;
5:seven_seg_r<=7'b001_0010;
6:seven_seg_r<=7'b000_0011;
7:seven_seg_r<=7'b111_1000;
8:seven_seg_r<=7'b000_0000;
9:seven_seg_r<=7'b001_0000;
default:seven_seg_r<=7'b100_0000;
endcase

  always@(posedge sys_clk)
    seven_seg<={1'b1,seven_seg_r};
```

c. Dynamic canning

The dynamic scanning of the segment display decoder utilizes the visual persistence characteristic of the human eye, and in addition to the speed of change that the human eye can distinguish, the segment corresponding to each decoder is quickly and time-divisionally illuminated. Because the time taken to illuminate all the decoders is less than the visual persistence of the human eye, in the eyes of the people, these decoders are continuously lit at the same time, and there is no feeling of flickering.

```
reg [1:0]    scan_st;
  always@(posedge sys_clk)
      if(!ext_rst) begin
        scan                  <=4'b1111;
  count_sel                   <=4'd0;
        scan_st<=0;
      end
    else case(scan_st)
0:begin
        scan                  <=4'b1110;
count_sel<=counta;
if(ms_f)
```

```
        scan_st<=1;
end
1:begin
        scan            <=4'b1101;
count_sel               <=countb;
  if(ms_f)
        scan_st         <=2;
end
2:begin
        scan<=4'b1011;
count_sel               <=countc;
  if(ms_f)
        scan_st<=3;
end
3:begin
        scan<=4'b0111;
count_sel<=countd;
if(ms_f)
         scan_st<=0;
end
default:scan_st<=0;
endcase
```

3.FPGA Pin Assignment

Signal Name	Port Description	Network Label	FPGA Pin
inclk	System clock 50 MHz	C10_50MCLK	U22
rst	Reset, high by default	KEY1	M4
seven_seg[0]	Segment a	SEG_PA	K26
seven_seg[1]	Segment b	SEG_PB	M20
seven_seg[2]	Segment c	SEG_PC	L20
seven_seg[3]	Segment d	SEG_PD	N21
seven_seg[4]	Segment e	SEG_PE	N22
seven_seg[5]	Segment f	SEG_PF	P21
seven_seg[6]	Segment g	SEG_PG	P23
seven_seg[7]	Segment h	SEG_DP	P24
scan[0]	Segment 1	SEG_3V3_D0	R16
scan[1]	Segment 2	SEG_3V3_D1	R17
scan[2]	Segment 3	SEG_3V3_D2	N18
scan[3]	Segment 4	SEG_3V3_D3	K25

(1) Lock the pin, compile, and download the program to the develop board

(2) Observe the test result

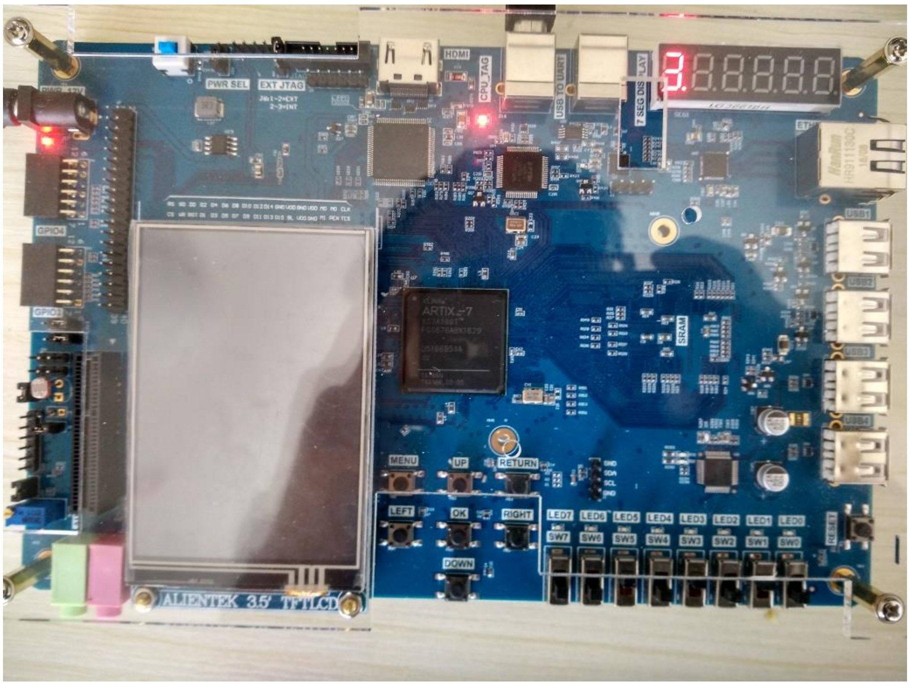


Fig 3. 2 Segment decoder illuminates

4.Configure the Serial Flash Programming

(1) The schematics of configuring serial Flash is as follows:

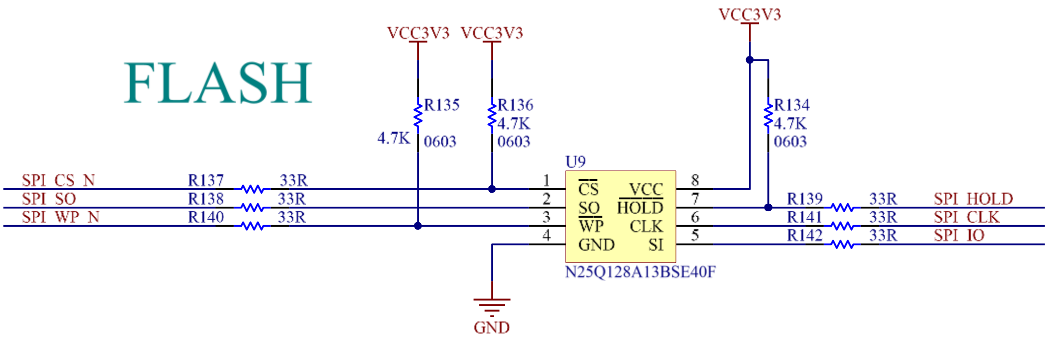


Figure 3. 3 Schematics of Serial Flash interface

(2) Configure FLASH and FPGA pin mapping

FLASH	*SPI_CS_N	SPI_SO	*SPI_WP_N	SPI_IO	SPI_SCLK	*SPI_HOLD
FPGA PINS	P18	R15	P14	R14	M22	N14

* SPI_CS_N, SPI_WP_N, SPI_HOLD must be connected to pull-up resistors

(3) FPGA configuration mode

Configuration Scheme	Valid MSEL[3..0]	POR Delay	Configuration Voltage Standard (V)
AS	1101	Fast	3.3
	0100	Fast	3.0
	0010	Standard	3.3
	0011	Standard	3.0
PS	1100	Fast	3.3/3.0/2.5
	0000	Standard	3.3/3.0/2.5
FPP	1110	Fast	3.3/3.0/2.5
	1111	Fast	1.8/1.5

(4) Configure the circuit, the resistor with the * mark in it is not soldered when the device is assembled, so the configuration circuit is selected as MSEL=0010, as shown in Table above.

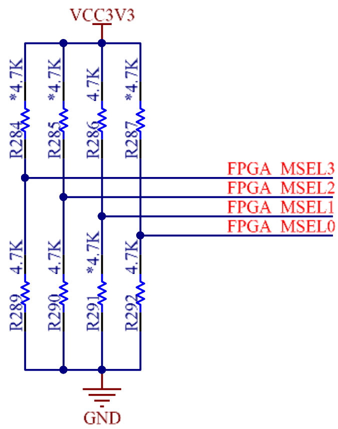


Fig 3. 4 Configuration option

(5) Generate a readable configuration file

a. See Fig 3. 5, right click on **PROGRAM AND DEBUG** to pop up the bitstream setting option.

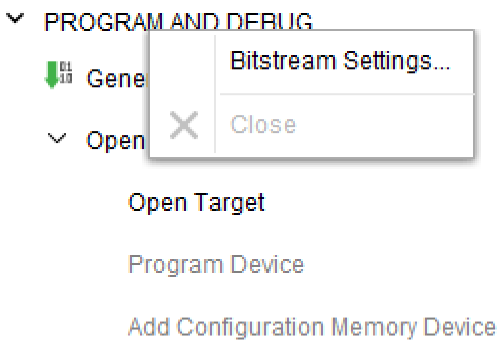


Fig 3. 5 Bit file generation setting

b. Click **Bitstream** setting, tick **bin_file***, click **OK**. See Fig 3. 6.

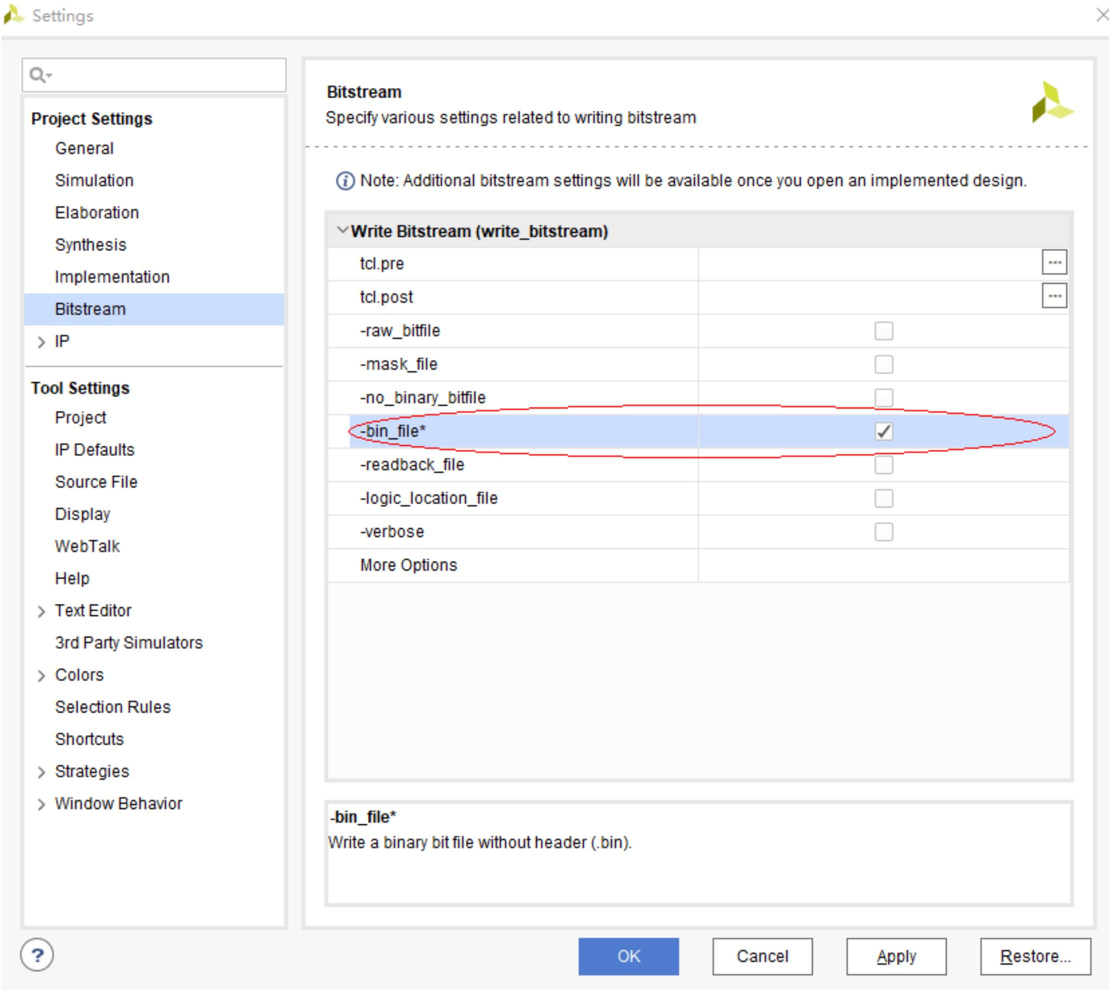


Fig 3. 6 Bin file generation setting

c. See Fig 3. 7, click **Generate Bitstream** to generate the bit file and bin file. Click **Open Hardware Manager** to connect the board

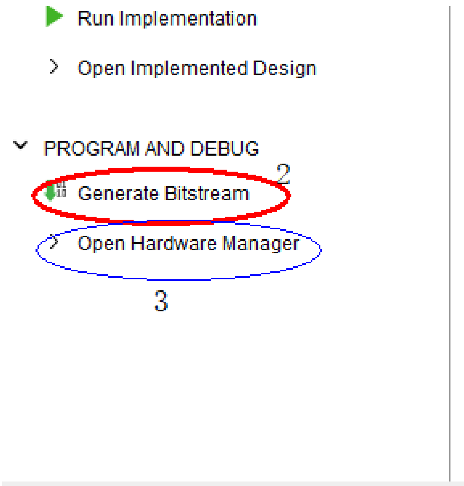


Fig 3. 7 Bit file generation

d. Click **Open target** to connect with the board. See Fig 3. 8.

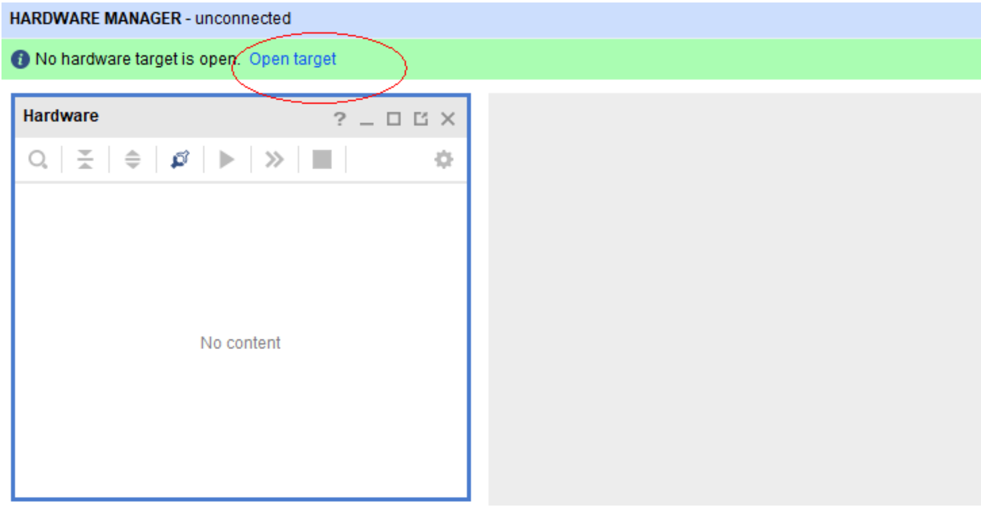


Fig 3. 8 Connect to the develop board

(6) Select the chip in step 1, right click to choose **Add Configuration Memory Device** in step 2. See Fig 3. 9.

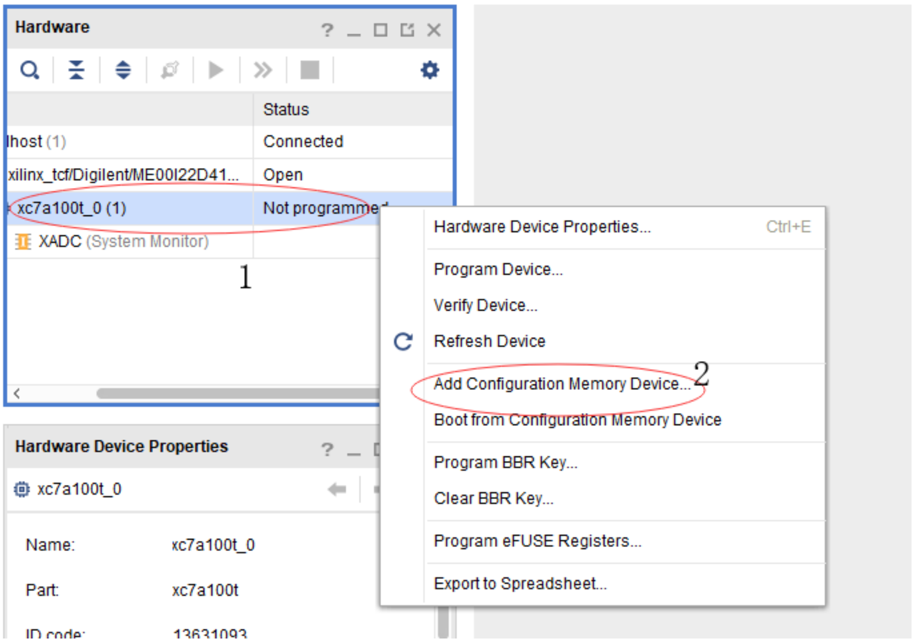


Fig 3. 9 Adding memory device

(7) Choose the Flash chip to be **mt25ql128**, then click **OK**. See Fig 3. 10.

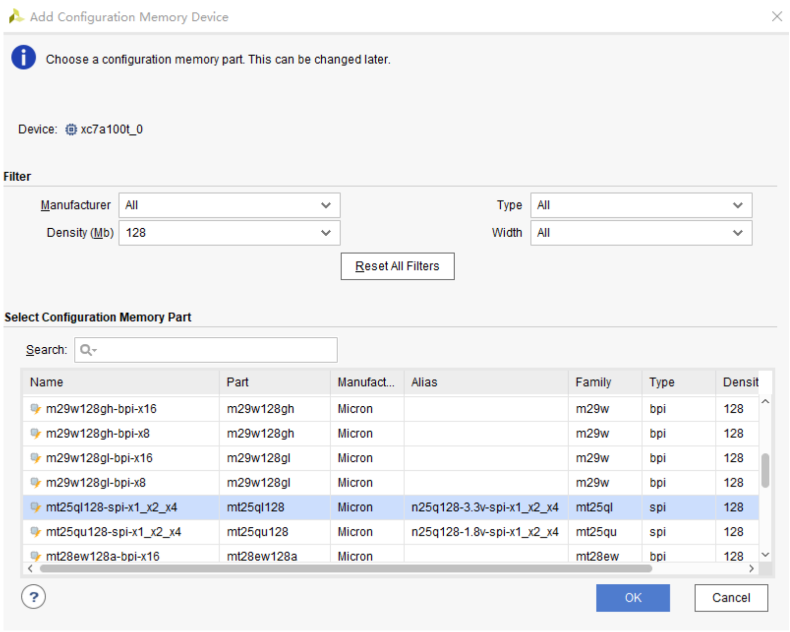


Fig 3. 10 Select Flash part

(8) Add bin file to be the **Configuration file**.

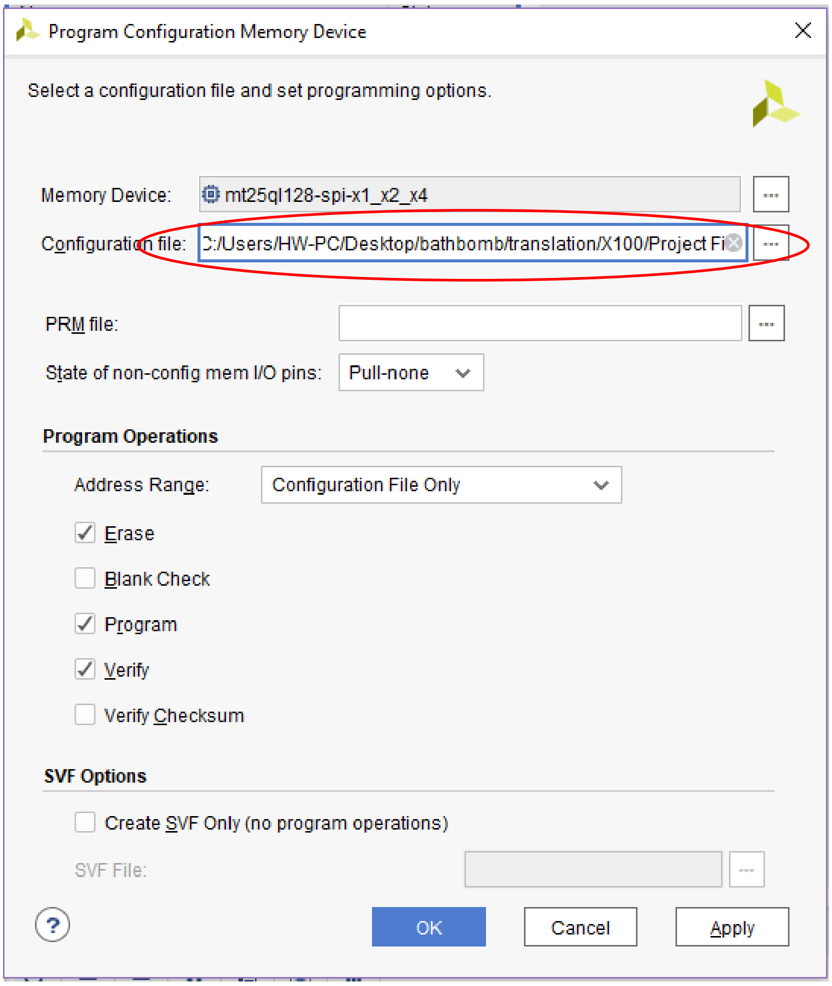


Fig 3. 11 Add the bin file

(9) The test result is shown in Fig 3. 12.

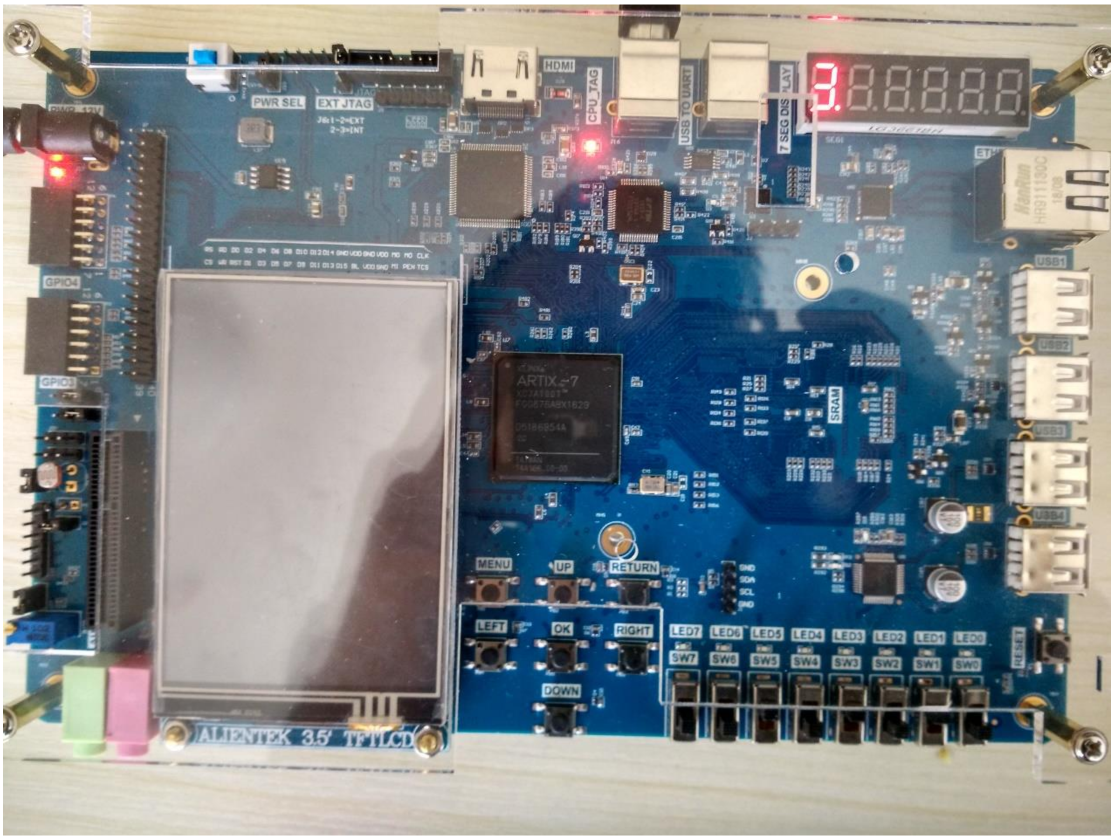


Fig 3. 12 Test result

Experiment 4 Block/SCH Digital Clock Design

1.Experiment Objective

(1) Review the new FPGA project building in Vivado, device selection, PLL creation, PLL frequency setting, Verilog tree hierarchy design, and the use of ILA

(2) Master the design method of graphics from top to bottom

(3) Combine the BCD_counter project to realize the movement of the decimal point (DP) of the decoder

(4) Observe the test result

2.Experiment Procedure

(1) File -> Project -> New

Project Name： block_counter

Select Device: **XC7A100T-2FGG676I**

(2) See Fig 4. 1, add source file, new top-level entity: *block_counter.v*

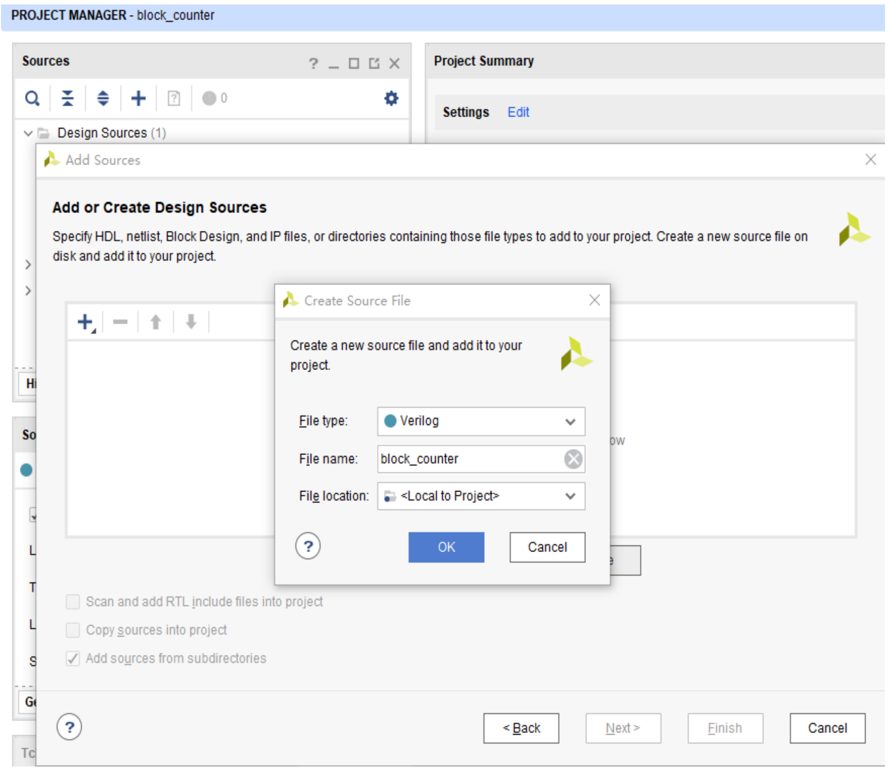


Fig 4. 1 Build source file

(3) As shown in Fig 4. 2, add the PLL as in the experiment 1, set the input clock to 50 MHz, and the output clock to 100 MHz.

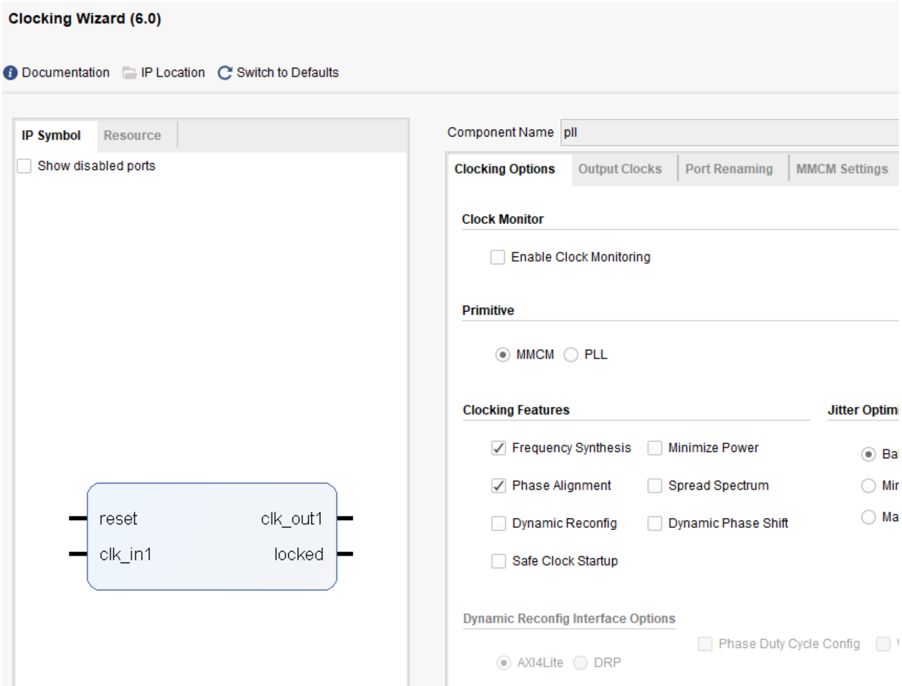


Fig 4. 2 Set the PLL IP core

(4) Create a new Verilog HDL file for the frequency divider

a. Divide the 100 MHz clock into a 1 MHz clock

```
module div_us(
input                   rst,
input                   sys_clk,
output           reg    us_f
);

reg[6:0] us_r;
always@(posedge sys_clk)
    if(rst)begin
            us_r<=0;
            us_f<=1'b0;
    end
    else begin
            us_f<=1'b0;
            if(us_r==99)begin
                us_r<=0;
                us_f<=1'b1;
            end
            else begin
                us_r<=us_r+1;
            end
    end
endmodule
```

b. Create a new 1000 division verilog HDL file again, *div_1000f.v*

```
module div_1000f
(
input                   rst,
input                   sys_clk,
input                   in_f,
output    reg   div1000_f

);

reg[9:0] div1000_r;

always@(posedge sys_clk)
        if(rst)begin
                div1000_r<=9'd0;
                div1000_f<=1'b0;
        end
        else begin
                div1000_f<=1'b0;
                if(in_f) begin
                        if(div1000_r==999)begin
                                div1000_r<=0;
                                div1000_f<=1'b1;
                        end
                        else begin
                        div1000_r<=div1000_r+1;
                        end
                end
        end
endmodule
```

(5) Use the 1000 frequency division program *div_1000f.v* to divide the 1 MHZ clock into 1000 HZ, 1 HZ clock.

```
module block_div(
               input wire sys_clk,
                input wire sys_rst,

                output wire us_f,
               output wire ms_f,
                output wire s_f
);



       div_us    div_us_inst(
```

```
                .rst        (sys_rst),
                .sys_clk (sys_clk) ,
                .us_f       (us_f)
    );

    div_1000f div_1000f_inst(
            .rst                (sys_rst) ,
            .sys_clk          (sys_clk) ,
            .in_f               (us_f     ) ,
            .div1000_f      (ms_f)
    );

    div_1000f div_1000f_inst2(
            .rst                (sys_rst) ,
            .sys_clk          (sys_clk) ,
            .in_f               (ms_f     ) ,
            .div1000_f      (s_f)
    );

endmodule
```

(6) Create a new Verilog file *bcd_counter.v*, design hour counter and minute counter

```
module bcd_counter(
        input                               rst,
        input                          sys_rst,
        input                          sys_clk,
        input                               ms_f,
        input                               s_f,
      output reg    [7:0]     seven_seg,
      output reg    [3:0]     scan

    );
  reg       ext_rst;
  reg           min_f;

  reg [3:0] counta,countb;
  reg       [3:0] countc,countd;
  reg [3:0] count_sel;

  reg     [6:0]seven_seg_r;



   always@(posedge sys_clk) begin
```

```
    ext_rst<=sys_rst;
    end


always@(posedge sys_clk)
if(ext_rst)begin
    counta<=0;
     countb<=0;
     min_f <=1'b0;
end
else begin
         min_f <=1'b0;
        if(s_f) begin
                if(counta==4'd9) begin
                    counta<=4'd0;
                        if(countb==5)begin
                         countb<=0;
                         min_f<=1'b1;
                          end
                        else
                        countb<=countb+1'b1;
                end
                else begin
                     counta<=counta+1'b1;
                end
        end

end
always@(posedge sys_clk)
if(ext_rst)begin
    countc<=4'd0;
     countd<=4'd0;
end
else begin

        if(min_f) begin
                if(countc==4'd9) begin
                    countc<=4'd0;
                        if(countd==5)begin
                         countd<=0;
                          end
                        else
                        countd<=countd+1'b1;
                end
```

```
                        else begin
                                countc<=countc+1'b1;
                        end
                end

    end

    reg [1:0]    scan_st;

    always@(posedge sys_clk)
       if(ext_rst) begin
         scan                <=4'b1111;
           count_sel         <=4'd0;
         scan_st<=0;
       end
     else case(scan_st)
    0:begin
         scan                <=4'b1110;
           count_sel         <=counta;
           if(ms_f)
         scan_st             <=1;
    end
    1:begin
           scan              <=4'b1101;
             count_sel       <=countb;
             if(ms_f)
           scan_st    <=2;
    end
    2:begin
           scan<=4'b1011;
             count_sel       <=countc;
             if(ms_f)
           scan_st<=3;
    end
    3:begin
           scan<=4'b0111;
             count_sel       <=countd;
             if(ms_f)
           scan_st<=0;
    end
    default:scan_st<=0;
    endcase

    always@(*)
```

```
case(count_sel)
0:seven_seg_r<=7'b100_0000;
1:seven_seg_r<=7'b111_1001;
2:seven_seg_r<=7'b010_0100;
3:seven_seg_r<=7'b011_0000;
4:seven_seg_r<=7'b001_1001;
5:seven_seg_r<=7'b001_0010;
6:seven_seg_r<=7'b000_0011;
7:seven_seg_r<=7'b111_1000;
8:seven_seg_r<=7'b000_0000;
9:seven_seg_r<=7'b001_0000;
default:seven_seg_r<=7'b100_0000;
endcase

  always@(posedge sys_clk)
  seven_seg<={1'b1,seven_seg_r};

endmodule
```

(7) Instantiate each function module subroutine into the top-level entity for comprehensive compilation.

```
module block_counter(
    input wire rst,
    input wire clk_in,

    output wire    [7:0]    seven_seg,
    output wire    [3:0]    scan
      );

     wire us_f;
     wire ms_f ;
     wire s_f ;

     reg    sys_rst;
     wire sys_clk;

     block_div block_div_inst(
          .sys_clk       (sys_clk)       ,
          .sys_rst       (sys_rst)     ,
          .us_f          (us_f)        ,
          .ms_f          (ms_f)              ,
          .s_f             (s_f)
     );
```

```
    always @(posedge   sys_clk)
         sys_rst <=!locked ;
    pll pll_inst
   (
    // Clock out ports
    .clk_out1(sys_clk),       // output clk_out1
    // Status and control signals
    .reset(1'b0), // input reset
    .locked(locked),          // output locked
   // Clock in ports
    .clk_in1(clk_in));        // input clk_in1

    bcd_counter bcd_counter_inst(
         .rst              (rst)          ,
         .sys_rst          (sys_rst)      ,
         .sys_clk          (sys_clk)      , //c0_50Mclk
         .ms_f             (ms_f)         ,
         .s_f              (s_f)          ,
         .seven_seg        (seven_seg)    ,
         .scan             (scan)
    );
endmodule
```

(8) Lock the Pin

Signal Name	Port Description	Network Label	FPGA Pin
inclk_in	Sytem clock 50 MHz	C10_50MCLK	U22
rst	Reset, high by default	KEY1	M4
seven_seg[0]	Segment a	SEG_PA	K26
seven_seg[1]	Segment b	SEG_PB	M20
seven_seg[2]	Segment c	SEG_PC	L20
seven_seg[3]	Segment d	SEG_PD	N21
seven_seg[4]	Segment e	SEG_PE	N22
seven_seg[5]	Segment f	SEG_PF	P21
seven_seg[6]	Segment g	SEG_PG	P23
seven_seg[7]	Segment h	SEG_DP	P24
scan[0]	Segment 6	SEG_3V3_D5	T24
scan[1]	Segment 5	SEG_3V3_D4	R25
scan[2]	Segment 4	SEG_3V3_D3	K25
scan[3]	Segment 3	SEG_3V3_D2	N18

(9) Compile, download to the board and test the program. The test result is shown in Fig 4. 3.

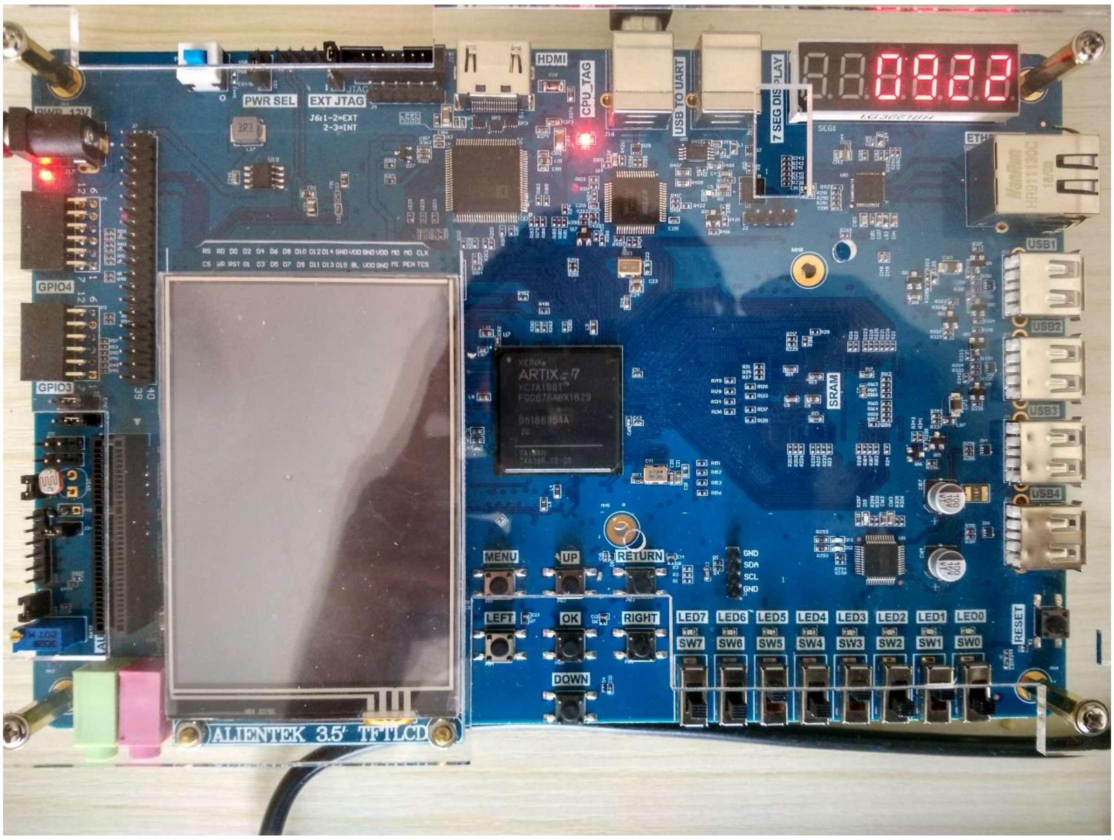


Fig 4. 3 Test result

3.More to Practice

(1) Practice the design of high-level digital clocks, month (positional system by base 30), day (positional system by base 24), hour (sexagesimal), and minute (sexagesimal).

(2) The content of this lab exercise is to use the design with a top-down design approach.

Experiment 5 Button Debounce Design and Experimental Experiment

1.Experiment Objective

(1) Review the design of blinking LED
(2) Learn the principle of button debounce, and adaptive programming
(3) Learn the connection and use of the FII-PRX100T button schematics
(4) Integrated application of button debounce and another compatible program design

2.Experiment

(1) Button debounce principle
Usually, the switches used for the buttons are mechanical elastic switches. When the mechanical contacts are opened and closed, due to the elastic action of the mechanical contacts, a push button switch does not immediately turn on when closed, nor is it off when disconnected. Instead, there is some bouncing when connecting and disconnecting. See Fig 5. 1

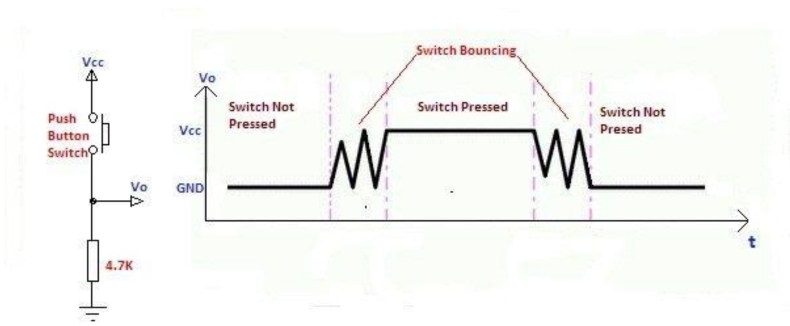


Fig 5. 1 Button bounce principle

The length of the button's stable closing time is determined by the operator. It usually takes more than 100ms. If you press it quickly, it will reach 40-50ms. It is difficult to make it even shorter. The bouncing time is determined by the mechanical characteristics of the button. It is usually between a few milliseconds and tens of milliseconds. To ensure that the program responds to the button's every on and off, it must be debounced. When the change of the button state is detected, it should not be immediately responding to the action, but waiting for the closure or the disconnection to be stabilized before processing. Button debounce can be divided into hardware debounce and software debounce.

In most of cases, we use software or programs to achieve debounce. The simplest debounce principle is to wait for a delay time of about 10ms after detecting the change of the button state, and then perform the button state detection again after the bounce disappears. If the state is the same as the previous state just detected, the button can be confirmed. The action has been stabilized. This type of detection is widely used in traditional software design. However, as the number of button usage increases, or the buttons of different qualities will react differently. If the delay is too short, the bounce cannot be filtered out. When the delay is too long, it affects the sensitivity of the button.

This chapter introduces an adaptive button debounce method: starts timing when a change in the state of the button is detected. If the state changes within 10ms, the button bouncing exists. It returns to the initial state, clears the delay counter, and re-detects the button state until the delay counter counts to 10ms. The same debounce method is used for pressing and releasing the button. The flow chart is shown in Fig 5. 2.

(2) Code for button debouncing

Verilog code is as follows:

```
module pb_ve(
        input      sys_clk,    //100 MHz
        input      sys_rst,//System reset
        input      ms_f,      //millisecond pulse
        input      keyin,    //input state of the key
        output    keyout //Output status of the key. Every time releasing the button, only one
system
        );                        //clock pulase outputs



                reg keyin_r;    //Input latch to eliminate metastable
                reg keyout_r;//Output pulse
        //push_button vibrating elemination
            reg      [1:0]      ve_key_st; //State machine status bit
            reg      [3:0]      ve_key_count;//delay counter



                always@(posedge sys_clk)
                keyin_r<=keyin;    // Input latch to eliminate metastable

            always@(posedge sys_clk)
            if(sys_rst) begin
                keyout_r            <=1'b0;
                ve_key_count      <=0;
                ve_key_st           <=0;
            end
            else case(ve_key_st)
            0:begin
                  keyout_r<=1'b0;
                  ve_key_count      <=0;
                  if(!keyin_r)
                  ve_key_st           <=1;
              end
          1:begin
                  if(keyin_r)
                  ve_key_st           <=0;
```

```
          else begin
               if(ve_key_count==10) begin
                 ve_key_st          <=2;
                end
               else if(ms_f)
                 ve_key_count<=ve_key_count+1;
          end
         end
     2:begin
             ve_key_count      <=0;
             if(keyin_r)
             ve_key_st          <=3;
           end
      3:begin
                if(!keyin_r)
                ve_key_st          <=2;
                else begin
                     if(ve_key_count==10) begin
                       ve_key_st          <=0;
                       keyout_r<=1'b1;//After releasing debounce, output a synchronized
                      end                    //clock pulse
                     else if(ms_f)
                       ve_key_count<=ve_key_count+1;

                end
          end
       default:;
       endcase
   assign keyout=keyout_r;
endmodule
```

Case 0 and 1 debounce the button press state. Case 2 and 3 debounce the button release state. After finishing the whole debounce procedure, the program outputs a synchronized clock pulse.

(3) Button debounce flow chart

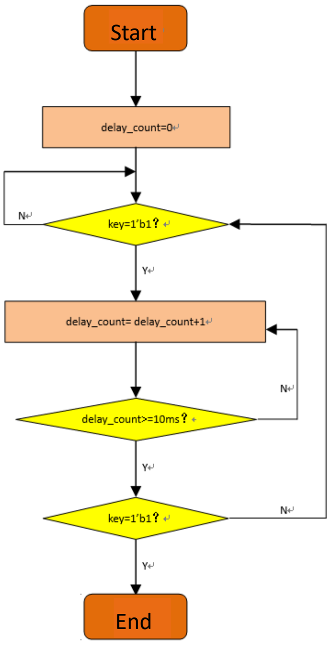


Fig 5. 2 Button debounce flow chart

(4) Combine running LED design and modify the button debounce.

a. Build new project
b. Create a PLL symbol
c. Create a button debounce symbol (See the Verilog HDL code in this experiment)
d. Create a top-level file key_filter

```
module key_filter(
            input clk_in,
            input left,
            input right,
            input wire rst,
            output wire [7:0] led

    );
    wire sys_rst_s= sys_rst;
    reg   sys_rst;
    wire   ms_f;
    wire   s_f ;
    wire   sys_clk;
    wire locked;
    wire left_flag, right_flag ;
    reg left_cmd=0;
    reg right_cmd =0;
 block_counter   block_counter_inst(
```

```
        . sys_rst ( sys_rst_s),
        . sys_clk (sys_clk),
         . ms_f(ms_f),
         . s_f (s_f)
            );
LED_shifting LED_shifting_inst    (
          . rst(sys_rst_s) ,
            . sys_clk(sys_clk),
             .key_left(left_cmd),
            .key_right(right_cmd),
            .s_f(s_f),
          . led(led)
            );
     pb_ve pb_ve_inst1(
              .sys_clk   (sys_clk),
              .sys_rst   (sys_rst_s),
              .ms_f              (ms_f       ),
              .keyin             (left  ),
              .keyout       (left_flag )
              );
     pb_ve pb_ve_inst2(
              .sys_clk   (sys_clk),
              .sys_rst   (sys_rst_s),
              .ms_f              (ms_f       ),
              .keyin             (right      ),
              .keyout       (right_flag )
              );
     always @ ( posedge sys_clk )
        if    (sys_rst_s)
                 {right_cmd,left_cmd}<=2'b00;
        else begin
          case({right_flag,left_flag})
           0:    {right_cmd,left_cmd}<={right_cmd,left_cmd};
           1:    {right_cmd,left_cmd}<=2'b01;
           2:    {right_cmd,left_cmd}<=2'b10;
           3:    {right_cmd,left_cmd}<={right_cmd,left_cmd};
          endcase
     end
     always @ (posedge sys_clk )
              sys_rst<=!locked;
          pll pll_inst(
            .clk_out1(sys_clk),
            .reset(!rst),
            .locked(locked),
```

```
        .clk_in1(clk_in)
    );

endmodule
```

3.Hardware Design

(1) Button schematics

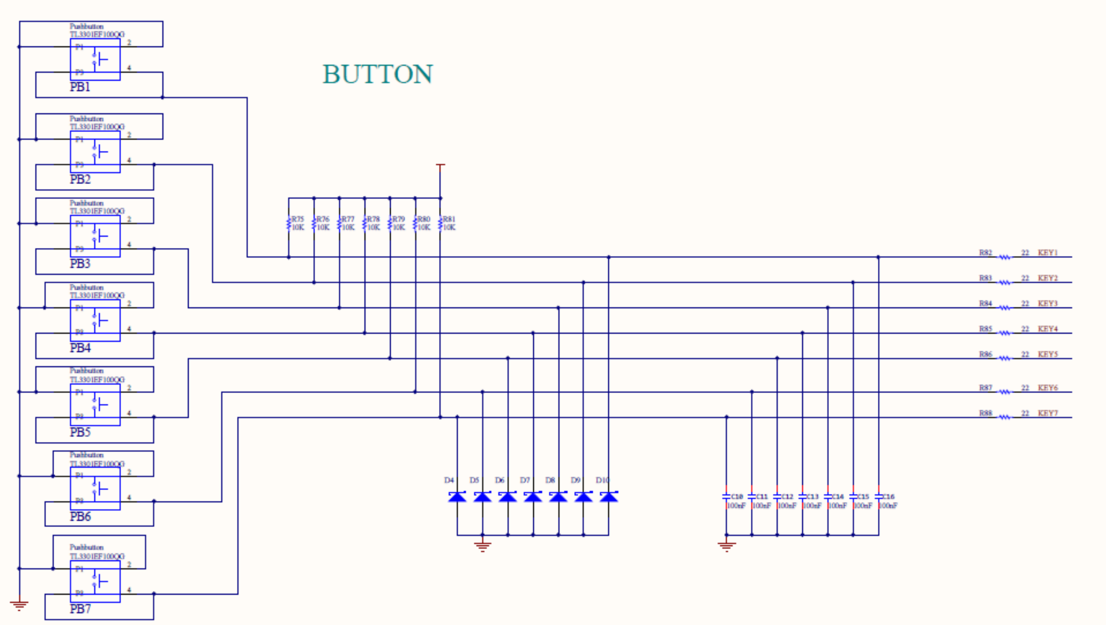


Fig 5. 4 Button schematics

(2) FPGA pin mapping

Signal Name	Port Description	Network Label	FPGA Pin
inclk_in	System clock 50 MHz	C10_50MCLK	U22
rst	Reset, high by default	KEY1	M4
led0	LED 0	LED0	N17
led1	LED 1	LED1	M19
led2	LED 2	LED2	P16
led3	LED 3	LED3	N16
led4	LED 4	LED4	N19
led5	LED 5	LED5	P19
led6	LED 6	LED6	N24
led7	LED 7	LED7	N23
left	Press left	KEY4	K5
right	Press right	KEY6	P1

a. Compile and debug
b. Download the program to the board and observe the test result. See Fig 5. 5

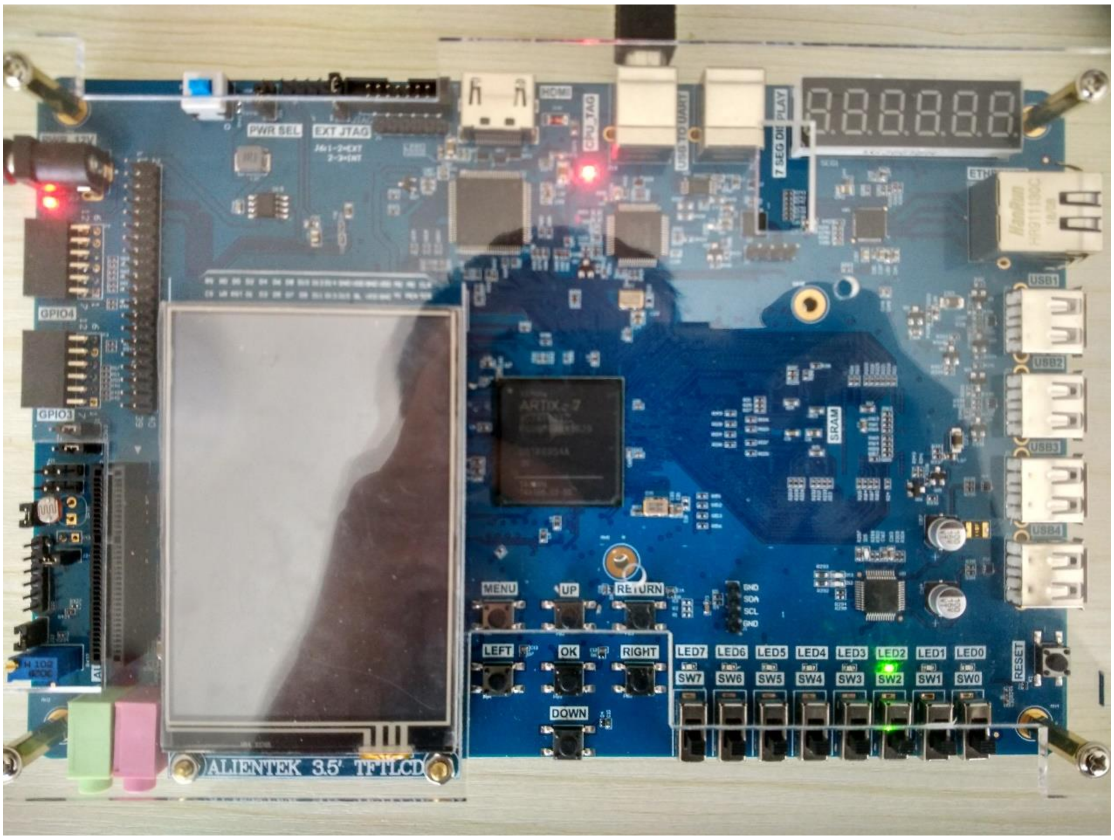


Fig 5. 5 Test Result

(3) Observe the test results. By default, 8 LEDs are off. Press the left button to switch the flow mode on the left side of the LED. Press the right button on the right side of the LED to switch between the flow mode. While holding down the left and right buttons, the LED remains in its original state.

Experiment 6 Digital Clock Comprehensive Design Experiment

1.Experiment Objective

(1) Design month, day, hour, minute, and second digital clock experiments, using 6 segment decoders
   a. 60 seconds carried to the minute
   b. 60 minutes carried to the hour
   c. 24 hours carried to the day
   d. 30 days carried to the month, and reset all

(2) Set four keys: *menu, left, up, down*
   a. The *menu* key controls the calibration function to switch between clock, date, and alarm.
   b. The *left* key selects which value is currently calibrated
   c. The *Up* and *down* keys add 1 and subtract 1 calibration to the data to be calibrated requires that the corresponding segment decoder is flashed.
   d. Modulate the design so that it can be reused

(3) Learn to use the module parameters

(4) Learn to use the timing analysis function of Vivado and correctly constrain the clock signal

2.Design Procedure

(1) Build new project
   a. Project name is *calendar_counter*
   b. Select the device **XC7A100TFGG676-2**
   c. The top-level entity is *calendar_counter.bdf* or *calendar_counter.v* (Here the Verilog file is used)

(2) Design and integrate of submodule
   a. PLL module
   b. Frequency divider
   c. Button debounce module
   d. Counting module *dual_num_count.v*
      Design a universal 2-bit counter that uses the parameter to specify the specified count setting.

```
Module dual_num_count
#(parameter PAR_COUNTA=9,
parameter     PAR_COUNTB=5
)
(
      input                    i_sys_clk,
```

```
        input               i_ext_rst,
        input               i_adj_up,
        input               i_adj_down,
        input         [1:0] i_adj_sel,
        input               i_trig_f,
        output reg          o_trig_f,
        output reg [3:0] o_counta,
        output reg [3:0] o_countb
);
always@(posedge i_sys_clk)
  if(!i_ext_rst)begin
        o_counta <=0;
          o_countb      <=0;
          o_trig_f <=1'b0;
  end
  else begin
        o_trig_f<=1'b0;
        if(i_adj_up)begin
                if(!i_adj_sel[0])begin
                        if(o_counta==9)
                                o_counta<=0;
                        else
                                o_counta<=o_counta+1;
                        end
                else if(!i_adj_sel[1])
                        begin
                                if(o_countb==9)
                                o_countb<=0;
                                else
                                o_countb<=o_countb+1;
                        end
                end
        else if(i_adj_down) begin
                        if(!i_adj_sel[0])begin
                                if(o_counta==0)
                                o_counta<=9;
                                else
                                o_counta<=o_counta-1;
                        end
                            else if(!i_adj_sel[1])begin
                                if(o_countb==0)
                                o_countb<=9;
                                else
                                o_countb<=o_countb-1;
```

```
                        end
        end
        else if(i_trig_f) begin
        if((o_countb==PAR_COUNTB)&&(o_counta==PAR_COUNTA))
                    begin
                        o_counta<=4'd0;
                        o_countb<=0;
                        o_trig_f<=1'b1;
                    end
                    else begin
                        if(o_counta==9)begin
                                o_counta<=4'd0;
                                o_countb<=o_countb+1;
                        end
                        else begin
                                o_counta<=o_counta+1;
                        end
                end
              end
    end
endmodule
```

(3) Button debounce

```
module pb_ve(
        input     sys_clk,
        input     sys_rst,
        input     ms_f,
        input     keyin,
        output    keyout
        );
                reg keyin_r;
        reg keyout_r;
        //push_button vibrating elemination
            reg     [1:0]     ve_key_st;
            reg     [3:0]     ve_key_count;
  always@(posedge sys_clk)
          keyin_r<=keyin;
            always@(posedge sys_clk)
            if(sys_rst) begin
                keyout_r              <=1'b0;
                ve_key_count     <=0;
                ve_key_st            <=0;
            end
            else case(ve_key_st)
            0:begin
```

```
          keyout_r<=1'b0;
          ve_key_count     <=0;
          if(!keyin_r)
          ve_key_st          <=1;
        end
      1:begin
          if(keyin_r)
          ve_key_st          <=0;
          else begin
               if(ve_key_count==10) begin
                 ve_key_st          <=2;
                end
               else if(ms_f)
                 ve_key_count<=ve_key_count+1;

          end
         end
     2:begin
             ve_key_count    <=0;
             if(keyin_r)
             ve_key_st       <=3;
           end
      3:begin
                if(!keyin_r)
                ve_key_st          <=2;
                else begin
                     if(ve_key_count==10) begin
                       ve_key_st          <=0;
                       keyout_r<=1'b1;
                      end
                     else if(ms_f)
                       ve_key_count<=ve_key_count+1;
                end
          end
       default:;
       endcase
   assign keyout=keyout_r;
endmodule
```

(4) Top-level entity design

```
module calendar_counter(

    input                    rst,
```

```
    input                left,   //key4
    input                right,
     input               up,
     input               down,
    input                inclk, //c0_50Mclk
    output  reg   [6:0]   seven_sega,
    output   reg        disp_pa,
    output  reg   [5:0]   scan

    );

wire   sys_clk;
wire   pll_locked;
reg     sys_rst;
reg     ext_rst;

reg [7:0] us_reg;
reg [9:0] ms_reg;
reg [9:0] s_reg;
reg           us_f,ms_f,s_f;
wire      min_f,hr_f,day_f;

reg [3:0] counta;

wire [3:0] count_secl,count_sech;
wire [3:0] count_minl,count_minh;
wire [3:0] count_hrl,count_hrh;

wire [3:0] count_dayl,count_dayh;

reg     [6:0]seven_seg_ra;
reg             [7:0]disp_p_r;

wire           left_r,right_r;
wire           up_r,down_r;


  always@(posedge sys_clk) begin
    sys_rst<=!pll_locked;
    ext_rst<=rst;
    end

always@(posedge sys_clk)
if(sys_rst) begin
```

```
        us_reg<=0;
        us_f<=1'b0;
    end
    else begin
        us_f<=1'b0;
        if(us_reg==99)begin
                us_reg<=0;
                us_f<=1'b1;
        end
        else begin
                us_reg<=us_reg+1;
        end

    end

      always@(posedge sys_clk)
      if(sys_rst) begin
          ms_reg<=0;
          ms_f<=1'b0;
      end
      else begin
          ms_f<=1'b0;
                         if(us_f) begin

if(ms_reg==999)begin
     ms_reg<=0;
     ms_f<=1'b1;
     end
     else

     ms_reg<=ms_reg+1;
     end
      end

always@(posedge sys_clk)
          if(sys_rst) begin
              s_reg<=0;
              s_f<=1'b0;
          end
          else begin
              s_f<=1'b0;
  if(ms_f)begin
if(s_reg==999)begin
       s_reg<=0;
```

```
     s_f<=1'b1;
      end
      else
s_reg<=s_reg+1;
      end
       end
dual_num_count
#(.PAR_COUNTA(9),
.PAR_COUNTB(5)
)
  dual_num_count_sec
(
.i_sys_clk  (sys_clk),
.i_ext_rst      (ext_rst),
.i_adj_up       (up_r),
.i_adj_down (down_r),
.i_adj_sel      (disp_p_r[1:0]),
.i_trig_f     (s_f),
.o_trig_f (min_f),
.o_counta (count_secl),
.o_countb (count_sech)


);

  dual_num_count
#(.PAR_COUNTA(9),
.PAR_COUNTB(5)
)
  dual_num_count_min
(
.i_sys_clk(sys_clk),
.i_ext_rst  (ext_rst),
.i_adj_up  (up_r),
.i_adj_down (down_r),
.i_adj_sel  (disp_p_r[3:2]),
.i_trig_f        (min_f),
.o_trig_f   (hr_f),
.o_counta (count_minl),
.o_countb (count_minh)


);
```

```
  dual_num_count
#(.PAR_COUNTA(3),.PAR_COUNTB(2))
  dual_num_count_hr
(
.i_sys_clk      (sys_clk),
.i_ext_rst  (ext_rst),
.i_adj_up      (up_r),
.i_adj_down (down_r),
.i_adj_sel      (disp_p_r[5:4]),
.i_trig_f    (hr_f),
.o_trig_f   (day_f),
.o_counta (count_hrl),
.o_countb (count_hrh)


);


  dual_num_count
#(.PAR_COUNTA(0),
.PAR_COUNTB(3)
)
  dual_num_count_day
(
.i_sys_clk  (sys_clk),
.i_ext_rst  (ext_rst),
.i_adj_up      (up_r),
.i_adj_down (down_r),
.i_adj_sel  (disp_p_r[7:6]),
.i_trig_f    (day_f),
.o_trig_f   (),
.o_counta (count_dayl),
.o_countb (count_dayh)


);

  always@(posedge sys_clk)
     if(!ext_rst) begin
       disp_p_r<=8'b1111_1110;
     end
  else begin
                 if(left_r)
                 disp_p_r<={disp_p_r[6:0],disp_p_r[7]};
```

```
                  else if(right_r)
                  disp_p_r<={disp_p_r[0],disp_p_r[7:1]};
  end
  reg [2:0]    scan_st;

  always@(posedge sys_clk)
     if(!ext_rst) begin
       scan<=6'b11_1111;
                    counta<=4'b0;
                    disp_pa<=1'b1;
         scan_st<=0;
     end
    else case(scan_st)
0:begin
       scan  <=6'b11_1110;
        counta <=count_secl;
        disp_pa<=disp_p_r[0];
                    if(ms_f)
       scan_st<=1;
end
1:begin
         scan<=6'b11_1101;
                      counta<=count_sech;
                      disp_pa<=disp_p_r[1];
             if(ms_f)
             scan_st<=2;
end
2:begin
         scan<=6'b11_1011;
                      counta<=count_minl;
                      disp_pa<=disp_p_r[2];
             if(ms_f)
             scan_st<=3;
end
3:begin
         scan<=6'b11_0111;
                      counta<=count_minh;
                      disp_pa<=disp_p_r[3];
         if(ms_f)
               scan_st<=4;
end
4:begin
         scan<=6'b10_1111;
                      counta<=count_hrl;
```

```
                    disp_pa<=disp_p_r[4];
         if(ms_f)
              scan_st<=5;

end
5:begin
         scan<=6'b01_1111;
                    counta<=count_hrh;
                    disp_pa<=disp_p_r[5];
         if(ms_f)
              scan_st<=0;

end
default:scan_st<=0;
endcase

always@(*)
case(counta)
0:seven_seg_ra<=7'b100_0000;
1:seven_seg_ra<=7'b111_1001;
2:seven_seg_ra<=7'b010_0100;
3:seven_seg_ra<=7'b011_0000;
4:seven_seg_ra<=7'b001_1001;
5:seven_seg_ra<=7'b001_0010;
6:seven_seg_ra<=7'b000_0010;
7:seven_seg_ra<=7'b111_1000;
8:seven_seg_ra<=7'b000_0000;
9:seven_seg_ra<=7'b001_0000;
default:seven_seg_ra<=7'b100_0000;
endcase
   always@(posedge sys_clk)
   seven_sega<=seven_seg_ra;

pb_ve pb_ve_left
(
.sys_clk          (sys_clk),
.sys_rst          (sys_rst),
.ms_f             (ms_f),
.keyin            (left),
.keyout           (left_r)
);

pb_ve pb_ve_right
(
```

```
.sys_clk            (sys_clk),
.sys_rst            (sys_rst),
.ms_f               (ms_f),
.keyin              (right),
.keyout             (right_r)
);

pb_ve pb_ve_up
(
.sys_clk            (sys_clk),
.sys_rst            (sys_rst),
.ms_f               (ms_f),
.keyin              (up),
.keyout             (up_r)
);

pb_ve pb_ve_down
(
.sys_clk            (sys_clk),
.sys_rst            (sys_rst),
.ms_f               (ms_f),
.keyin              (down),
.keyout             (down_r)
);

calendar_pll calendar_pll
(
            .reset   (1'b0),
            .inclk0  (inclk),
            .c0      (sys_clk),
            .locked (pll_locked)
            );

endmodule
```

(5) Lock the Pins

Signal Name	Port Description	Network Label	FPGA Pin
inclk	System clock, 50 MHz	C10_50MCLK	U22
rst	Reset, high by default	KEY1	M4
seven_seg[0]	Segment a	SEG_PA	K26
seven_seg[1]	Segment b	SEG_PB	M20
seven_seg[2]	Segment c	SEG_PC	L20
seven_seg[3]	Segment d	SEG_PD	N21
seven_seg[4]	Segment e	SEG_PE	N22
seven_seg[5]	Segment f	SEG_PF	P21

seven_seg[6]	Segment g	SEG_PG	P23
seven_seg[7]	Segment h	SEG_DP	P24
scan[0]	Segment 6	SEG_3V3_D5	T24
scan[1]	Segment 5	SEG_3V3_D4	R25
scan[2]	Segment 4	SEG_3V3_D3	K25
scan[3]	Segment 3	SEG_3V3_D2	N18
scan[4]	Segment 2	SEG_3V3_D1	R17
scan[5]	Segment 1	SEG_3V3_D0	R16
left	Left button	KEY4	K5
right	Right button	KEY6	P1
up	Up button	KEY2	L4
down	Bottom button	KEY2	R7

(6) Compile

(7) Download the program to the develop board for verification

a. Observe the test result
b. Use the left, right keys to move the decimal point of the segment decoder
c. Use up, down keys to calibrate time

The test result is shown in Fig 6. 1, displaying time 10:27:05

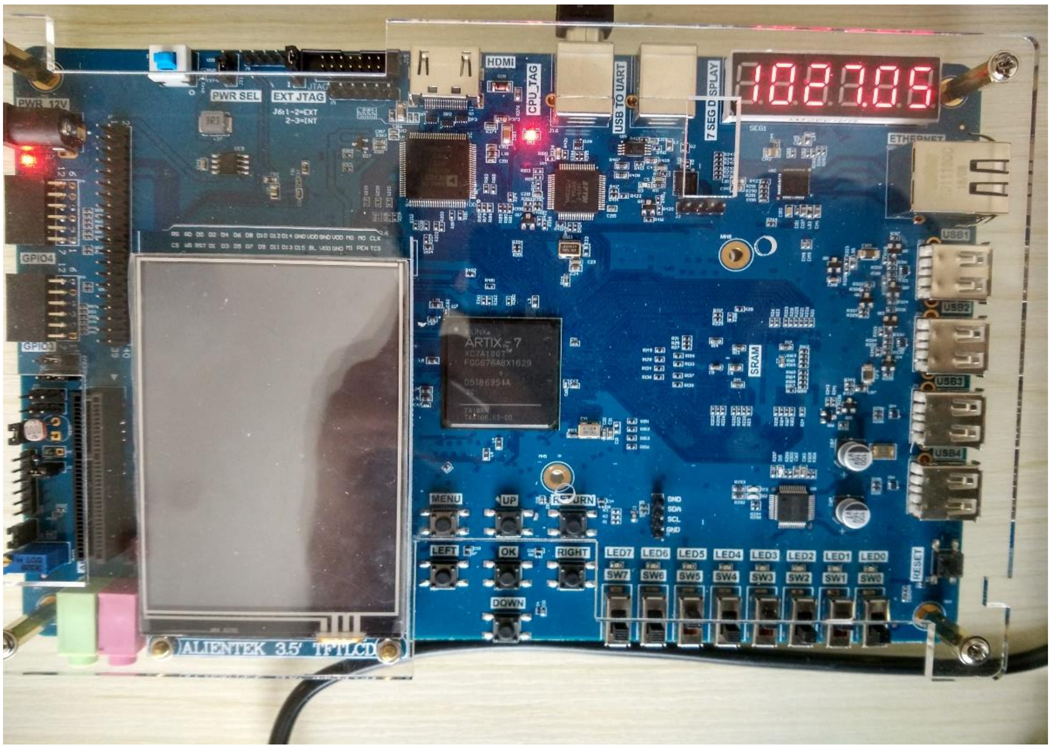


Fig 6. 1 Test result

3.Create an XDC File to Constrain the Clock

(1) Create constrain file

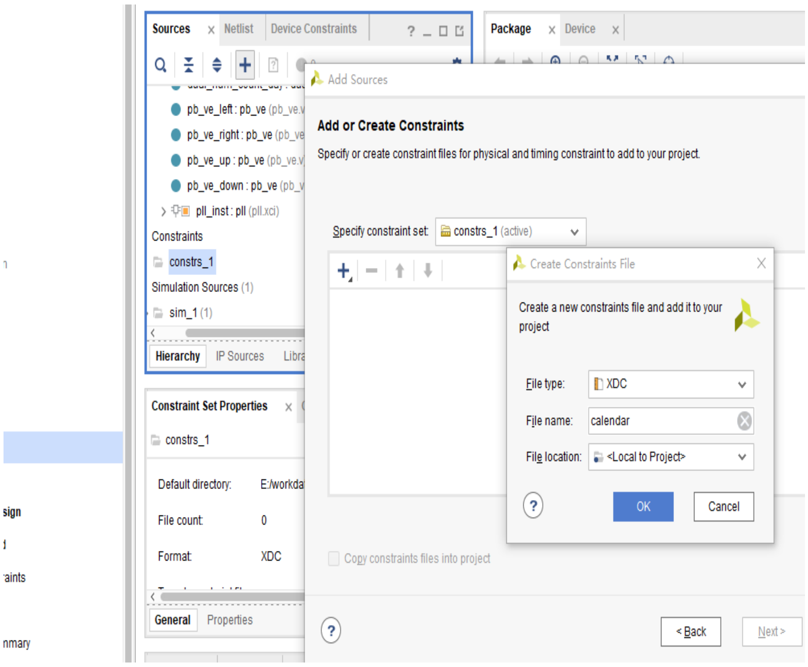


Fig 6. 2 Craete SDC file

XDC file is as follows:

Create Clock

create_clock -period 20 -name inclk -waveform {0.000 10.000} [get_ports inclk]

(2) Improve the precision when using up, down to calibrate

a. The maximum value is automatic recognized, such as in the sexagesimal decimal digit calibration time, if the value reaches 5, the next Up will make the value become 0. When the timing of Down is reduced to 0, the next Down pulse will automatically change to 5.
b. Compile, and download the program to the develop board
c. Program to the flash memory

Experiment 7 Multiplier Use and ISIM Simulation

1.Experiment Objective

(1) Learn to use multiplier

(2) Use ISIM to simulate design output

2.Experiment Design

(1) Build new project *mult_sim*

a. Select device **XC7A100TFGG676-2**

(2) Design implement

a. 8x8 multiplier, the first input value is an 8-bit switch, and the second input value is the output of an 8-bit counter.

b. Observe the result on Modelsim

c. Observe the result on 6 segment decoders

(3) Design procedure

a. Create new file *mult_sim.v*

b. Add PLL， set the input clock to be 50 MHz, and the output clock to be 100 MHz

c. Add LPM_MULT IP

IP Catalog -> input **Mult** in the search box. Invoke the multipliers. See Fig 7. 1.

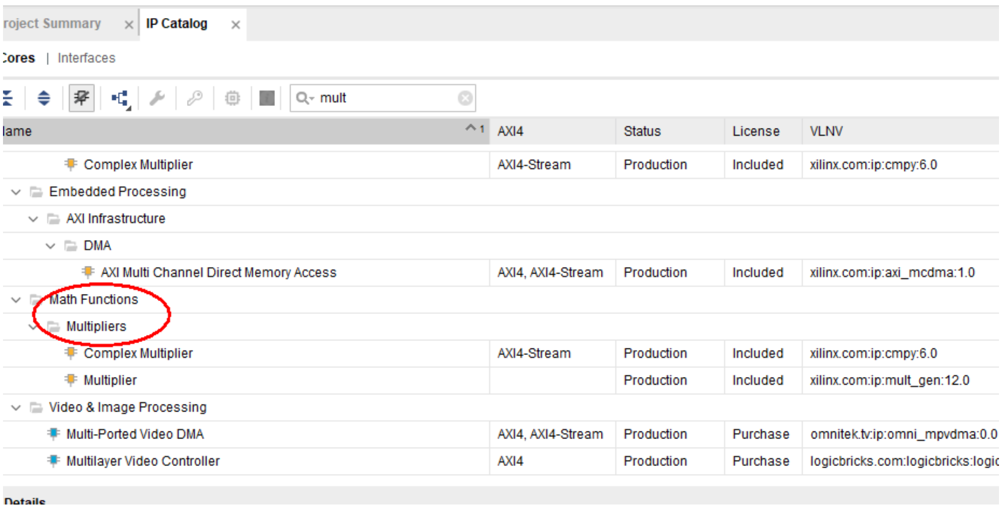


Fig 7. 1 Build IP core for multiplier

d. Choose input data type to be **unsigned** and width to be **8**. See Fig 7. 2.

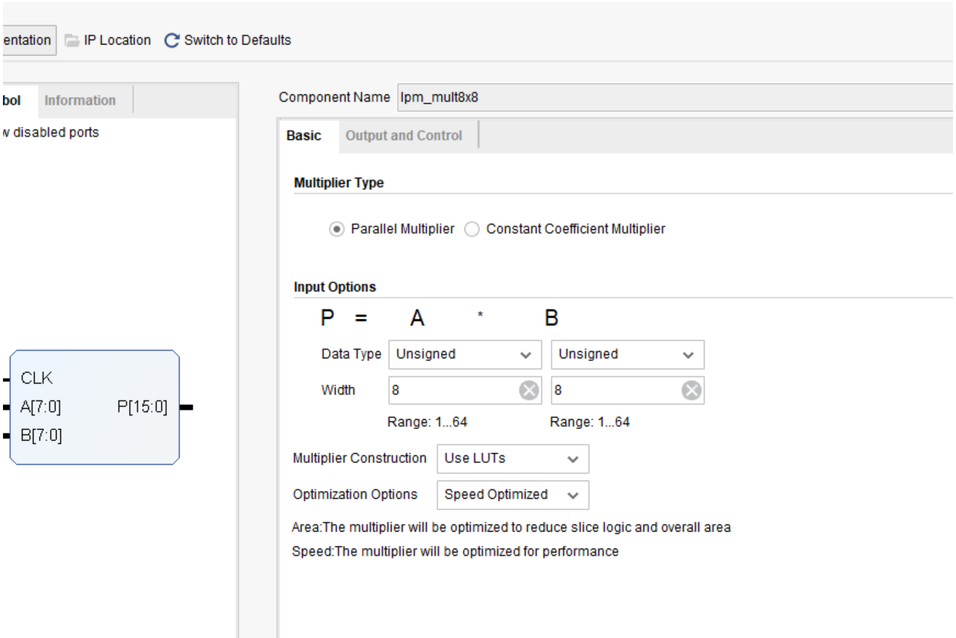


Fig 7. 2 Set the input data type and data width

e. Choose **Pipelining and Control Signals**. See Fig 7. 3. Add a delay of 1 stage. The default optimum stage is 3 stages.

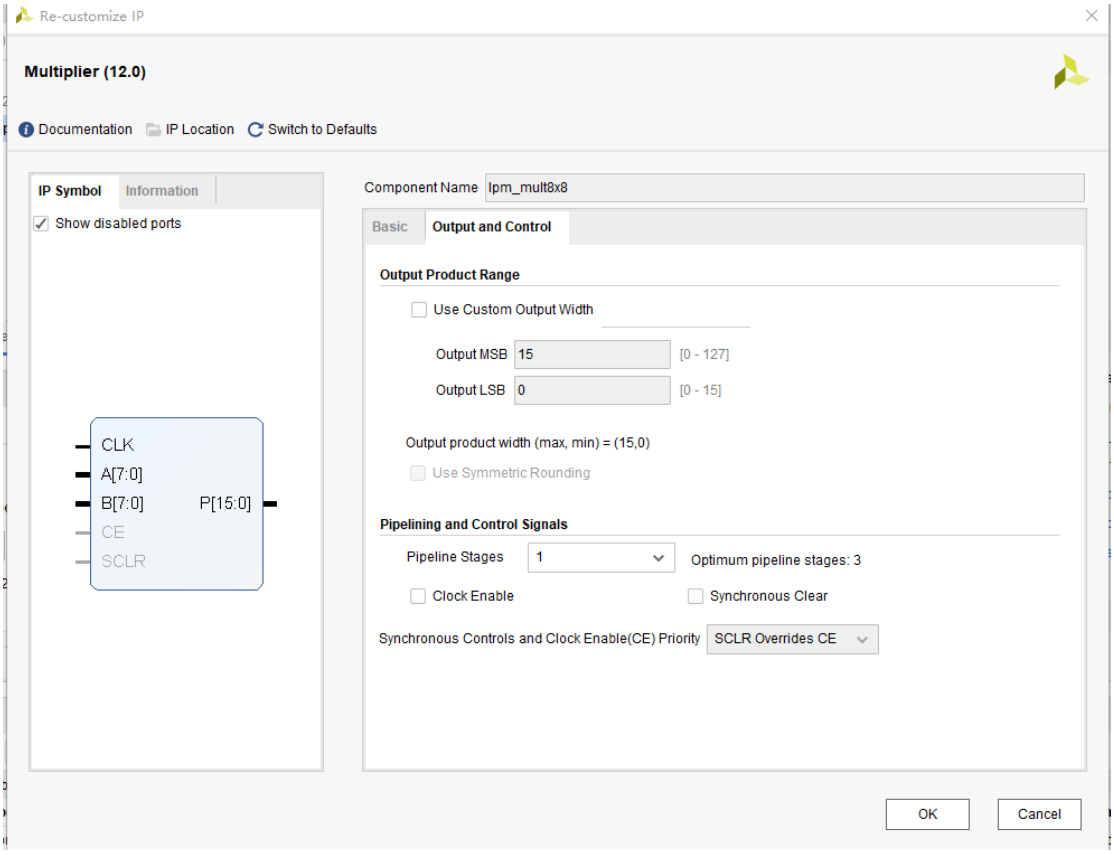


Fig 7. 3 Pipelining setting

(4) Choose default for other settings

(5) Instantiate in the top-level entity

3.The Top-level Entity Is as Follows:

```
module mult_sim
(
input       rst,
input       inclk,
input         [7:0] sw,
output     [6:0] seven_seg,
output     [3:0] scan
);

wire [15:0]    mult_res;
wire              sys_clk;
wire              sys_rst;
reg   [7:0]      count;
always@(posedge sys_clk)
if(sys_rst)
count<=0;
else
count<=count+1;

lpm_mult8x8
  (
.clock       (sys_clk),
.dataa       (sw),
.datab       (count),
.result      (mult_res)
       );
pll_sys_rst pll_sys_rst_inst
(
.inclk       (inclk),
.sys_clk   (sys_clk),
.sys_rst    (sys_rst)
);
endmodule
```

4.ISIM Simulation Library Compilation and Call

Under the Vivado platform, you can choose to use built-in simulation tool ISIM or third-party simulation tools for functional simulation of the project. Simulating with the Modelsim simulation tool requires a separate compilation of the simulation library. This routine uses the built-in ISIM tool emulation and briefly introduce Modelsim's Xilinx simulation library file compilation for simulation using Modelsim.

(1) Build simulation project files.

Add the testbench file under **Simulation Sources**. See Fig 7. 4.

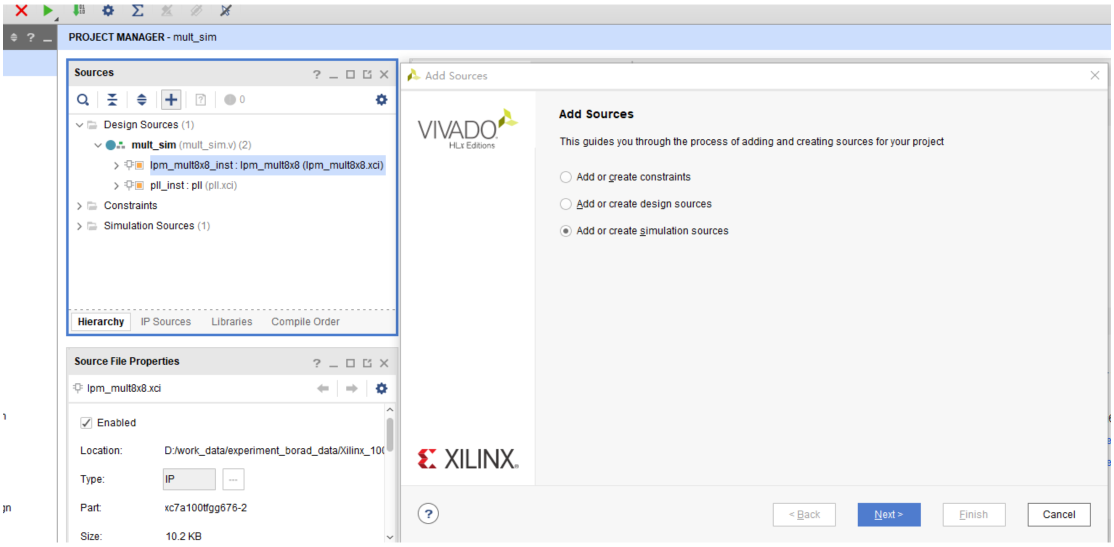


Fig 7. 4 Add the testbench file

Simulation testbench code is as follows:

```
module mult_sim_tb;
    //Define simulation signals
 reg         rst_n;
 reg [7:0]       sw;
 reg                clk;

 wire     [7:0]      seven_seg;
 wire      [3:0]    scan;
 wire      [15:0] mult_res;
 wire     [7: 0] count ;
 mult_sim    mult_sim_inst
(
        .rst_n(rst_n),
        .inclk(clk),
        .sw(sw),
        .count(count),
        .mult_res(mult_res),
        .seven_seg(seven_seg),
        .scan(scan)
 );

 initial
 begin
```

```
  rst_n=0;
  clk = 1;
  sw    = 0;
  #5 rst_n=1;
  #15 sw = 20;
  #20 sw = 50;
  #20 sw = 100;
  #20 sw = 101;
  #20 sw = 102;
  #20 sw = 103;
  #20 sw = 104;
  #50 sw = 105;
  $monitor("%d * %d=%d", count, sw, mult_res);
  #1000000 $stop;
  end
  always
  #10 clk=~clk;
endmodule
```

(2) As shown in Fig 7. 5, after the simulation stimulus file is added, ISIM can be started in **Simulation->Run Simulation --> Run Behavioral Simulation** on the left side of the project management.

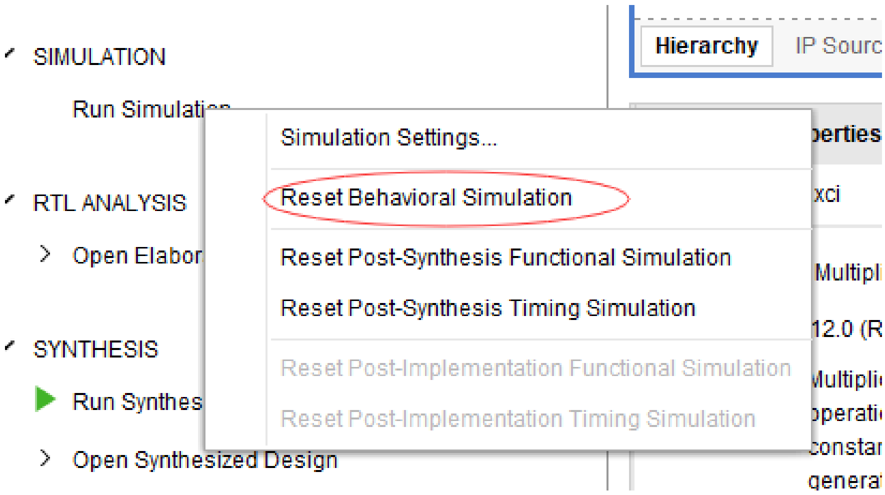


Fig 7. 5 Simulation library compiled

(3) Simulation result is shown in Fig 7. 6.

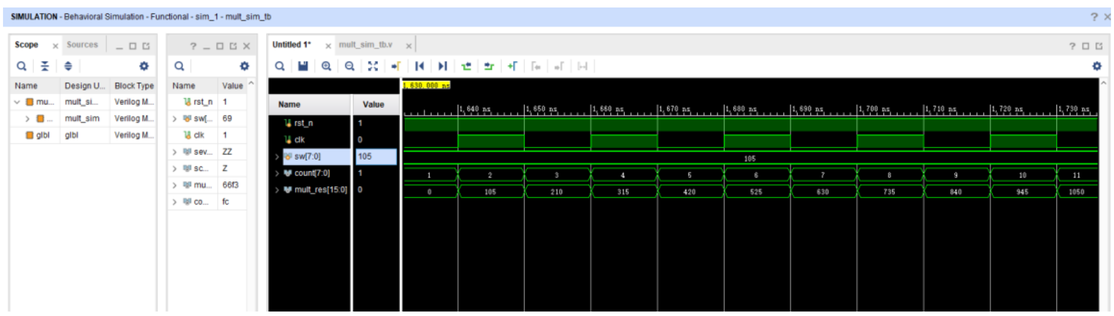


Fig 7. 6 Simulation result

(4) Compile ModelSim library

After installing ModelSim, compile the Xilinx simulation library file first. The specific process is as follows:

a. **Tools -> Compile Simulation Libraries**. See Fig 7. 7 for the popup window.

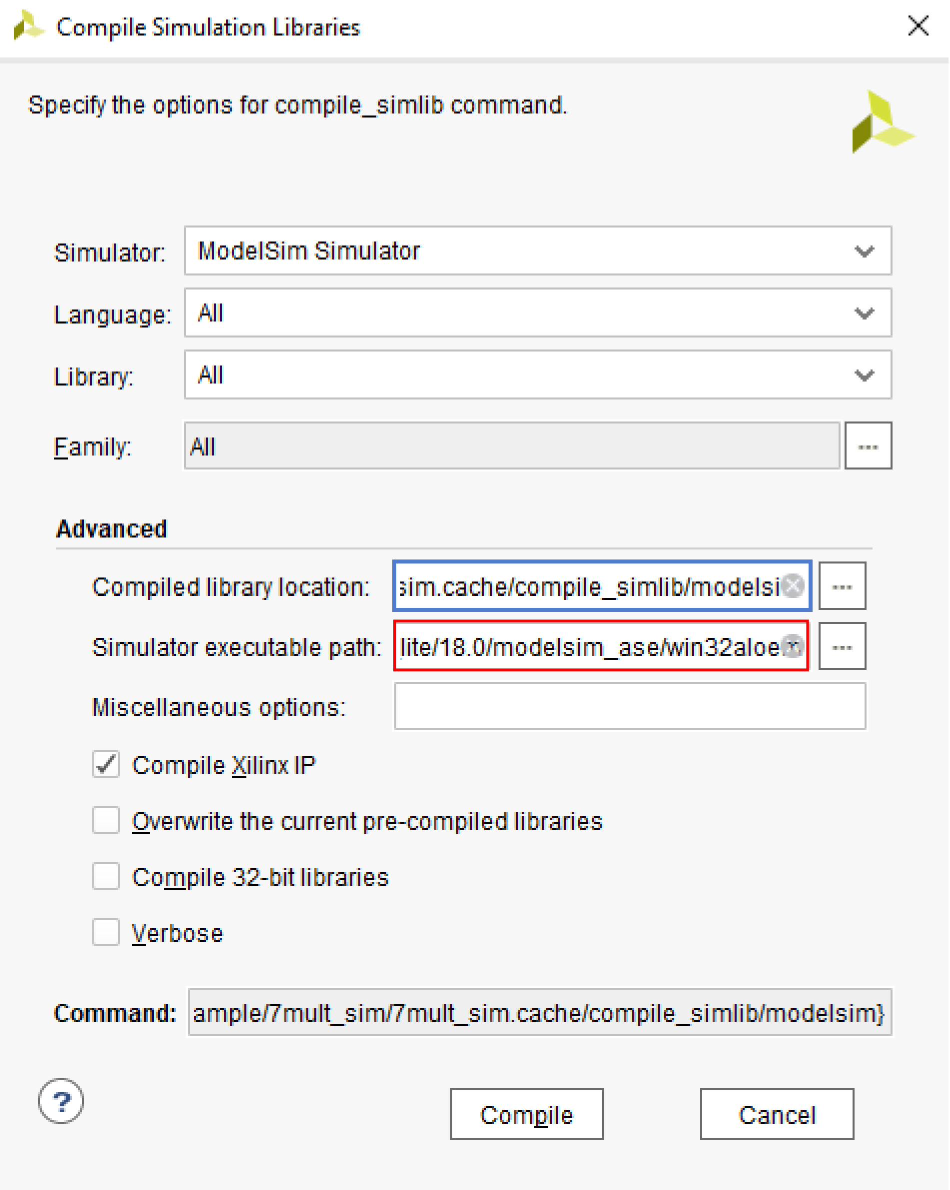


Fig 7. 7 Compilation library address setting

b. As shown in Fig 7. 8, the compilation is completed. Note that the process is very time consuming.

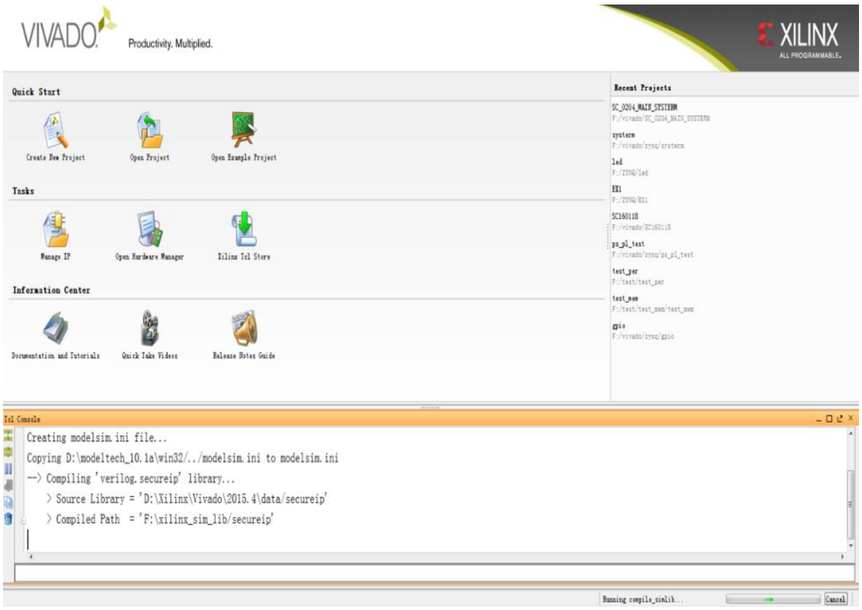


Fig 7. 8 Simulation library compiled

Approachable advanced information for ModelSim can be referred online. Here would not go into more details.

(5) More to practice

a. Design an 8-bit trigger, simulate with Modelsim
b. Learn to write testbenchs for simulation

Experiment 8 Hexadecimal Number to BCD Code Conversion and Application

1.Experiment Objective

(1) Since the hexadecimal display is not intuitive, decimal display is more widely used in real life.

(2) Human eye recognition is relatively slow, so the display from hexadecimal to decimal does not need to be too fast. Generally, there are two methods

a. Countdown method: Under the control of the synchronous clock, the hexadecimal number is decremented by 1 until it is reduced to 0. At the same time, the appropriate BCD code decimal counter is designed to increment. When the hexadecimal number is reduced to 0, the BCD counter just gets with the same value to display.

b. Bitwise operations (specifically, shift bits and plus 3 here). The implementation is as follows:

1) Set the maximum decimal value of the expression. Suppose you want to convert the 16-digit binary value (4-digit hexadecimal) to decimal. The maximum value can be expressed as 65535. First define five four-digit binary units: ten thousand, thousand, hundred, ten, and one to accommodate calculation results

2) Shift the hexadecimal number by one to the left, and put the removed part into the defined variable, and judge whether the units of ten thousand, thousand, hundred, ten, and one are greater than or equal to 5, and if so, add the corresponding bit to 3 until the 16-bit shift is completed, and the corresponding result is obtained.

Note: Do not add 3 when moving to the last digit, put the operation result directly

3) The Principle of hexadecimal number to BCD number conversion

Suppose ABCD is a 4-digit binary number (possibly ones, 10 or 100 bits, etc.), adjusts it to BCD code. Since the entire calculation is implemented in successive shifts, ABCDE is obtained after shifting one bit (E is from low displacement and its value is either 0 or 1). At this time, it should be judged whether the value is greater than or equal to 10. If so, the value is increased by 6 to adjust it to within 10, and the carry is shifted to the upper 4-bit BCD code. Here, the pre-movement adjustment is used to first determine whether ABCD is greater than or equal to 5 (half of 10), and if it is greater than 5, add 3 (half of 6) and then shift.

For example, ABCD = 0110 (decimal 6)

A. After shifting it becomes 1100 (12), greater than 1001 (decimal 9)

B. By plus 0110 (decimal 6), ABCD = 0010, carry position is 1, the result is expressed as decimal

C. Use pre-shift processing, ABCD = 0110 (6), greater than 5, plus 3

D. ABCD=1001(9), shift left by one

E. ABCD=0010, the shifted shift is the lowest bit of the high four-bit BCD.

F. Since the shifted bit is 1, ABCD = 0010(2), the result is also 12 in decimal

G. The two results are the same

H. Firstly, make a judgement, and then add 3 and shift. If there are multiple BCD codes at the same time, then multiple BCD numbers all must first determine whether need to add 2 and then shift.

(3) The first way is relatively easy. Here, the second method is mainly introduced.

Example 1:

100's	10's	1's	Binary	Operation
			1010 0010	
		1	010 0010	<< #1
		10	10 0010	<< #2
		101	0 0010	<< #3
		1000		add 3
	1	0000	0010	<< #4
	10	0000	010	<< #5
	100	0000	10	<< #6
	1000	0001	0	<< #7
	1011			add 3
1	0110	0010		<< #8

Fig 8. 1 Binary to decimal

Example 2:

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

Fig 8. 2 Hex to BCD

(4) Write a Verilog HDL to convert 16-bit binary to BCD. (You can find reference in the project folder, *HEX_BCD.v*

```
`timescale 10ns/1ns
module HEX_BCD
(
input                                                    [15:0] hex,
output      reg[3:0]   ones=0,
output      reg[3:0]   tens=0,
output      reg[3:0]   hundreds=0,
output      reg[3:0]   thousands=0,
output      reg[3:0]   ten_thousands=0
);

reg [15:0] hex_reg;
integer    i;

always@(*)
begin
        hex_reg          =hex;
            ones               =0;
            tens               =0;
            hundreds           =0;
```

```
                    thousands         =0;
                    ten_thousands=0;

for (i=15;i>=0;i=i-1)begin

if(ten_thousands>=5)
ten_thousands=ten_thousands+3;

if(thousands>=5)
thousands=thousands+3;

if(hundreds>=5)
  hundreds=hundreds+3;

if(tens>=5)
  tens=tens+3;

if(ones>=5)
ones=ones+3;

ten_thousands =ten_thousands<< 1;//Left shift operation
ten_thousands[0]=thousands[3];

thousands =thousands<<1;
thousands[0]=hundreds[3];

hundreds=hundreds<<1;
hundreds[0]=tens[3];

tens=tens<<1;
tens[0]=ones[3];

ones=ones<<1;
ones[0]=hex_reg[15];

hex_reg={hex_reg[14:0],1'b0};
  end
end
endmodule
```

(5) Modelsim simulation

a. Refer to last experiment for setting Modelsim
b. Simulation result shown in Fig 8. 3.

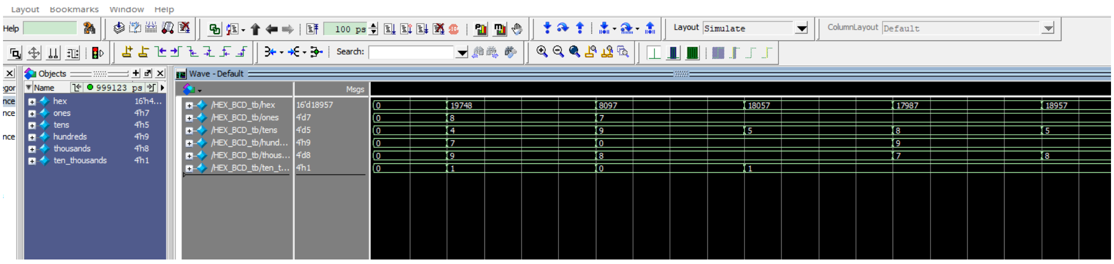


Fig 8. 3 Simulation result for Hex to BCD

(6) Remark

The assignment marks for the examples above are “=” instead of “<=”. Why?

Since the whole program is designed to be combinational logic, when invoking the modules, the other modules should be synchronized the timing.

2.Application of Hexadecimal Number to BCD Number Conversion

(1) Continue to complete the multiplier of experiment 7 and display the result in segment decoders in decimal. The code is as follows:

```
module mult_sim(
                input     rst,
                input     inclk,
                input     [7:0] sw,
                output    reg[6:0] seven_sega,
                output    reg[5:0] scan
);



wire      [15:0] mult_res;
wire      sys_clk;
wire      sys_rst;
wire      us_f;
wire      ms_f;
wire      s_f;

reg           [7:0]       count;
reg           [3:0]       counta;
reg           [6:0]       seven_seg_ra;

wire          [3:0]       ones;
wire          [3:0]       tens;
wire          [3:0]       hundreds;
wire          [3:0]       thousands;
wire          [3:0]       ten_thousands;
```

```
reg          [3:0]     ones_r;
reg          [3:0]     tens_r;
reg          [3:0]     hundreds_r;
reg          [3:0]     thousands_r;
reg          [3:0]     ten_thousands_r;

always@(posedge sys_clk)
if(sys_rst) begin
count        <=0;
ones_r       <=0;
tens_r       <=0;
hundreds_r<=0;
thousands_r<=0;
ten_thousands_r<=0;
end
else if(s_f) begin
count<=count+1;
ones_r       <=ones;
tens_r       <=tens;
hundreds_r<=hundreds;
thousands_r<=thousands;
ten_thousands_r<=ten_thousands;
end
reg    ext_rst;

 always@(posedge sys_clk)
 ext_rst<=rst;

reg [2:0]    scan_st;

 always@(posedge sys_clk)
     if(!ext_rst) begin
      scan<=6'b11_1111;
                 counta<=4'b0;
      scan_st<=0;
     end
   else case(scan_st)
0:begin
      scan<=6'b11_1110;
                 counta<=ones_r;
                 if(ms_f)
      scan_st<=1;

end
```

```
1:begin
        scan<=6'b11_1101;
                counta<=tens_r;
          if(ms_f)
          scan_st<=2;

end
2:begin
        scan<=6'b11_1011;
                counta<=hundreds_r;
          if(ms_f)
          scan_st<=3;

end
3:begin
        scan<=6'b11_0111;
                counta<=thousands_r;
        if(ms_f)
            scan_st<=4;
end
4:begin
        scan<=6'b10_1111;
                counta<=ten_thousands_r;
        if(ms_f)
            scan_st<=5;

end
5:begin
        scan<=6'b01_1111;
                counta<=0;
        if(ms_f)
            scan_st<=0;
end


default:scan_st<=0;
endcase

always@(*)
case(counta)
0:seven_seg_ra<=7'b100_0000;
1:seven_seg_ra<=7'b111_1001;
2:seven_seg_ra<=7'b010_0100;
3:seven_seg_ra<=7'b011_0000;
```

```
4:seven_seg_ra<=7'b001_1001;
5:seven_seg_ra<=7'b001_0010;
6:seven_seg_ra<=7'b000_0010;
7:seven_seg_ra<=7'b111_1000;
8:seven_seg_ra<=7'b000_0000;
9:seven_seg_ra<=7'b001_0000;
default:seven_seg_ra<=7'b100_0000;
endcase

    always@(posedge sys_clk)
    seven_sega<=seven_seg_ra;

lpm_mult8x8 lpm_mult8x8_inst (
    .CLK(inclk),    // input wire CLK
    .A(sw),          // input wire [7 : 0] A
    .B(count),          // input wire [7 : 0] B
    .P(mult_res)          // output wire [15 : 0] P
    );

pll_sys_rst pll_sys_rst_inst
(
.clk_in                  (inclk),
.sys_clk                 (sys_clk),
.sys_rst                 (sys_rst),
.BCD_clk                  (          )
);

us_ms_s_div us_ms_s_div_inst
(
.sys_rst        (sys_rst),
.sys_clk        (sys_clk),
.us_f           (us_f),
.ms_f           (ms_f),
.s_f             (s_f)
);

HEX_BCD HEX_BCD_inst
(
.hex                     (mult_res),
.ones                    (ones),
.tens                    (tens),
.hundreds               (hundreds),
.thousands             (thousands),
.ten_thousands    (ten_thousands)
```

```
);

endmodule
```

(2) After completing the implementation process, click **Open Implementation Design** as shown in Fig 8. 4. Observe the **Report Timing Summary** and view the circuit timing report.

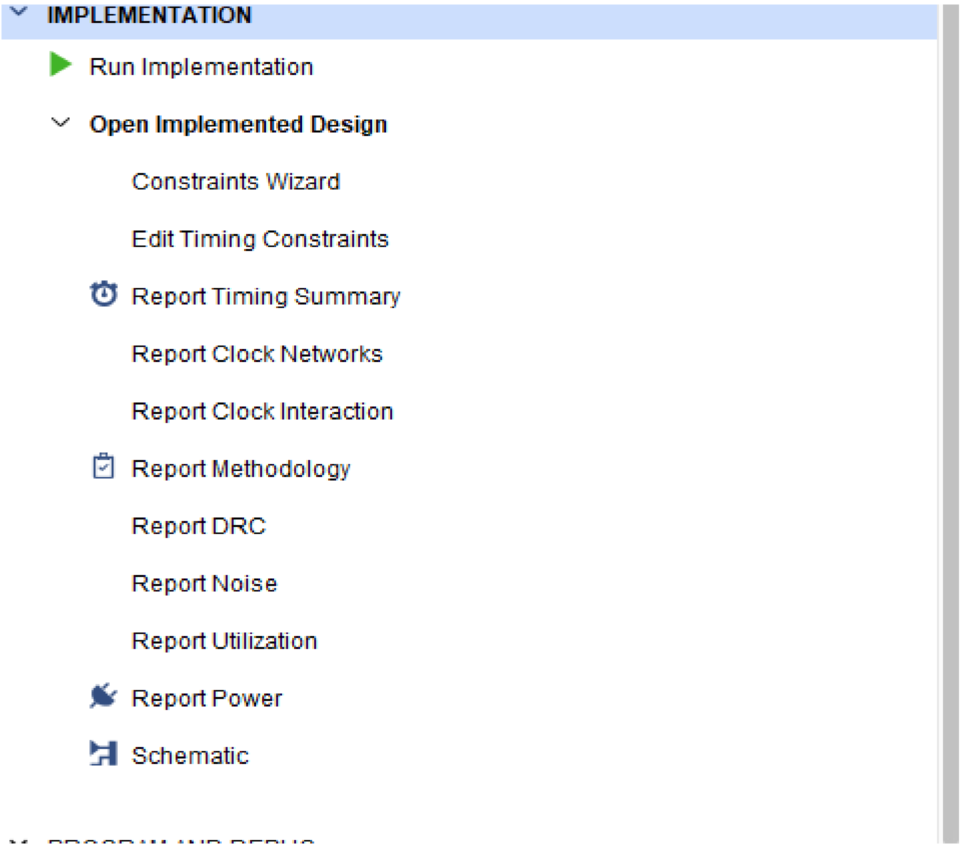


Fig 8. 4 Timing report check

The result is shown in Fig 8. 5.

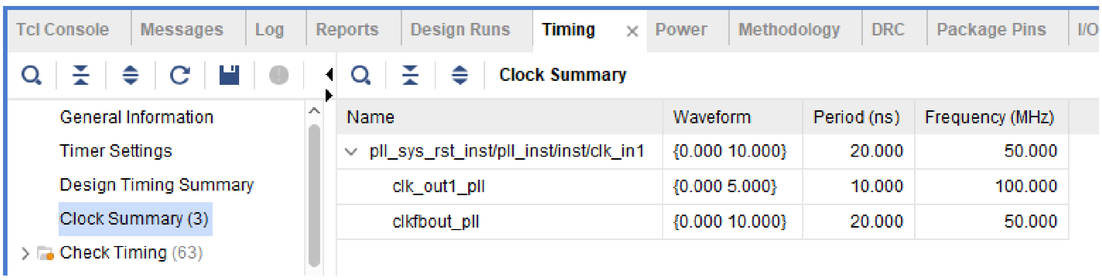


Fig 8. 5 Timing report

It satisfies the timing requirement.

(3) Pin assignment

Signal Name	Port Description	Network Label	FPGA Pin
inclk	System clock, 50 MHz	C10_50MCLK	U22
rst	Reset, high by default	KEY1	M4

seven_sega[0]	Segment a	SEG_PA	K26
seven_sega[1]	Segment b	SEG_PB	M20
seven_sega[2]	Segment c	SEG_PC	L20
seven_sega[3]	Segment d	SEG_PD	N21
seven_sega[4]	Segment e	SEG_PE	N22
seven_sega[5]	Segment f	SEG_PF	P21
seven_sega[6]	Segment g	SEG_PG	P23
seven_sega[7]	Segment h	SEG_DP	P24
scan[0]	Segment 6	SEG_3V3_D5	T24
scan[1]	Segment 5	SEG_3V3_D4	R25
scan[2]	Segment 4	SEG_3V3_D3	K25
scan[3]	Segment 3	SEG_3V3_D2	N18
scan[4]	Segment 2	SEG_3V3_D1	R17
scan[5]	Segment 1	SEG_3V3_D0	R16
sw[0]	Swicth input	GPIO_DIP_SW0	N8
sw[1]	Swicth input	GPIO_DIP_SW1	M5
sw[2]	Swicth input	GPIO_DIP_SW2	P4
sw[3]	Swicth input	GPIO_DIP_SW3	N4
sw[4]	Swicth input	GPIO_DIP_SW4	U6
sw[5]	Swicth input	GPIO_DIP_SW5	U5
sw[6]	Swicth input	GPIO_DIP_SW6	R8
sw[7]	Swicth input	GPIO_DIP_SW7	P8

(4) Compile, and download the program to the board. The test result is shown below:

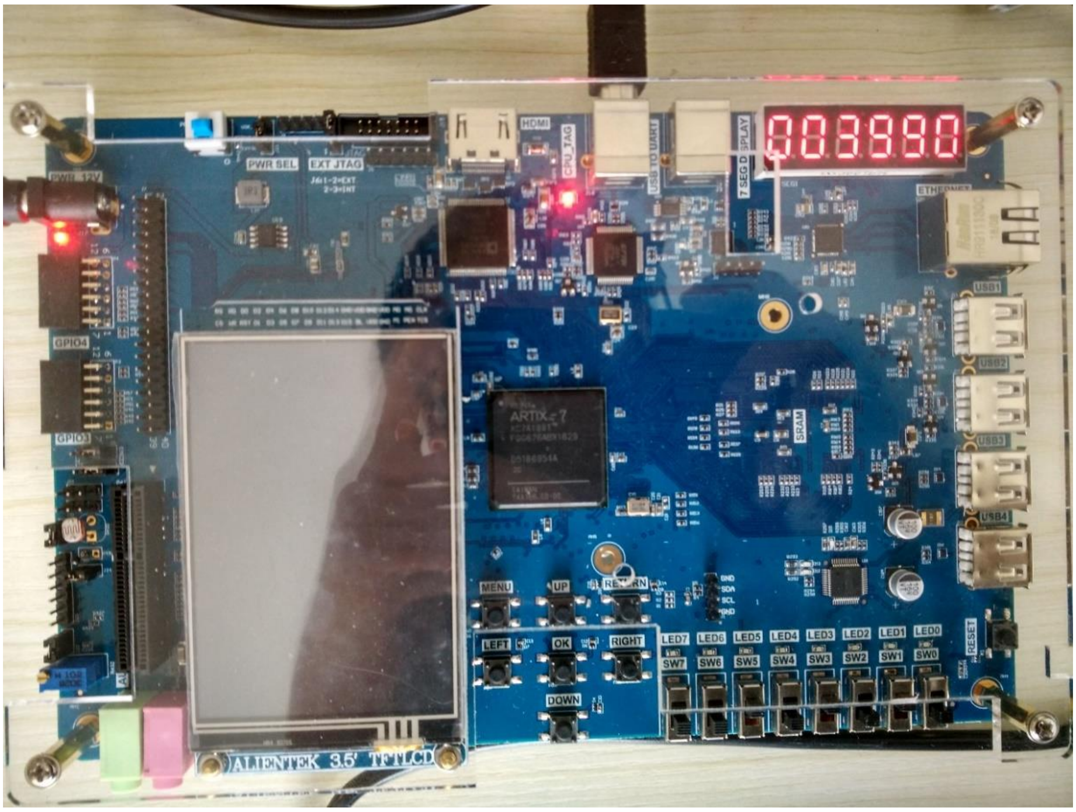


Fig 8. 6 Hex to BCD result

3.Experiment Reflection

(1) How to implement BCD using more than 16 bits binary numbers
(2) How to handle an asynchronous clock
(3) Learn how to design circuits that meet timing requirements based on actual needs

Experiment 9 Use of ROM

1.Experiment Objective

(1) Study the internal memory block of FPGA

(2) Study the format of *.mif and how to edit *.mif file to configure the contents of ROM

(3) Learn to use RAM, read and write RAM

2.Experiment Design

(1) Design 16 outputs ROM, address ranging 0-255

(2) Interface 8-bit switch input as ROM's address

(3) Segment decoders display the contents of ROM and require conversion of hexadecimal to BCD output.

3.Design Procedure

(1) Create a coe file. This experiment *.coe file is generated based on Matlab2018. The *.m file is as follows:

```
% --by Fraser Innovation Inc--
% function : create .coe
clear all;
close all;
clc;
depth= 256;
width =16;
fid_s = fopen('test_rom.coe', 'w+');
fprintf(fid_s, 'MEMORY_INITIALIZATION_RADIX = %d;\n',width);
fprintf(fid_s, '%s\n', 'MEMORY_INITIALIZATION_VECTOR =');

for i=0:depth-1
    data =i*i;
    b=dec2hex(data);
    fprintf(fid_s, '%s', b);
    fprintf(fid_s, '%s\n', ',');
end
fclose(fid_s);
disp('=======mif file completed=======');
```

(2) *.coe file syntax is shown in Fig 9. 1.

```
MEMORY_INITIALIZATION_RADIX = 16;
MEMORY_INITIALIZATION_VECTOR =
0,
1,
4,
9,
10,
19,
24,
31,
40,
51,
64,
79,
90,
A9,
C4,
E1,
100,
121,
144,
```

Fig 9. 1 *.coe file syntax

(3) Create new project, **rom_test**, select device **XC7A100TFGG676-2**

(4) Click **IP Catalog**, and input **ROM** in the search box. Choose **Block Memory Generator**. See Fig 9. 2.

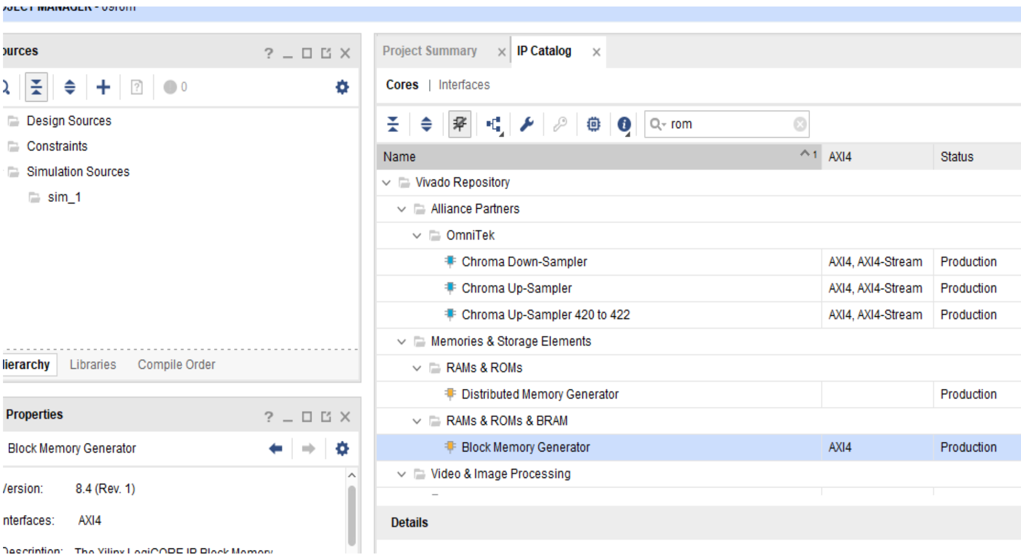


Fig 9. 2 Use of ROM IP core

(5) Select the memory type be **Single Port ROM**. See Fig 9. 3.

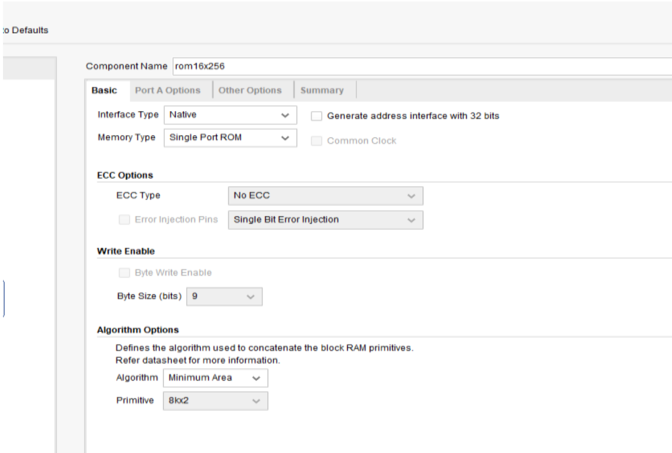


Fig 9. 3 Memory type selection

(6) Click **Port A Options** tag. Set as shown in Fig 9. 4.

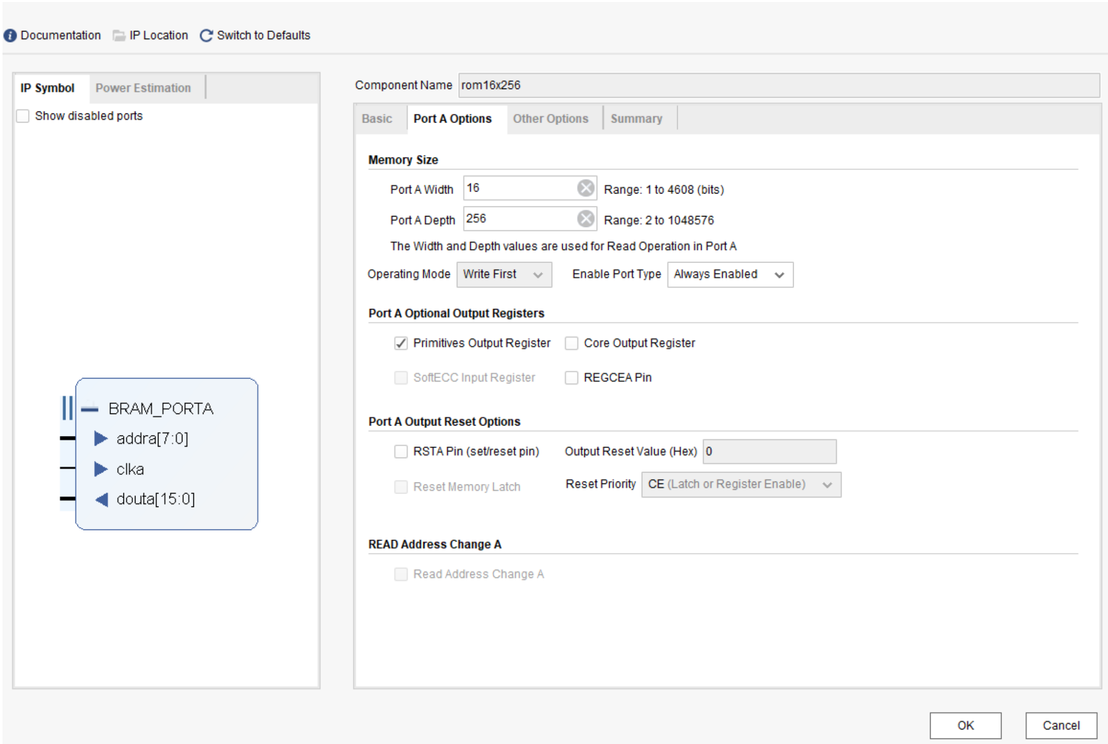


Fig 9. 4 Port memory width setting

(7) Click the **Other Options** tab shown in Fig 9. 5, select the **Load Init File** check box, set the correct *.coe file location, and initialize the rom.

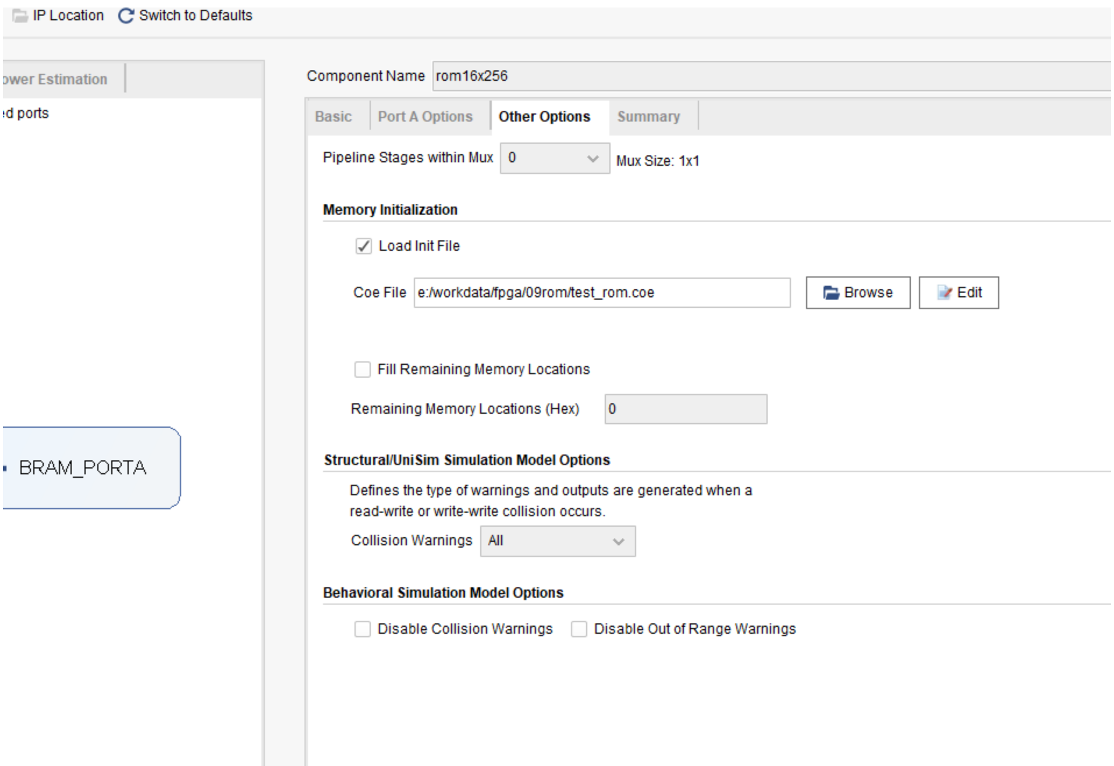


Fig 9. 5 ROM initialization

(8) Set others as default

(9) Click **OK** to finish setting for IP core. Generate other files related as default setting.

(10) Create top-level entity, *rom_test.v*

(11) Add PLL (Input clock 50 MHz, output clock 100 MHz)

(12) Add *us_ms_s_div.v* and instantiate it. Refer previous experiments for more

(13) Add HEX_BCD and instantiate it

(14) The code is given below:

```
module rom_test(
        input       rst,
        input       inclk,
        input       [7:0] sw,
        output      reg[5:0] scan,
        output      reg[6:0] seven_sega

);

        wire [15:0] rom_q;
        wire sys_clk;
        wire BD_clk;
        wire sys_rst;
        wire u_f;
        wire m_f;
        wire sf;
```

```
reg  [7:0] count;
reg  [5:0] counta;
reg  [6:0] seven_seg_ra;

wire [3:0]   ones;
wire [3:0]   tens;
wire [3:0]   hundreds;
wire [3:0]   thousands;
wire [3:0]   ten_thousands;

reg  [3:0]   ones_r;
reg  [3:0]   tens_r;
reg  [3:0]   hundreds_r;
reg  [3:0]   thousands_r;
reg  [3:0]   ten_thousands_r;

reg  [3:0]   ones_x;
reg  [3:0]   tens_x;
reg  [3:0]   hundreds_x;
reg  [3:0]   thousands_x;
reg  [3:0]   ten_thousands_x;

always@(posedge BCD_clk)
    begin
        ones_x          <=   ones;
        tens_x          <=   tens;
        hundreds_x      <=   hundreds;
        thousands_x     <=   thousands;
        ten_thousands_x <= ten_thousands;
    end

always@(posedge sys_clk)
    if(sys_rst)
        begin
            count           <=   0;
            ones_r          <=   0;
            tens_r          <=   0;
            hundreds_r      <=   0;
            thousands_r     <=   0;
            ten_thousands_r <= 0;
        end
    else if(s_f)
            begin
```

```
                count            <=  count+1;
                ones_r           <=  ones_x;
                tens_r           <=  tens_x;
                hundreds_r       <=  hundreds_x;
                thousands_r      <=  thousands_x;
                ten_thousands_r  <=  ten_thousands_x;
            end

reg ext_rst;

always@(posedge sys_clk)
    ext_rst<=rst;

reg [2:0] scan_st;

always@(posedge sys_clk)
    if(!ext_rst)
        begin
            scan<=6'b11_1111;
            counta<=4'b0;

            scan_st<=0;
        end

    else case(scan_st)

        0   :   begin
                    scan <= 6'b11_1110;
                    counta    <= ones_r;
                    if( ms_f )
                    scan_st <= 1;
                end

        1   :   begin
                    scan      <=6'b11_1101;
                    counta    <=tens_r;
                    if ( ms_f )
                    scan_st<=2;
                end

        2   :   begin
                    scan <=6'b11_1011;
                    counta    <=hundreds_r;
                    if(ms_f)
```

```
                    scan_st<=3;
                end

            3   :   begin
                    scan <=6'b11_0111;
                    counta    <=thousands_r;
                    if(ms_f)
                    scan_st<=4;
                end

            4   :   begin
                    scan <=6'b10_1111;
                    counta    <=ten_thousands_r;
                    if(ms_f)
                    scan_st<=5;
                end

            5   :   begin
                    scan      <=6'b01_1111;
                    counta    <=0;
                    if (ms_f)
                    scan_st<=0;
                end

            default:scan_st<=0;
        endcase
    always@(*)
        case(counta)
            0 : seven_sega <= 7'b100_0000 ;
            1 : seven_sega <= 7'b111_1001 ;
            2 : seven_sega <= 7'b010_0100 ;
            3 : seven_sega <= 7'b011_0000 ;
            4 : seven_sega <= 7'b001_1001 ;
            5 : seven_sega <= 7'b001_0010 ;
            6 : seven_sega <= 7'b000_0010 ;
            7 : seven_sega <= 7'b111_1000 ;
            8 : seven_sega <= 7'b000_0000 ;
            9 : seven_sega <= 7'b001_0000 ;
            default: seven_sega<=7'b100_0000 ;
        endcase

pll_sys_rst pll_sys_rst_inst(
        .clk_in(inclk),
        .sys_clk(sys_clk),
```

```
        .BCD_clk(BCD_clk),
        .sys_rst(sys_rst)
        );

    us_ms_s_div us_ms_s_div_inst
    (
        .sys_rst(sys_rst),
        .sys_clk(sys_clk),
        .us_f(us_f),
        .ms_f       (ms_f),
        .s_f  (s_f)
    );

    reg  [15:0] rom_q_r;

    always@(posedge BCD_clk)
        rom_q_r<=rom_q;

    HEX_BCD HEX_BCD_inst(
        .hex                (rom_q_r),
        .ones               (ones),
        .tens               (tens),
        .hundreds           (hundreds),
        .thousands          (thousands),
        .ten_thousands      (ten_thousands)
    );

    rom16x256 rom16x256_inst (
        .clka(sys_clk),      // input wire clka
        .addra(sw),    // input wire [7 : 0] addra
        .douta(rom_q)    // output wire [15 : 0] douta
    );

endmodule
```

a. Compile
b. Lock the pins

Signal Name	Port Description	Network Label	FPGA Pin
inclk	Sytem clock, 50 MHz	C10_50MCLK	U22
rst	Reset, hight by default	KEY1	M4
seven_sega[0]	Segment a	SEG_PA	K26
seven_sega[1]	Segment b	SEG_PB	M20
seven_sega[2]	Segment c	SEG_PC	L20

seven_sega[3]	Segment d	SEG_PD	N21
seven_sega[4]	Segment e	SEG_PE	N22
seven_sega[5]	Segment f	SEG_PF	P21
seven_sega[6]	Segment g	SEG_PG	P23
seven_sega[7]	Segment h	SEG_DP	P24
scan[0]	Segment 6	SEG_3V3_D5	T24
scan[1]	Segment 5	SEG_3V3_D4	R25
scan[2]	Segment 4	SEG_3V3_D3	K25
scan[3]	Segment 3	SEG_3V3_D2	N18
scan[4]	Segment 2	SEG_3V3_D1	R17
scan[5]	Segment 1	SEG_3V3_D0	R1 6
sw[0]	Switch input	GPIO_DIP_SW0	N8
sw[1]	Switch input	GPIO_DIP_SW1	M5
sw[2]	Switch input	GPIO_DIP_SW2	P4
sw[3]	Switch input	GPIO_DIP_SW3	N4
sw[4]	Switch input	GPIO_DIP_SW4	U6
sw[5]	Switch input	GPIO_DIP_SW5	U5
sw[6]	Switch input	GPIO_DIP_SW6	R8
sw[7]	Switch input	GPIO_DIP_SW7	P8

c. Download the program and test the result

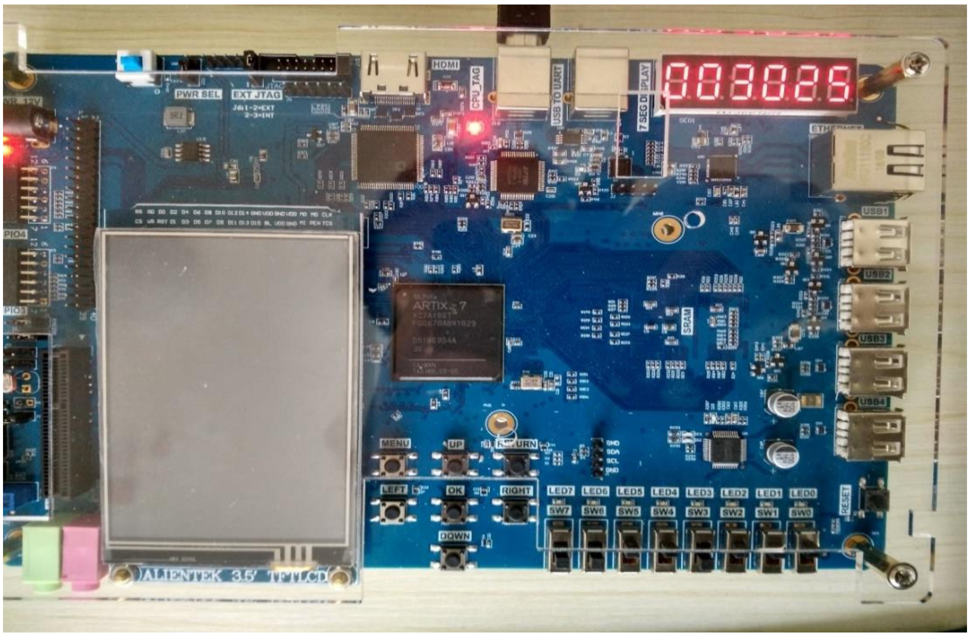


Fig 9. 6 Test result

(15) Experiment summary and reflection

a. How to use the initial file of ROM to realize the decoding, such as decoding and scanning the segment decoders.

b. Write a *.mif file to generate sine, cosine wave, and other function generators.

c. Comprehend application, combine the characteristic of ROM and PWM to form

SPWM modulation waveform.

Experiment 10 Use Dual_port RAM to Read and Write Frame Data

1.Experiment Objective

(1) Learn to configure and use dual-port RAM
(2) Learn to use synchronous clock to control the synchronization of frame structure
(3) Learn to use asynchronous clock to control the synchronization of frame structure
(4) Observing the synchronization structure of synchronous clock frames using ILA
(5) Extended the use of dual-port RAM
(6) Design the use of three-stage state machine

2.Experiment Implement

(1) Generate dual-port RAM and PLL
  a. 16-bit width, 256-depth dual-port RAM
  b. 2 PLL, both 50 MHz input, different 100 MHz and 20 MHz outputs

(2) Design a 16-bit data frame
  a. Data is generated by an 8-bit counter: Data={~counta,counta}
  b. The ID of the data frame inputted by the switch (7 bits express maximum of 128 different data frames)
  c. 16-bit *checksum* provides data verification
    1) 16-bit *checksum* accumulates, discarding the carry bit
    3) After the *checksum* is complemented, append to the frame data
  d. Provide configurable data length *data_len* by *parameter*
  e. Packet: When the data and *checksum* package are written to the dual-port RAM, the userID, the frame length and the valid flag are written to the specific location of the dual-port RAM. The structure of the memory is shown below

Wr_addr	Date/ Flag	Rd_addr
8’hff	{valid, ID, data_len}	8’hff
...	N/A	...
8’hnn+2	N/A	8’hnn+2
8’hnn+1	~checksum+1	8’hnn+1
8’hnn	datann	8’hnn
...	....	...
8’h01	Data1	8’h01
8’h00	Data0	8’h00

  f. Read and write in an agreed order

Firstly, write in the order

1) Read the flag of the 8'hff address (control word). If *valid*=1'b0, the program proceeds to the next step, otherwise waits
2) Address plus 1, 8’hff+1 is exactly zero, write data from 0 address and calculate the *checksum*
3) Determine whether the interpretation reaches the predetermined data length. If so, proceeds to next step, otherwise the data is written, and the checksum is calculated.
4) *checksum* complements and write to memory
5) Write the control word in the address 8'hff, packet it

Secondly, read in the order

1) *Idle* is the state after reset
2) *Init*: Initialization, set the address to 8’hff
3) *Rd_pipe0*: Add a latency (since the read address and data are both latched). Address +1, forming a pipeline structure
4) *Read0*: Set the address to 8’hff, read the control word and judge whether the valid bit is valid.
   If *valid*=1’b1, address +1, proceeds to the next step
   If *valid*=1’b0, it means the packet is not ready yet, the address is set to be 8’hff and returns to the *init* state.
5) *Read1*: Read the control word again
   If *valid*=1’b1, address+1, ID and data length are assigned to the corresponding variables and proceeds to the next step
   If *valid*=1’b0, it means the packet is not ready yet, the address is set to 8’hff, and returns to the *init* state.
6) Rd_data:
   Read data and pass to data variables
   Calculate *checksum*, *data_len* – 1
   Determine whether the *data_len* is 0, if so, all data has been read, proceeds to the next step, otherwise, continue the operation in current state
7) *grd_chsum*: Read the value of *checksum* and calculate the last *checksum*. Correct the data and set the flag of *rd_err*
8) *rd_done*: The last step clears the valid flag in memory and opens the write enable for the next packet.

Thirdly, *valid* is the handshake signal. This flag provides the possibility of read and write synchronization, so the accuracy of this signal must be ensured in the program design. See the project files for more details.

3.Program Design

(1) Port

```
module frame_ram
#(parameter data_len=250)
(
input                       inclk,
```

```
input                   rst,        //external reset
input           [6:0]  sw,          //used as input ID
output   reg[6:0] oID,              //used as output ID
output   reg            rd_done,    //frame read is done
output   reg     rd_err      //frame read has errors


);
```

(2) Definition of state machine

```
parameter    [2:0] mema_idle=0,
                    mema_init=1,
                    mema_pipe0=2,
                    mema_read0=3,
                    mema_read1=4,
                        mema_wr_data=5,
                        mema_wr_chsum=6,
                        mema_wr_done=7;

parameter    [2:0]      memb_idle=0,
                        memb_init=1,
                        memb_pipe0=2,
                        memb_read0=3,
                        memb_read1=4,
                        memb_rd_data=5,
                        memb_rd_chsum=6,
                        memb_rd_done=7;
```

(3) Define clock parameter

```
wire            sys_clk;
wire            BCD_clk;
wire            sys_rst;
reg             ext_clk;
```

(4) Define two-port RAM interface

```
reg   [7:0]     addr_a;
reg   [15:0]    data_a;
reg                 wren_a;
wire [15:0]     q_a;

reg   [7:0]     addr_b;
reg                 wren_b;
wire [15:0]     q_b;
```

(5) Write state machine partial variable definition

a. Write stste machine variables

```
reg[6:0]     user_id;
```

```
reg[7:0]      wr_len;

reg[15:0]     wr_chsum;
Wire          wr_done;

reg[7:0]      counta;
Wire[7:0]     countb=~counta;

Reg           ext_rst;
Reg [2:0]     sta;
reg[2:0]      sta_nxt;
```

b. Read state machine variables

```
reg[15:0]     rd_chsum;
reg[7:0]      rd_len;
reg[15:0]     rd_data;

Reg           ext_rst;
reg[2:0]      stb;
reg[2:0]      stb_nxt;
```

(6) Data generation counter

```
always@(posedge BCD_clk)
ext_rst<=rst;

always@(posedge sys_clk)
if(sys_rst) begin
counta     <=0;
user_id    <=0;
end
else begin
counta <=counta+1;
user_id<=sw;
End
```

(7) Write state machine

a. First and second stages

```
assign wr_done=(wr_len==data_len-1);//Think why using wr_len==data_len-1
                                    //instead of wr_len==data_len
always@(posedge sys_clk)
if(sys_rst) begin
    sta=mema_idle;
end
else
    sta=sta_nxt;

always@(*)
```

```
case (sta)
mema_idle    : sta_nxt=mema_init;

mema_init    : sta_nxt=mema_pipe0;

mema_pipe0 : sta_nxt=mema_read0;

mema_read0 :begin
    if(!q_a[15])
    sta_nxt=mema_read1;
    else
    sta_nxt=sta;
end
mema_read1:begin
    if(!q_a[15])
    sta_nxt=mema_wr_data;
    else
    sta_nxt=sta;
end
mema_wr_data: begin
    if(wr_done)
    sta_nxt=mema_wr_chsum;
    else
    sta_nxt=sta;
end
mema_wr_chsum:    sta_nxt=mema_wr_done;
mema_wr_done: sta_nxt=mema_init;
default:sta_nxt=mema_idle;
endcase
```

b. Third stage

```
always@(posedge sys_clk)
case (sta)
mema_idle: begin
addr_a<=8'hff;
wren_a<=1'b0;
data_a<=16'b0;
wr_len<=8'b0;
wr_chsum<=0;
end
mema_init,mema_pipe0,mema_read0,mema_read1: begin
addr_a<=8'hff;
wren_a<=1'b0;
data_a<=16'b0;
wr_len<=8'b0;
```

```
wr_chsum<=0;
end
mema_wr_data:begin
addr_a<=addr_a+1;
wren_a<=1'b1;
data_a<={countb,counta};
wr_len<=wr_len+1;

wr_chsum<=wr_chsum+{countb,counta};
end


mema_wr_chsum:begin
addr_a<=addr_a+1;
wr_len<=wr_len+1;
wren_a<=1'b1;
data_a<=(~wr_chsum)+1'b1;
end

mema_wr_done:begin
addr_a<=8'hff;
wren_a<=1'b1;
data_a<={1'b1,user_id,wr_len};
end
default:;
endcase
```

(8) Read state machine

a. First stage

```
always@(posedge sys_clk)
if(!ext_rst) begin
    stb=memb_idle;
end
else
     stb=stb_nxt;
```

b. Second stage

```
always@(*)
case (stb)
memb_idle    : stb_nxt=memb_init;

memb_init    : stb_nxt=memb_pipe0;

memb_pipe0 : stb_nxt=memb_read0;

memb_read0 :begin
```

```
    if(q_b[15])
    stb_nxt=memb_read1;
    else
    stb_nxt=memb_init;
end
memb_read1:begin
    if(q_b[15])
    stb_nxt=memb_rd_data;
    else
    stb_nxt=memb_init;
end
memb_rd_data: begin
    if(rd_done)
    stb_nxt=memb_rd_chsum;
    else
    stb_nxt=stb;
end
memb_rd_chsum:    stb_nxt=memb_rd_done;
memb_rd_done: stb_nxt=memb_init;
default:stb_nxt=memb_idle;
endcase
```

c. Third stage. The actual operation is driven by the edge of the clock

```
always@(posedge sys_clk)
case(stb)
memb_idle: begin
addr_b<=8'hff;
rd_data<=0;
rd_chsum<=0;
wren_b<=1'b0;
rd_len<=8'b0;
oID<=7'b0;
rd_err<=1'b0;
end

memb_init: begin
addr_b<=8'hff;
rd_data<=0;
rd_chsum<=0;
wren_b<=1'b0;
rd_len<=8'b0;
oID<=7'b0;
rd_err<=1'b0;
endmemb_pipe0: begin
addr_b<=8'b0;
```

```
end

memb_read0: begin
if(q_b[15])
addr_b<=addr_b+1'b1;
else
addr_b<=8'hff;

rd_data<=0;
rd_chsum<=0;
wren_b<=1'b0;
rd_len<=8'b0;
oID<=7'b0;
end
memb_read1: begin
if(q_b[15])
addr_b<=addr_b+1'b1;
else
addr_b<=8'hff;

rd_data<=0;
rd_chsum<=0;
wren_b<=1'b0;
rd_len<=q_b[7:0];
oID<=q_b[14:8];
end

memb_rd_data: begin
addr_b<=addr_b+1'b1;
rd_data<=q_b;
rd_chsum<=rd_chsum+rd_data;
wren_b<=1'b0;
rd_len<=rd_len-1'b1;
end

memb_rd_chsum: begin
    addr_b<=8'hff;
    wren_b<=1'b0;

    if(!rd_chsum)//Determine if rd_chsum is not 0, else error occurs when reading data
    rd_err<=1'b1;
end

memb_rd_done: begin
```

```
      addr_b<=8'hff;
      wren_b<=1'b1;
      end
      default:;
      endcase

      always@(*)begin
      if(stb==memb_rd_data)
      rd_done=(rd_len==0);
      else
      rd_done=1'b0;
      end
```

(9) Instantiate dual_port RAM and PLL

```
//Instantiate dual-port RAM
dp_ram dp_ram_inst
(
.address_a(addr_a),
.address_b(addr_b),
.clock     (sys_clk),
.data_a    (data_a),
.data_b    (16'b0),
.wren_a(wren_a),
.wren_b(wren_b),
.q_a       (q_a),
.q_b       (q_b)
);

//Instantiate PLL
pll_sys_rst pll_sys_rst_inst
(
.inclk     (inclk),
.sys_clk (sys_clk),
.BCD_clk(BCD_clk),
.sys_rst (sys_rst)

);

endmodule
```

4.Lock the Pins, Compile, and Download to The Board to Test

(1) Pin assignment

Signal Name	Port Description	Network Label	FPGA Pin
inclk	System clock, 50 MHz	C10_50MCLK	U22

rst	Reset, high by default	KEY1	M4
oID_r[0]	LED 0	LED0	N17
oID_r[1]	LED 1	LED1	M19
oID _r[2]	LED 2	LED2	P16
oID _r[3]	LED 3	LED3	N16
oID _r[4]	LED 4	LED4	N19
oID_r [5]	LED 5	LED5	P19
oID_r [6]	LED 6	LED6	N24
sw[0]	Switch input	GPIO_DIP_SW0	N8
sw[1]	Switch input	GPIO_DIP_SW1	M5
sw[2]	Switch input	GPIO_DIP_SW2	P4
sw[3]	Switch input	GPIO_DIP_SW3	N4
sw[4]	Switch input	GPIO_DIP_SW4	U6
sw[5]	Switch input	GPIO_DIP_SW5	U5
sw[6]	Switch input	GPIO_DIP_SW6	R8
rd_err_r	Read error flag	SEG_PA	P24
rd_done_r	Dual-port end reading	SEG_PB	K26
weixuan	Segment 1	SEG_3V3_D0	R16

(2) Download the program to the develop board

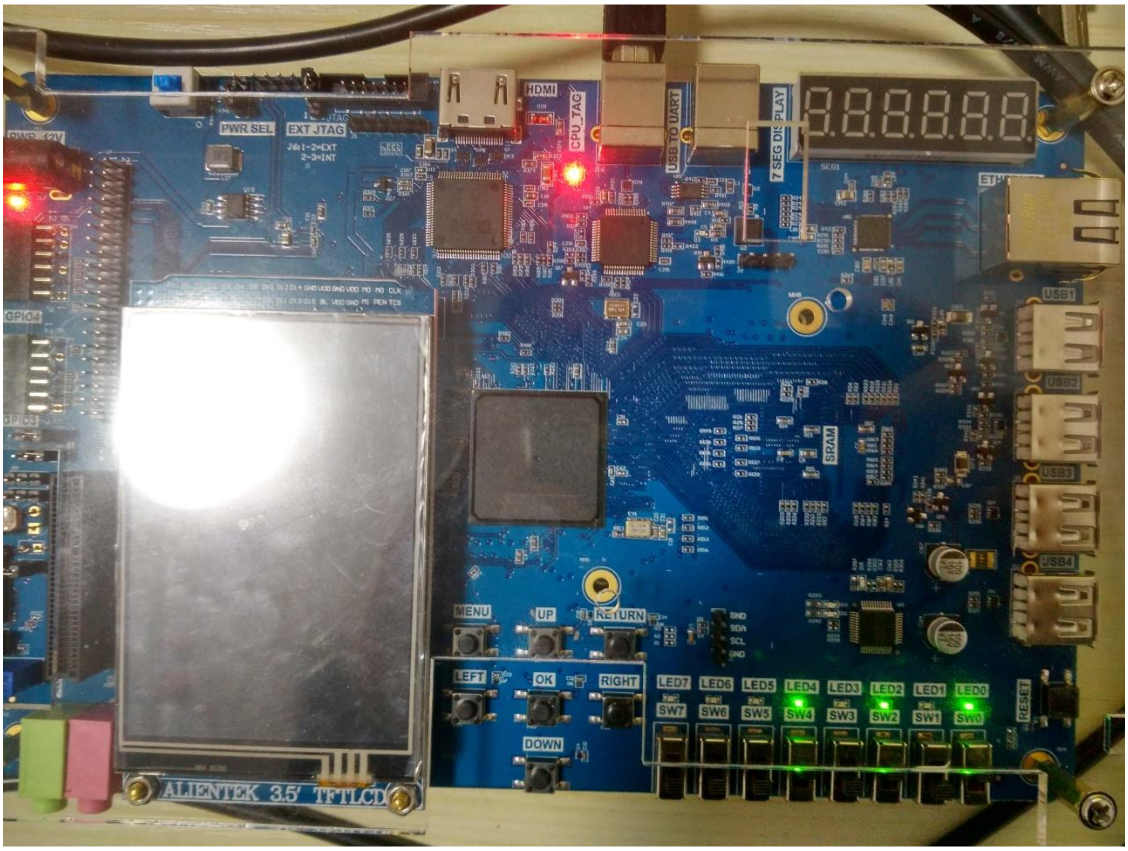


Fig 10. 1 Dual_port RAM test result

From the test results, SW6~SW0 (write ID) and read ID (LEDs) are completely consistent. And no error reading occurred during the reading and writing process. The results can be derived from the ILA plot.

5.Use ILA to Observe Dual_port RAM Read and Write

(1) To facilitate the observation of the read and write state machine synergy results, the data length is changed to 4 here, recompile and download. Users can test themselves using long data.

```
module frame_ram
#(parameter data_len=4)
(
input    inclk,
input    rst,            //external reset
input    [6:0]sw,            //used as input ID
output  reg[6:0] oID,          //used as output ID
output  reg    rd_done,   //frame read is done
output  reg     rd_err //frame read has errors

);
```

(2) ILA test result. See Fig 10. 2

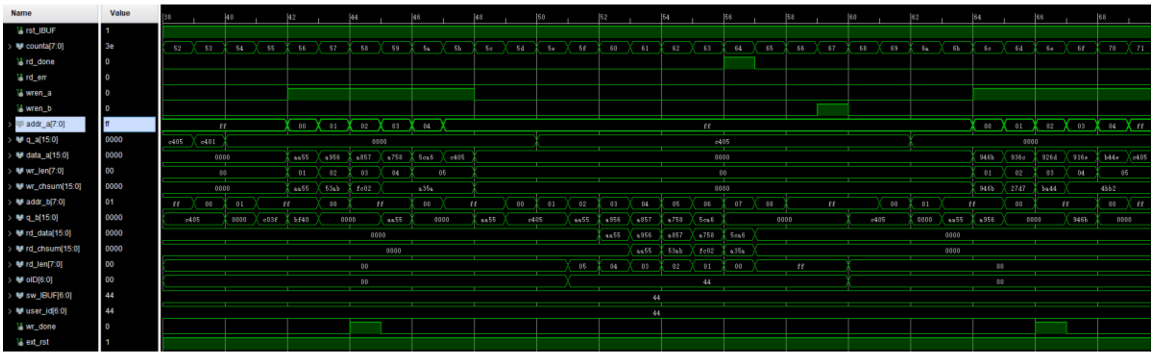


Fig 10. 2 Signals observed form ILA

(3) Observe the test result

a. Observe the handshake mechanism through dual-port RAM

Determine whether the reading is started after the packet is written, whether the write packet is blocked before reading the entire packet is completed.

b. Observe the external interface signal and status

Rd_done, *rd_err*

Set *rd_err* = 1, or the rising edge is the trigger signal to observe whether the error signal is captured.

Observe whether *wren_a*, *wren_b* signal and the state machine jump are strictly matched to meet the design implements.

6.Experiment Summary and Reflection

(1) Review the design implements. How to analyze an actual demand, gradually establish a model of digital control and state machine and finally design.
(2) Modify the third stage of the state machine into the if...else model and implement.
(3) Focus on thinking If the read and write clocks are different, it becomes an asynchronous mechanism, how to control the handshake.
(4) According to the above example, consider how dual-port RAM can be used in data acquisition, asynchronous communication, embedded CPU interface, and DSP chip interface.
(5) How to build ITCM with dual-port RAM and DTCM preparing for future CPU design.

Experiment 11 Asynchronous Serial Port Design and Experiment

1.Experiment Objective

(1)Because asynchronous serial ports are very common in industrial control, communication, and software debugging, they are also vital in FPGA development.
(2)Learning the basic principles of asynchronous serial port communication, handshake mechanism, data frame
(3)Master asynchronous sampling techniques
(4)Review the frame structure of the data packet
(5)Learning FIFO
(6)Joint debugging with common debugging software of PC (SSCOM, teraterm, etc.)

2.Experiment Implement

(1) Design and transmit full-duplex asynchronous communication interface Tx, Rx
(2) Baud rate of 11520 bps, 8-bit data, 1 start bit, 1 or 2 stop bits
(3) Receive buffer (Rx FIFO), transmit buffer (Tx FIFO)
(4) Forming a data packet
(5) Packet parsing

3.Experiment Design

(1) Build new project named *uart_frame*, select **XC7A100TFGG676-2** for device.
(2) Add new file named *uart_top*, add a PLL (can be copied from the previous experiment)

```
module uart_top
(
input       inclk,
input       rst,
input       baud_sel,
input       rx,
output      intx
);

wire   sys_clk;
wire   uart_clk;
wire   sys_rst;
wire   uart_rst;

pll_sys_rst pll_sys_rst_inst
(
.inclk      (inclk),
```

```
.sys_clk    (sys_clk),
.uart_clk  (uart_clk),
.sys_rst (sys_rst),
.uart_rst(uart_rst)

);

endmodule
```

(3) New baud rate generator file

a. Input clock 7.3728MHz (64 times 115200). The actual value is 7.377049MHz, which is because the coefficient of the PLL is an integer division, while the error caused by that is not large, and can be adjusted by the stop bit in asynchronous communication. See Fig 11. 1.

Fine solution

1) Implemented with a two-stage PLL for a finer frequency
2) The stop bit is set to be 2 bits, which can effectively eliminate the error.

This experiment will not deal with the precision. The default input frequency is 7.3728 MHz.

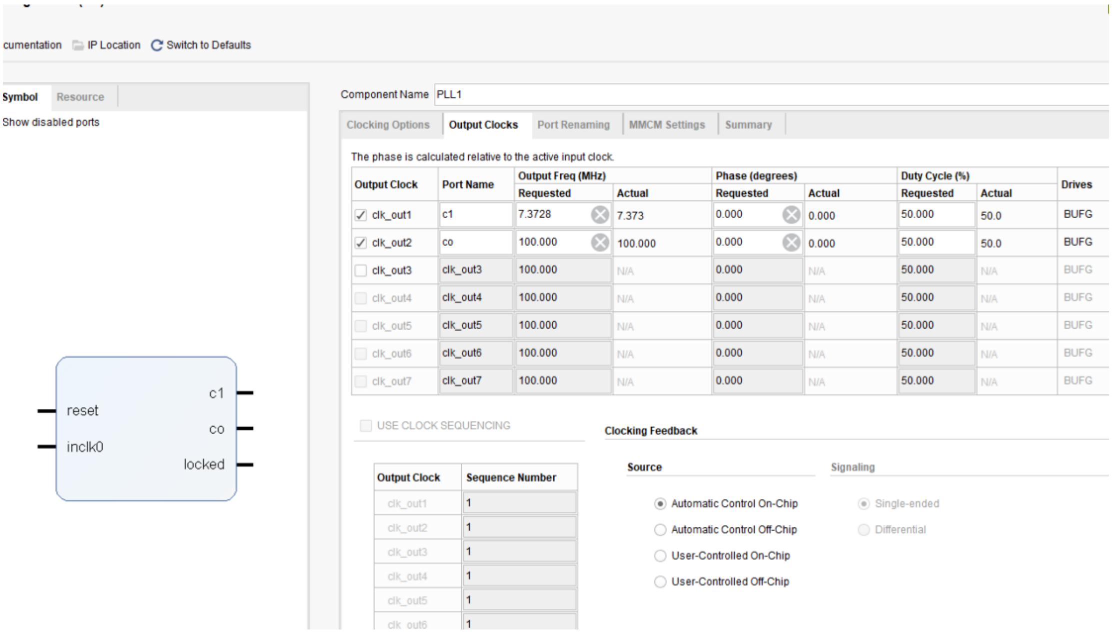


Fig 11. 1 PLL setting

b. Supported baud rates are 115200，57600，38400，19200

c. The default baud rate is 115200

(4) Design of baud rate

a. Instantiate and set it top-level entity

```
wire              tx_band;
wire             tx_band;
baud_rate
```

```
#(.div(64))
  baud_rate_inst
(
.rst            (uart_rst),
.inclk          (uart_clk),
.baud_sel       (baud_sel),
.baud_tx        (baud_tx),
);
```

b. Baud rate design source file

```
`timescale 1ns / 10ps
module baud_rate
#(parameter div=64)
(

input                   rst,
input                   inclk,
input       [1:0] baud_sel,
output reg          baud_tx,
output reg        baud_rx


);

//Send baud rate, clock frequency division selection
wire [8:0] frq_div_tx;
assign frq_div_tx=(baud_sel==2'b0)?9'd63:
                               (baud_sel==2'b01)?9'd127:
                               (baud_sel==2'b10)?9'd255:9'd511;


reg   [8:0] count_tx=9'd0;
always@(posedge inclk)
if(rst) begin
   count_tx    <=9'd0;
   baud_tx <=1'b0;
end
else begin
       if(count_tx==frq_div_tx) begin
          count_tx <=9'd0;
          baud_tx<=1'b1;
       end
       else begin
       count_tx<=count_tx+1'b1;
```

```
        baud_tx<=1'b0;
        end

end

//Accept partial baud rate design
wire [6:0] frq_div_rx;
assign frq_div_rx=(baud_sel==2'b0)?7'd7:
                   (baud_sel==2'b01)?7'd15:
                   (baud_sel==2'b10)?7'd31:7'd63;



reg   [8:0] count_rx=9'd0;

always@(posedge inclk)
if(rst) begin
    count_rx    <=9'd0;
    baud_rx <=1'b0;
end
else begin
        if(count_rx==frq_div_rx) begin
            count_rx <=9'd0;
            baud_rx<=1'b1;
        end
        else begin
        count_rx<=count_rx+1'b1;
        baud_rx<=1'b0;
        end

end
endmodule
```

(5) Design the buffer file *tx_buf*

a. 8-bit FIFO, depth is 256, read/write clock separation, full flag, read empty flag
b. Interface and handshake
   1) *rst* reset signal
   2) *wr_clk* write clock
   3) *tx_clk* send clock
   4) 8-bit write data *tx_data*
   5) *wr_en* write enable
   6) *ctrl* writes whether the data is a data or a control word
   7) *rdy* buffer ready, can accept the next data frame
c. Send buffer instantiation file

tx_buf

```
#(.TX_BIT_LEN(8),.STOP_BIT(2))
tx_buf_inst
(
.sys_rst        (sys_rst),
.uart_rst    (uart_rst),
.wr_clk         (sys_clk),
.tx_clk         (uart_clk),
.tx_baud      (tx_baud),
.tx_wren      (tx_wren),
.tx_ctrl        (tx_ctrl),
.tx_datain (tx_data),
.tx_done        (tx_done),
.txbuf_rdy (txbuf_rdy),
.tx_out        (tx_out)

);
```

d. Send buffer source file

```
`timescale 1ns / 10ps
module tx_buf
#(
parameter TX_BIT_LEN=8,
parameter STOP_BIT=1
)
(
input                    sys_rst,
input                    uart_rst,
input                    wr_clk,
input                    tx_clk,
input                    tx_baud,
input                    tx_wren,
input                    tx_ctrl,
input                    tx_done,
input            [7:0] tx_datain,
output     reg       txbuf_rdy,
output                 tx_out

);

parameter      [2:0] TXWR_IDLE=0,
                              TXWR_RST=1,
                                      TXWR_INIT=2,
                                      TXWR_WAIT=3,
                                      TXWR_WR    =4,
                                      TXWR_DONE=5;
```

```
parameter      [2:0] TXRD_IDLE     =0,
                     TXRD_INIT     =1,
                     TXRD_WAIT0 =2,
                     TXRD_WAIT1 =3,
                     TXRD_SEND0 =4,
                     TXRD_SEND1 =5,
                     TXRD_DONE     =6;

reg                 wr_clr=1'b1;
reg                  wr_en;
reg   [8:0] wr_data;
reg    [5:0]delay;
wire            rst_done=(delay==0);


wire                trans_rdy;//from low level transmit module

wire                 wr_full;

reg                  rd_ack;
wire    [8:0] txbuf_q;
reg           tx_en;
reg   [7:0] tx_len;

wire                rd_empty;
reg   [7:0] tx_data;


reg   [2:0] wr_st,wr_st_nxt;

always@(posedge wr_clk)
if(sys_rst)
wr_st<=TXWR_IDLE;
else
wr_st<=wr_st_nxt;

always@(*) begin
        case(wr_st)
                TXWR_IDLE: wr_st_nxt=TXWR_RST;
                TXWR_RST: begin
                    if(rst_done)
                    wr_st_nxt=TXWR_INIT;
                    else
```

```
            wr_st_nxt=wr_st;
          end
          TXWR_INIT: wr_st_nxt=TXWR_WAIT;
          TXWR_WAIT:begin
           if(!wr_full)
           wr_st_nxt=TXWR_WR;
           else
           wr_st_nxt=wr_st;
          end
          TXWR_WR: begin
               if(tx_done)
                 wr_st_nxt=TXWR_DONE;
                 else if(wr_full)
                 wr_st_nxt=TXWR_WAIT;
                 else
                 wr_st_nxt=wr_st;
          end
          TXWR_DONE: begin
                 wr_st_nxt=TXWR_INIT;
          end
     endcase
end

always@(posedge wr_clk) begin

if(wr_st==TXWR_IDLE) begin
     wr_clr <=1'b1;
     wr_en    <=1'b0;
     wr_data<=9'b0;
     txbuf_rdy <=1'b0;
     delay          <=31;

end
if(wr_st==TXWR_RST) begin
     delay<=delay-1'b1;
end
if(wr_st==TXWR_INIT) begin
     wr_clr <=1'b0;
     wr_en    <=1'b0;
     wr_data<=9'b0;
     txbuf_rdy <=1'b0;
end

if(wr_st==TXWR_WAIT) begin
```

```
        wr_clr <=1'b0;
        wr_en    <=1'b0;
        wr_data<=9'b0;
        txbuf_rdy <=1'b0;
end

if(wr_st==TXWR_WR) begin
        if(tx_done)
        txbuf_rdy <=1'b0;
        else
        txbuf_rdy <=1'b1;

        if(tx_wren) begin
        wr_en    <=1'b1;
        wr_data<={tx_ctrl,tx_datain};
        end

end

if(wr_st==TXWR_DONE) begin
        wr_en    <=1'b0;
        wr_data<=9'b0;
        txbuf_rdy <=1'b0;
end

end


reg   [2:0] rd_st,rd_st_nxt;

always@(posedge tx_clk)
if(uart_rst)
rd_st<=TXRD_IDLE;
else
rd_st<=rd_st_nxt;


always@(*)
case(rd_st)
TXRD_IDLE:rd_st_nxt=TXRD_INIT;
TXRD_INIT:begin
if(!rd_empty)
rd_st_nxt=TXRD_WAIT0;
end
```

```
TXRD_WAIT0:begin
if(txbuf_q[8])
rd_st_nxt=TXRD_WAIT1;
else if(rd_empty)
rd_st_nxt=TXRD_INIT;
else
rd_st_nxt=rd_st;

end

TXRD_WAIT1:begin
if(trans_rdy)
rd_st_nxt=TXRD_SEND0;
else
rd_st_nxt=rd_st;
end
TXRD_SEND0:begin

rd_st_nxt=TXRD_SEND1;
end
TXRD_SEND1:begin
if(tx_len==0)
rd_st_nxt=TXRD_DONE;
else if(!rd_empty)
rd_st_nxt=TXRD_WAIT1;
else
rd_st_nxt=rd_st;
end
TXRD_DONE:rd_st_nxt=TXRD_INIT;
endcase

always@(posedge tx_clk) begin
case(rd_st)
TXRD_IDLE: begin
rd_ack     <=1'b0;
tx_en           <=1'b0;
tx_len     <=8'b0;
tx_data    <=8'b0;
end
TXRD_INIT: begin
rd_ack     <=1'b0;
tx_en           <=1'b0;
tx_len     <=8'b0;
```

```
tx_data    <=8'b0;
end
TXRD_WAIT0: begin
rd_ack     <=1'b1;
tx_en              <=1'b0;
tx_len     <=txbuf_q[7:0];
tx_data    <=txbuf_q[7:0];
end
TXRD_WAIT1: begin
rd_ack     <=1'b0;
        if(trans_rdy) begin

                tx_en              <=1'b1;
                tx_len      <=tx_len  -1;
        end
        else begin
        tx_en              <=1'b0;
        end

end

TXRD_SEND0: begin
rd_ack     <=1'b0;

tx_en              <=1'b0;
end
TXRD_SEND1: begin

tx_data    <=txbuf_q[7:0];
if(trans_rdy)begin
        rd_ack     <=1'b1;
        tx_en      <=1'b1;
  end
  else    begin
      rd_ack      <=1'b0;
      tx_en <=1'b0;
  end
end

TXRD_DONE: begin
rd_ack     <=1'b0;
tx_en              <=1'b0;
end
default:;
```

```
endcase
end
```

(6) Serial transmission, interface and handshake file design

a. Interface design
   1) *tx_rdy*, send vacancy, can accept new 8-bit data
   2) *tx_en*, send data enable, pass to the sending module 8-bit data enable signal
   3) *tx_data*, 8-bit data to be sent
   4) *tx_clk*, send clock
   5) *tx_baud*, send baud rate

b. Instantiation

```
tx_transmit
#(.DATA_LEN(TX_BIT_LEN),
   .STOP_BIT(STOP_BIT)
)
tx_transmit_inst
(
.tx_rst     (uart_rst),
.tx_clk     (tx_clk),
.tx_baud  (tx_baud),
.tx_en      (tx_en),
.tx_data   (tx_data),
.tx_rdy     (trans_rdy),
.tx_out     (tx_out)

);
```

c. Source file

```
`timescale 1ns / 10ps
module tx_transmit
#(parameter DATA_LEN=8,
   parameter STOP_BIT=1
)
(
input                       tx_rst,
input                       tx_clk,
input                       tx_baud,
input                       tx_en,
input            [7:0] tx_data,
output reg              tx_rdy,
output reg           tx_out


);

parameter    [2:0] TX_IDLE=0,
```

```
                    TX_INIT=1,
                    TX_WAIT=2,
                    TX_SEND_START=3,
                    TX_SEND_DATA=4,
                    TX_SEND_STOP1=5,
                    TX_SEND_STOP2=6,
                    TX_DONE=7;


reg  [1:0]  stop_bit=STOP_BIT;
reg  [3:0]  tx_len;
reg  [8:0]   tx_data_r;
reg  [2:0]   tx_st,tx_st_nxt;

//wire[2:0]      tx_len=(stop_bit==0)?7:    //8bit
//                       (stop_bit==1)?6:   //7bit
//                       (stop_bit==2)?5:4; //6bit:5bit


always@(posedge tx_clk)
if(tx_rst)
    tx_st<=TX_IDLE;
else
    tx_st<=tx_st_nxt;


always@(*)
case(tx_st)
TX_IDLE: tx_st_nxt=TX_INIT;

TX_INIT: tx_st_nxt=TX_WAIT;
TX_WAIT: begin
    if(tx_en)
    tx_st_nxt=TX_SEND_START;
    end
TX_SEND_START: begin
   if(tx_baud)
    tx_st_nxt=TX_SEND_DATA;
end

TX_SEND_DATA: begin
    if((tx_len==0)&tx_baud)
    tx_st_nxt=TX_SEND_STOP1;
end
```

```
TX_SEND_STOP1: begin
      if(tx_baud) begin
                if(stop_bit==2'b01)
                    tx_st_nxt=TX_DONE;
                  else
                tx_st_nxt=TX_SEND_STOP2;
        end
end
TX_SEND_STOP2:    begin

      if(tx_baud)
        tx_st_nxt=TX_DONE;
        else
        tx_st_nxt=tx_st;
end
TX_DONE:begin
        tx_st_nxt=TX_IDLE;
end
default:tx_st_nxt=TX_IDLE;
endcase

always@(posedge tx_clk) begin
case(tx_st)
TX_IDLE:begin
        tx_rdy    <=1'b0;
        tx_data_r <='b0;
        tx_len    <=3'd0;
        tx_out    <=1'b1;
end
TX_INIT:begin
        tx_rdy    <=1'b1;
        tx_data_r <=8'b0;
        tx_len    <=4'd8;
        tx_out    <=1'b1;
end

TX_WAIT:begin
        tx_rdy    <=1'b1;
        tx_len    <=4'd8;
        tx_data_r <=tx_data;
        tx_out    <=1'b1;
end
TX_SEND_START:begin
        tx_rdy    <=1'b0;
```

```
            if(tx_baud)
            tx_out    <=1'b0;
        end

        TX_SEND_DATA:begin
            if(tx_baud) begin
                tx_len     <=tx_len-1'b1;
                tx_out     <=tx_data_r[0];
                tx_data_r<={1'b0,tx_data_r[7:1]};
            end
        end
        TX_SEND_STOP1:begin
                tx_len     <=0;
                if(tx_baud) begin
                tx_out     <=1'b1;
                end
        end
        TX_SEND_STOP2:begin
                if(tx_baud) begin
                tx_out     <=1'b1;
                end
            end
        TX_DONE:begin
                tx_rdy     <=1'b0;
                tx_out     <=1'b1;
            end
        default:;
        endcase
        end
        endmodule
```

(7)Send *testbench.v*

```
    `timescale 1ns / 10ps
    module tb_uart(

        );

      reg       inclk;
      parameter PERIOD = 20;

          initial begin
             inclk = 1'b0;
             //#(PERIOD/2);

          end
```

```
  always
                    #(PERIOD/2) inclk = ~inclk;

reg              rst=0;
wire [1:0]    baud_sel=2'b00;

reg              tx_wren=0;
reg              tx_ctrl=0;
reg      [7:0] tx_data=0;
reg      [7:0]    tx_len=0;
reg              tx_done;
wire             txbuf_rdy;

wire             sys_clk;
wire             sys_rst;
reg              rx_in=0;

wire            tx_out;

initial begin
rst=1'b1;
#100 rst=1'b0;
end

//transmit test
reg    [7:0] count=0;


reg    [3:0] trans_st;
always@(posedge sys_clk)
if(sys_rst)begin
       trans_st       <=0;
       tx_wren         <=1'b0;
       tx_ctrl         <=1'b0;
       tx_data         <=8'b0;
       tx_done         <=1'b0;
       tx_len          <=0;
       tx_len          <=0;
       count           <=8'd0;
end
else case(trans_st)
0:begin
       trans_st       <=1;
       tx_wren         <=1'b0;
```

```
        tx_ctrl         <=1'b0;
        tx_data         <=8'b0;
        tx_done         <=1'b0;
        tx_len          <=16;
        end
1:begin
        tx_wren         <=1'b0;
        tx_ctrl         <=1'b0;
        tx_data         <=8'b0;
        tx_done         <=1'b0;
        if(txbuf_rdy)
        trans_st        <=2;
        end
  2:begin
      tx_wren         <=1'b1;
      tx_ctrl         <=1'b1;
      tx_data         <=tx_len;
      trans_st        <=3;
        end
  3:begin
      tx_wren         <=1'b0;
      tx_ctrl         <=1'b0;
      if(tx_len==0)
            trans_st        <=4;
      else if(txbuf_rdy) begin
              tx_data         <=count;
              count           <=count+1;
              tx_wren         <=1'b1;
              tx_len          <=tx_len-1;
      end
      end
4:begin
        tx_wren         <=1'b0;
        tx_ctrl         <=1'b0;
        tx_data         <=0;
        tx_len          <=16;
        tx_done         <=1'b1;
        trans_st        <=5;
      end
5:begin
            tx_done         <=1'b0;
            trans_st        <=1;
          end
endcase
```

```
uart_top uart_top_dut
(
.inclk          (inclk),
.rst            (rst),
.baud_sel       (baud_sel),
.tx_wren        (tx_wren),
.tx_ctrl        (tx_ctrl),
.tx_data        (tx_data),
.tx_done        (tx_done),
.txbuf_rdy      (txbuf_rdy),
.sys_clk        (sys_clk),
.sys_rst        (sys_rst),
.rx_in          (rx_in),
.tx_out         (tx_out)

);


endmodule
```

(8)Send Modelsim simulation. See Fig 11. 2.

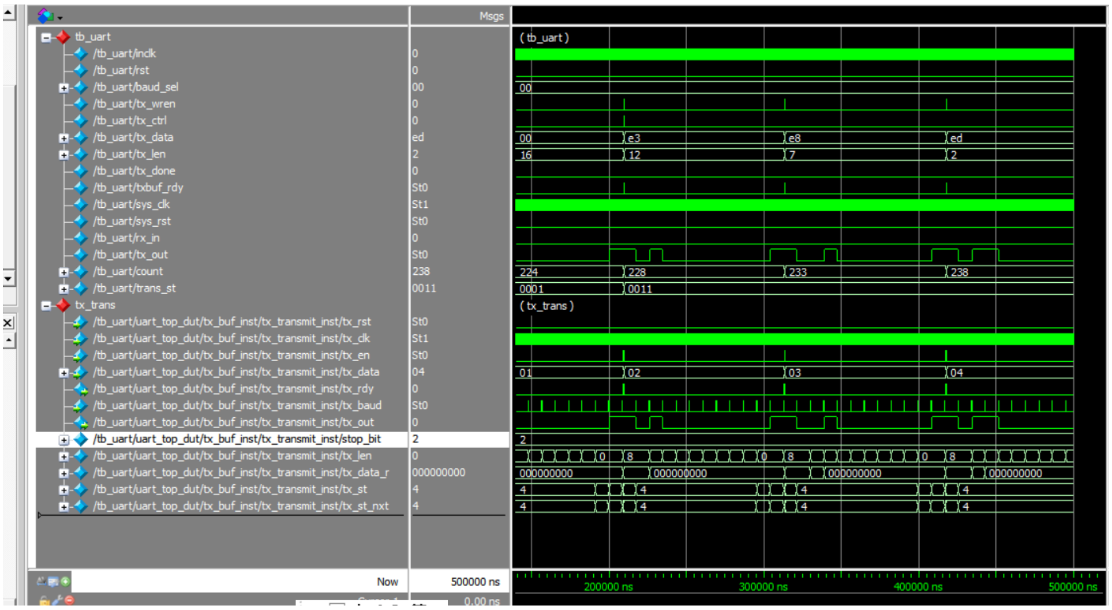


Fig 11. 2 ModelSim simulation waves sent by serial

(9)Extended design (extended content is only reserved for users to think and practice)

a. Design the transmitter to support 5, 6, 7, 8-bit PHY (Port physical layer)
b. Support parity check

(10)The settings of the above steps involve FIFO, PLL, etc. (Refer to *uart_top* project file)

UART accept file design

a. Design of *rx_phy.v*

Design strategies and steps

1) Use 8 times sampling: so *rx_baud* is different from *tx_baud*, here sampling is *rx_band* = 8**tx_band*
2) Adopting multiple judgments to realize the judgment of receiving data. Determine whether the data counter is greater than 4 after the sampling value is counted.
3) Steps to receive data:
   A. Synchronization: refers to how to find the start bit from the received 0101... *sync_dtc*
   B. Receive start bit (start)
   C. Cyclically receive 8-bit data
   D. Receive stop bit (determine whether it is one stop bit or two stop bits)

   Determine if the stop bit is correct

   Correct, jump to step 2)

   Error, jump to step 1), resynchronize

   Do not judge, jump directly 2), this design adopts the scheme of no judgment

b. *rx_phy* source file

```
module rx_phy
#(
parameter DATA_LEN=8,
parameter STOP_BIT=1
  )
(
input                  rst,
input                  rx_clk,
input                  rx_baud,
input                  rx_in,
output reg [7:0]rx_byte,
output reg           rx_rdy
);

localparam [3:0]    RX_IDLE=0,
                       RX_INIT=1,
                       RX_SYNC=2,
                       RX_START_DTC=3,
                       RX_START1=4,
                       RX_START2=5,
                       RX_DATA1=6,
                       RX_DATA2=7,
                       RX_STOP1=8,
                       RX_STOP2=9,
                       RX_DONE=10;
```

```
wire [1:0] stop_bit=STOP_BIT;
reg             rx_inr=1'b1;

reg [3:0]    bit_len=4'd0;
reg [6:0]    sync_len=7'b0;
reg [3:0]    sample_len=4'd0;
reg [3:0]    sample_count=4'd0;

wire            bit_value=(sample_count>4);

wire            sync_done=(sync_len==0);
reg             start_det=1'b0;


reg [3:0] rx_st,rx_st_nxt;

always@(rx_clk)
if(rst)
rx_inr<=1'b1;
else
rx_inr<=rx_in;

always@(posedge rx_clk)
if(rst)
rx_st<=RX_IDLE;
else
rx_st<=rx_st_nxt;


always@(*)
case(rx_st)
RX_IDLE: rx_st_nxt=RX_INIT;
RX_INIT: begin
        rx_st_nxt=RX_SYNC;
end
RX_SYNC: begin
        if(sync_done)
        rx_st_nxt=RX_START_DTC;
        else
        rx_st_nxt=rx_st;
end
RX_START_DTC:begin
        if(start_det)
```

```
        rx_st_nxt=RX_START1;
        else
        rx_st_nxt=rx_st;
end

RX_START1:begin
if(sample_len==0)
rx_st_nxt=RX_START2;
else
rx_st_nxt=rx_st;
end
RX_START2:begin
        if(sample_count>4)
        rx_st_nxt=RX_DATA1;
        else
        rx_st_nxt=RX_START_DTC;
end
RX_DATA1:begin
        if(sample_len==0)
        rx_st_nxt=RX_DATA2;
        else
        rx_st_nxt=rx_st;
end

RX_DATA2:begin
      if(bit_len==0)begin
              if(stop_bit==2)
                      rx_st_nxt=RX_STOP1;
              else
              rx_st_nxt=RX_STOP2;
      end
      else
          rx_st_nxt=RX_DATA1;
end
RX_STOP1:begin
        if(rx_baud&(sample_len==0))
              rx_st_nxt = RX_STOP2;
end
RX_STOP2:begin
        if(rx_baud&(sample_len==0))
        rx_st_nxt=RX_DONE;
end
RX_DONE:begin
    rx_st_nxt=RX_START_DTC;
```

```
end
endcase

always@(posedge rx_clk)
case(rx_st)
RX_IDLE: begin
bit_len <=4'd0;
sync_len<=7'd0;
rx_rdy    <=1'b0;
sample_count<=4'd0;
rx_byte <=8'b0;

end

RX_INIT:begin
bit_len          <=4'd8;
sync_len         <=7'd81;
sample_len    <=4'd8;
sample_count<=4'd0;
rx_rdy           <=1'b0;
rx_byte          <=8'b0;
end
RX_SYNC:begin
          if (rx_baud) begin
                if(rx_inr)
                sync_len<=sync_len-1'b1;
                else
                sync_len<=7'd81;
        end
end
RX_START_DTC:begin
    rx_rdy<=1'b0;
    sync_len<=7'd81;
    rx_byte <=8'b0;
    sample_len<=4'd7;
  if (rx_baud) begin
        if(!rx_inr) begin
        start_det<=1'b1;
        sample_count<=4'd1;
        end
  end
end
RX_START1: begin
        start_det<=1'b0;
```

```
  if (rx_baud) begin
        if(!rx_inr) begin
          sample_count<=sample_count+4'd1;
          end
          sample_len<=sample_len-1'b1;
      end
end

RX_START2:begin
sample_count<=0;
sample_len<=4'd8;
end
RX_DATA1:begin

if (rx_baud) begin
                if(rx_inr) begin
                  sample_count<=sample_count+4'd1;
                  end
                  sample_len<=sample_len-1'b1;
  end
end
RX_DATA2:begin
sample_len<=4'd8;
sample_count<=0;
bit_len<=bit_len-4'd1;
rx_byte<={bit_value,rx_byte[7:1]};
end
RX_STOP1:begin
bit_len<=4'd7;
    if (rx_baud) begin
              if(sample_len==0) begin
                  sample_len<=4'd8;
              end
              else
                sample_len<=sample_len-1'b1;
    end

end
RX_STOP2:begin
bit_len<=4'd7;
if (rx_baud) begin
              if(sample_len==0) begin
                  sample_len<=4'd8;
              end
```

```
        else
         sample_len<=sample_len-1'b1;
  end
end

RX_DONE:begin
rx_rdy<=1'b1;
end
endcase

Endmodule
```

c. The design of *rx_buf*

Design strategies and steps

1) Add 256 depth, 8-bit fifo
   A. Read and write clock separation
   B. Asynchronous clear (internal synchronization)
   C. Data appears before the *rdreq* in the read port
2) Steps:
   A. Initialization: *fifo*, *rx_phy*
   B. Wait: FIFO full signal (*wrfull)* is 0
   C. Write: Triggered by *rx_phy*: *rx_phy_byte*, *rx_phy_rdy*
   D. End of writing
   E. Back to ii and continue to wait

Rx_buf.v source code

```
module rx_buf
#(
parameter DATA_LEN=8,
parameter STOP_BIT=1
 )
(
input             sys_clk,
input             rx_clk,
input             sys_rst,
input             uart_rst,
input             rx_in,
input             rx_baud,
input             rx_rden,
output [7:0]      rx_byte,
output    reg     rx_byte_rdy

);


localparam [2:0]   WR_IDLE=0,
```

```
                    WR_RST =1,
                    WR_INIT=2,
                    WR_WAIT=3,
                    WR_WR    =4,
                    WR_DONE=5;


 wire              wr_full;
 wire                    rd_empty;
 wire                    wr_rst_busy;

 reg               wr_clr=0;
 reg               wr_en=0;
 reg       [7:0]   wr_data=0;

 wire      [7:0] rx_phy_byte;
 wire             rx_phy_rdy;
 //wire             rd_rst_busy;

 reg       [2:0] wr_st,wr_st_nxt;


 always@(posedge sys_clk)
 if(sys_rst)
 rx_byte_rdy<=1'b0;
 else
 rx_byte_rdy<=!rd_empty;

 always@(posedge rx_clk)
 if(uart_rst)
 wr_st<= WR_IDLE;
 else
 wr_st<=wr_st_nxt;

 always@(*)
case(wr_st)
WR_IDLE: wr_st_nxt=WR_RST;
WR_RST : begin
    // if(wr_rst_busy)
    // wr_st_nxt=wr_st;
   //    else
      wr_st_nxt=WR_INIT;
end
WR_INIT: begin
```

```
        wr_st_nxt=WR_WAIT;
end
WR_WAIT: begin
        if(!wr_full)
          wr_st_nxt=WR_WR;
end
WR_WR:    begin
        if(rx_phy_rdy)
        wr_st_nxt=WR_DONE;
end
WR_DONE: begin
        wr_st_nxt=WR_WAIT;
end

endcase

always@(posedge rx_clk)
case(wr_st)
WR_IDLE:begin
        wr_clr    <=1'b1;
        wr_en      <=1'b0;
        wr_data <=8'd0;
end
WR_RST: begin
        wr_clr    <=1'b0;
        wr_en      <=1'b0;
        wr_data <=8'd0;
end
WR_INIT: begin
        wr_clr    <=1'b0;
        wr_en      <=1'b0;
        wr_data <=8'd0;
end
WR_WAIT: begin
      wr_clr    <=1'b0;
      wr_en      <=1'b0;
      wr_data <=8'd0;
end

WR_WR:begin
        wr_en      <=rx_phy_rdy;
        wr_data <=rx_phy_byte;
end
WR_DONE: begin
```

```
wr_en      <=1'b0;
wr_data <=8'd0;
end
endcase

  rx_phy
#(
        .DATA_LEN(8),
        .STOP_BIT(1)
  )
  rx_phy_inst
  (
.rst                (uart_rst),
.rx_clk          (rx_clk),
.rx_baud        (rx_baud),
.rx_in            (rx_in),
.rx_byte        (rx_phy_byte),
.rx_rdy          (rx_phy_rdy)
  );

  rx_fifo    rx_fifo_inst
  (
  .aclr                   (wr_clr),
  .data                   (wr_data),
  .rdclk                 (sys_clk),
  .rdreq                 (rx_rden),
  .wrclk                 (rx_clk),
  .wrreq                 (wr_en),
  .q                       (rx_byte),
  .rdempty             (rd_empty),
  .wrfull               (wr_full)
  //.wr_rst_busy      (wr_rst_busy),
// .rd_rst_busy      (rd_rst_busy)
  );
Endmodule
```

(3) Receive simulation incentive

Content and steps

a. *tx*, *rx* loopback test (assign *rx_in = tx_out*)
b. Continue to use the incentive file in the TX section
c. Writing the incentive part of *rx*

Some parts of *tb_uart.v*

```
assign    rx_in=tx_out;
wire        [7:0] rx_byte;
wire                rx_byte_rdy;
```

```
reg [7:0]       rx_byte_r;
reg             rx_rden;

always@(posedge sys_clk)
  if(rx_byte_rdy)begin
                rx_rden <=1'b1;
                rx_byte_r<=rx_byte;
        end
else begin
rx_rden<=1'b0;
end
  uart_top uart_top_dut
(
.inclk          (inclk),
.rst            (rst),
.baud_sel       (baud_sel),
.tx_wren        (tx_wren),
.tx_ctrl        (tx_ctrl),
.tx_data        (tx_data),
.tx_done        (tx_done),
.txbuf_rdy      (txbuf_rdy),
.rx_rden        (rx_rden),
.rx_byte        (rx_byte),
.rx_byte_rdy(rx_byte_rdy),
.sys_clk        (sys_clk),
.sys_rst        (sys_rst),
.rx_in          (rx_in),
.tx_out         (tx_out)
);
```

(4) ModelSim simulation. See Fig 11. 3.

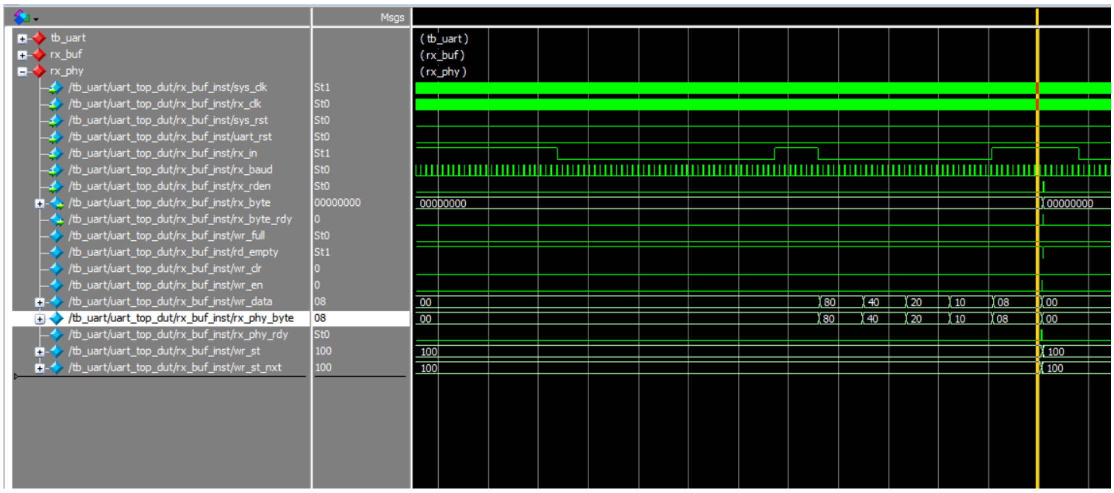


Fig 11. 3 Simulation

Reflection and expansion

a. Modify the program to complete the 5, 6, 7, 8-bit design
b. Completing the design of the resynchronization when the start and stop have errors of the receiving end *rx_phy*
c. Complete the analysis and packaging of the receipt frame of *rx_buf*
d. Using multi-sampling to design 180° alignment of data, compare with FPGA resources, timing and data recovery effects

Hardware test

a. Use develop board to test

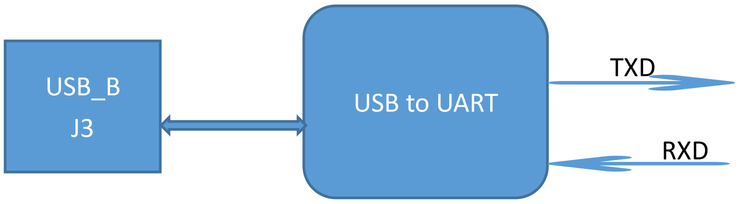


Fig 11. 4 USB to serial conversion

Write a hardware test file.

a. Development board J3 is connected to the host USB interface
   1) Using test software such as teraterm, SSCOM3, etc. You can also write a serial communication program (C#, C++, JAVA, Python...).
   2) PC sends data in a certain format
   3) The test end uses a counter to generate data in a certain format.

The test procedure is as follows module hw_tb_uart

```
(

input                   inclk,
input                   rst,
input       [1:0] baud_sel,
input           rx_in,

output          tx_out

);



reg             tx_wren=0;
reg             tx_ctrl=0;
reg     [7:0] tx_data=0;
reg     [7:0] tx_len=0;
reg             tx_done;
wire            txbuf_rdy;

wire            sys_clk;
```

```
wire            sys_rst;

//transmit test
reg    [7:0] count=0;

reg    [3:0] trans_st;
always@(posedge sys_clk)
if(sys_rst)begin
       trans_st       <=0;
       tx_wren         <=1'b0;
       tx_ctrl        <=1'b0;
       tx_data        <=8'b0;
       tx_done         <=1'b0;
       tx_len          <=0;
       tx_len          <=0;
       count           <=8'd0;
end
else case(trans_st)
0:begin
       trans_st       <=1;
       tx_wren         <=1'b0;
       tx_ctrl        <=1'b0;
       tx_data        <=8'b0;
       tx_done         <=1'b0;
       tx_len          <=16;
       end
1:begin
       tx_wren         <=1'b0;
       tx_ctrl        <=1'b0;
       tx_data        <=8'b0;
       tx_done         <=1'b0;
       if(txbuf_rdy)
       trans_st       <=2;
       end
  2:begin
     tx_wren         <=1'b1;
     tx_ctrl        <=1'b1;
     tx_data         <=tx_len;
     trans_st       <=3;
       end
  3:begin
     tx_wren         <=1'b0;
     tx_ctrl        <=1'b0;
     if(tx_len==0)
```

```
            trans_st        <=4;
        else if(txbuf_rdy) begin
            tx_data         <=count;
            count           <=count+1;
            tx_wren         <=1'b1;
            tx_len          <=tx_len-1;
        end
        end
4:begin
        tx_wren         <=1'b0;
        tx_ctrl         <=1'b0;
        tx_data         <=0;
        tx_len          <=16;
        tx_done         <=1'b1;
        trans_st        <=5;
    end
5:begin
            tx_done         <=1'b0;
            trans_st        <=1;
        end
endcase

wire        [7:0] rx_byte;
wire                rx_byte_rdy;
reg [7:0]       rx_byte_r;
reg                 rx_rden;
always@(posedge sys_clk)
  if(rx_byte_rdy)begin
            rx_rden <=1'b1;
            rx_byte_r<=rx_byte;
    end
else begin
rx_rden<=1'b0;
end
  uart_top uart_top_dut
(
.inclk          (inclk),
.rst            (rst),
.baud_sel       (baud_sel),
.tx_wren        (tx_wren),
.tx_ctrl        (tx_ctrl),
.tx_data        (tx_data),
.tx_done        (tx_done),
.txbuf_rdy      (txbuf_rdy),
```

```
.rx_rden       (rx_rden),
.rx_byte       (rx_byte),
.rx_byte_rdy(rx_byte_rdy),
.sys_clk       (sys_clk),
.sys_rst       (sys_rst),
.rx_in         (rx_in),
.tx_out        (tx_out)
);
endmodule
```

(5) Lock the pins, and test

Signal Name	Port Description	Network Label	FPGA Pin
clk	System clock, 50 MHz	C10_50MCLK	U22
rst_n	Reset, high by default	KEY1	M4
tx_data[0]	Switch input	GPIO_DIP_SW0	N8
tx_data[1]	Switch input	GPIO_DIP_SW1	M5
tx_data[2]	Switch input	GPIO_DIP_SW2	P4
tx_data[3]	Switch input	GPIO_DIP_SW3	N4
tx_data[4]	Switch input	GPIO_DIP_SW4	U6
tx_data[5]	Switch input	GPIO_DIP_SW5	U5
tx_data[6]	Switch input	GPIO_DIP_SW6	R8
tx_data[7]	Switch input	GPIO_DIP_SW7	P8
tx_out	Serial output	TTL_RX	L18
rx_in	Serial input	TTL_TX	L17
txbuf_rdy	Segment a	SEG_PA	P24
rx_byte_rdy	Segment h	SEG_DP	K26
weixuan	Segment 1	SEG_3V3_D0	R16
rx_byte[0]	LED 0	LED0	N17
rx_byte[1]	LED 1	LED1	M19
rx_byte[2]	LED 2	LED2	P16
rx_byte[3]	LED 3	LED3	N16
rx_byte[4]	LED 4	LED4	N19
rx_byte[5]	LED 5	LED5	P19
rx_byte[6]	LED 6	LED6	N24
rx_byte[7]	LED 7	LED7	N23
tx_wren	Write control	KEY2	L4
tx_ctrl	Write data control	KEY3	L5
tx_done	Write ending control	KEY4	K5
rx_rden	Read enable	KEY5	R1

(6) Observe the data received

(7) Using ILA to observe the data sent by FPGA

(8) See Fig 11. 5, when FPGA sends A0

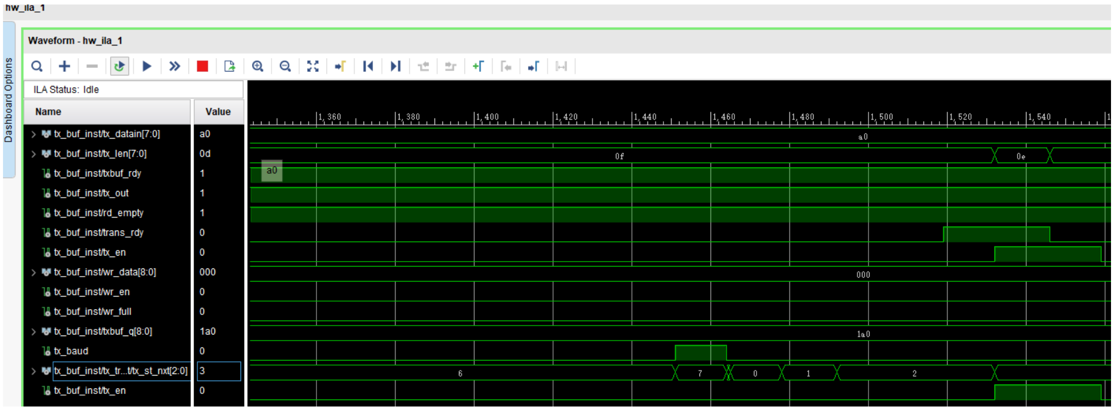


Fig 11. 5 Sending A0

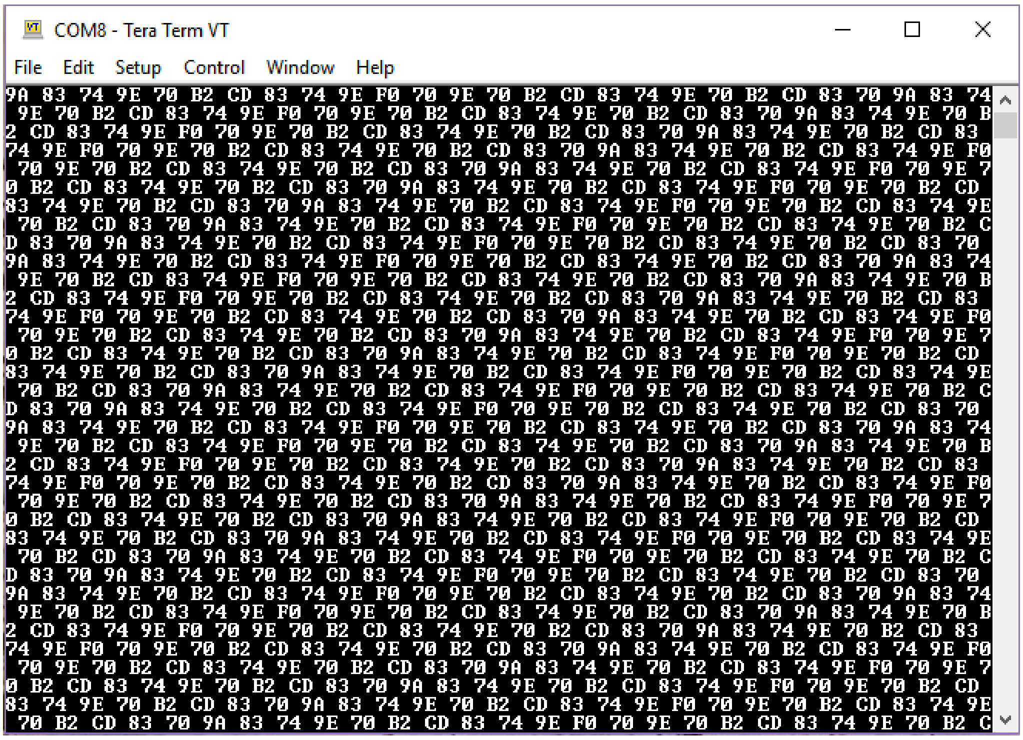


Fig 11. 6 Data receive by host computer

(9) The receiving part has been eliminated here. You are encouraged to try it on your own.

Experiment 12 IIC Protocol Transmission

1.Experiment Objective

There is an IIC interface EEPROM chip 24LC02 in the test plate, capacity sized 2 kbit (256 bite). Since the data is not lost after the EEPROM is powered down, users can store some hardware setup data or user information.

(1) Learning the basic principles of the different IIC bus, mastering the IIC communication protocol

(2) Master the method of reading and writing EEPROM

(3) Joint debugging using logic analyzer

2.Experiment Implement

(1) Correctly write a number to any address in the EEPROM (this experiment writes to the register of 8'h03 address) through the FPGA (here changes the written 8-bit data value by (SW7~SW0)). After writing in successfully, read the data as well. The read data is displayed directly on the segment decoders.

(2) Download the program into the FPGA and press the **Up** button PB2 to execute the data write EEPROM operation. Press the **Return** button PB3 to read the data that was just written.

(3) Determine whether the value read is correct or not by reading the value displayed on the segment decoders. If the segment decoders display the same value as written value, the experiment is successful.

(4) Analyze the correctness of the internal data with ILA and verify it with the display of the segment decoders.

3.Introduction to the IIC Agreement

3.1 The Overall Timing Protocol of IIC Is as Follows

(1) Bus idle state: *SDA*, *SCL* are high

(2) Start of IIC protocol: *SCL* stays high, *SDA* jumps from high level to low level, generating a start signal

(3) IIC read and write data phase: including serial input and output of data and response model issued by data receiver

(4) IIC transmission end bit: *SCL* is high level, *SDA* jumps from low level to high level, and generates an end flag. See Fig 12. 1.

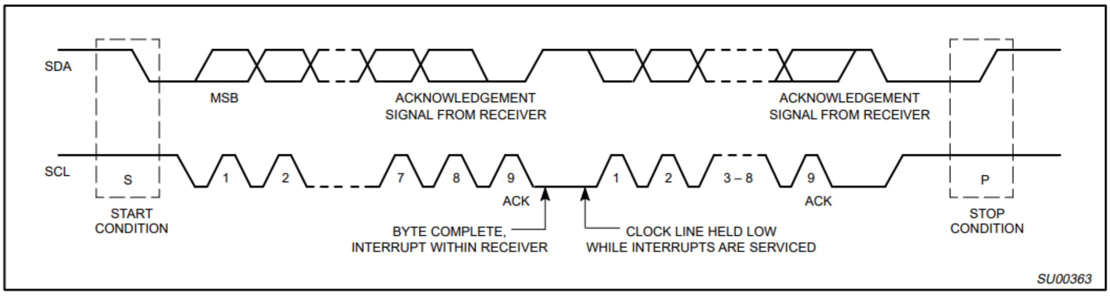


Fig 12. 1 Timing protocol of IIC

3.2 IIC Device Address

Each IIC device has a device address. When some device addresses are shipped from the factory, they are fixed by the manufacturer (the specific data can be found in the manufacturer's data sheet). Some of their higher bits are determined, and the lower bits can be configured by the user according to the requirement. The higher four-bit address of the EEPROM chip 24LC02 used by the develop board has been fixed to 1010 by the component manufacturer. The lower three bits are linked in the develop board as shown below, so the device address is 1010000. (The asterisk resistance indicates that it is not soldered). See Fig 12.2.

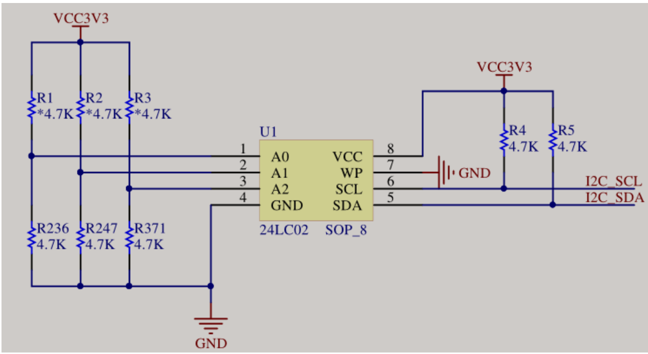


Fig 12. 2 IIC device schematics

4. Main Code

```
module iic_com(
            clk,rst_n,
            data,

            sw1,sw2,
            scl,sda,
            iic_done,
            dis_data
        );

input clk;      // 50MHz
```

```
input rst_n;
input sw1,sw2;
inout scl;
inout sda;
output[7:0] dis_data;
input [7:0] data ;
output reg     iic_done =0 ;
reg   [7:0] data_tep;
reg   scl_link ;

reg     [19:0] cnt_5ms    ;
reg sw1_r,sw2_r;
reg[19:0] cnt_20ms;

always @ (posedge clk or negedge rst_n)
    if(!rst_n) cnt_20ms <= 20'd0;
    else cnt_20ms <= cnt_20ms+1'b1;

always @ (posedge clk or negedge rst_n)
    if(!rst_n) begin
            sw1_r <= 1'b1;
            sw2_r <= 1'b1;
        end
    else if(cnt_20ms == 20'hfffff) begin
            sw1_r <= sw1;
            sw2_r <= sw2;
        end

//---------------------------------------------

reg[2:0] cnt;
reg[8:0] cnt_delay;
reg scl_r;

always @ (posedge clk or negedge rst_n)
    if(!rst_n) cnt_delay <= 9'd0;
    else if(cnt_delay == 9'd499) cnt_delay <= 9'd0;
    else cnt_delay <= cnt_delay+1'b1;

always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) cnt <= 3'd5;
    else begin
        case (cnt_delay)
            9'd124:   cnt <= 3'd1;    //cnt=1:scl
```

```
                9'd249:   cnt <= 3'd2;    //cnt=2:scl
                9'd374:   cnt <= 3'd3;    //cnt=3:scl
                9'd499:   cnt <= 3'd0;    //cnt=0:scl
                default: cnt<=3'd5;
                endcase
            end
end


`define SCL_POS         (cnt==3'd0)         //cnt=0:scl
`define SCL_HIG         (cnt==3'd1)         //cnt=1:scl
`define SCL_NEG         (cnt==3'd2)         //cnt=2:scl
`define SCL_LOW         (cnt==3'd3)         //cnt=3:scl

always @ (posedge clk or negedge rst_n)
        if(!rst_n) data_tep <= 8'h00;
        else     data_tep<= data ;     //



always @ (posedge clk or negedge rst_n)
        if(!rst_n) scl_r <= 1'b0;
        else if(cnt==3'd0) scl_r <= 1'b1;    //scl
        else if(cnt==3'd2) scl_r <= 1'b0;    //scl

assign scl = scl_link?scl_r: 1'bz ;
//---------------------------------------------


`define    DEVICE_READ       8'b1010_0001
`define DEVICE_WRITE   8'b1010_0000
`define    WRITE_DATA        8'b1000_0001
`define BYTE_ADDR       8'b0000_0011
reg[7:0] db_r;
reg[7:0] read_data;

//---------------------------------------------

parameter    IDLE      = 4'd0;
parameter    START1    = 4'd1;
parameter    ADD1      = 4'd2;
parameter    ACK1      = 4'd3;
parameter    ADD2      = 4'd4;
parameter    ACK2      = 4'd5;
```

```
parameter     START2   = 4'd6;
parameter     ADD3     = 4'd7;
parameter     ACK3     = 4'd8;
parameter     DATA     = 4'd9;
parameter     ACK4     = 4'd10;
parameter     STOP1    = 4'd11;
parameter     STOP2    = 4'd12;

reg[3:0] cstate;
reg sda_r;
reg sda_link;
reg[3:0] num;


always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
                        cstate <= IDLE;
                        sda_r <= 1'b1;
                        scl_link <= 1'b1;
                        sda_link <= 1'b1;
                        num <= 4'd0;
                        read_data <= 8'b0000_0000;
                        cnt_5ms      <=20'h00000 ;
                        iic_done<=1'b0 ;
                end
        else
                case (cstate)
                        IDLE:   begin
                                        sda_link <= 1'b1;
                                        scl_link <= 1'b1;
                                        iic_done<=1'b0 ;
                                        if(!sw1_r || !sw2_r) begin
                                                db_r <= `DEVICE_WRITE;
                                                cstate <= START1;
                                                end
                                        else cstate <= IDLE;
                                end
                        START1: begin
                                        if(`SCL_HIG) begin
                                                sda_link <= 1'b1;
                                                sda_r <= 1'b0;
                                                cstate <= ADD1;
                                                num <= 4'd0;
                                                end
```

```
				else cstate <= START1;
			end
		ADD1:	begin
				if(`SCL_LOW) begin
					if(num == 4'd8) begin
						num <= 4'd0;
						sda_r <= 1'b1;
						sda_link <= 1'b0;
						cstate <= ACK1;
					end
					else begin
						cstate <= ADD1;
						num <= num+1'b1;
						case (num)
							4'd0: sda_r <= db_r[7];
							4'd1: sda_r <= db_r[6];
							4'd2: sda_r <= db_r[5];
							4'd3: sda_r <= db_r[4];
							4'd4: sda_r <= db_r[3];
							4'd5: sda_r <= db_r[2];
							4'd6: sda_r <= db_r[1];
							4'd7: sda_r <= db_r[0];
							default: ;
							endcase
					//	sda_r <= db_r[4'd7-num];
					end
				end
		//	else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
				else cstate <= ADD1;
			end
		ACK1:	begin
				if(/*!sda*/`SCL_NEG) begin
					cstate <= ADD2;
					db_r <= `BYTE_ADDR;
				end
				else cstate <= ACK1;
			end
		ADD2:	begin
				if(`SCL_LOW) begin
					if(num==4'd8) begin
						num <= 4'd0;
						sda_r <= 1'b1;
						sda_link <= 1'b0;
						cstate <= ACK2;
```

```
                    end
                else begin
                        sda_link <= 1'b1;
                        num <= num+1'b1;
                        case (num)
                            4'd0: sda_r <= db_r[7];
                            4'd1: sda_r <= db_r[6];
                            4'd2: sda_r <= db_r[5];
                            4'd3: sda_r <= db_r[4];
                            4'd4: sda_r <= db_r[3];
                            4'd5: sda_r <= db_r[2];
                            4'd6: sda_r <= db_r[1];
                            4'd7: sda_r <= db_r[0];
                            default: ;
                            endcase
                //      sda_r <= db_r[4'd7-num];
                        cstate <= ADD2;
                    end
            end
//      else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
        else cstate <= ADD2;
    end
ACK2:   begin
        if(/*!sda*/`SCL_NEG) begin
            if(!sw1_r) begin
                    cstate <= DATA;
                    db_r <= data_tep;
                end
            else if(!sw2_r) begin
                    db_r <= `DEVICE_READ;
                    cstate <= START2;
                end
            end
        else cstate <= ACK2;
    end
START2: begin
        if(`SCL_LOW) begin
            sda_link <= 1'b1;
            sda_r <= 1'b1;
            cstate <= START2;
            end
        else if(`SCL_HIG) begin
            sda_r <= 1'b0;
```

```
                    cstate <= ADD3;
                    end
                else cstate <= START2;
            end
        ADD3:   begin
                if(`SCL_LOW) begin
                        if(num==4'd8) begin
                                num <= 4'd0;
                                sda_r <= 1'b1;
                                sda_link <= 1'b0;
                                cstate <= ACK3;
                            end
                        else begin
                                num <= num+1'b1;
                                case (num)
                                    4'd0: sda_r <= db_r[7];
                                    4'd1: sda_r <= db_r[6];
                                    4'd2: sda_r <= db_r[5];
                                    4'd3: sda_r <= db_r[4];
                                    4'd4: sda_r <= db_r[3];
                                    4'd5: sda_r <= db_r[2];
                                    4'd6: sda_r <= db_r[1];
                                    4'd7: sda_r <= db_r[0];
                                    default: ;
                                    endcase

                            //  sda_r <= db_r[4'd7-num];
                                cstate <= ADD3;
                            end
                    end
            //  else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
                else cstate <= ADD3;
            end
        ACK3:   begin
                if(/*!sda*/`SCL_NEG) begin
                        cstate <= DATA;
                        sda_link <= 1'b0;
                    end
                else cstate <= ACK3;
            end
        DATA:   begin
                if(!sw2_r) begin
                        if(num<=4'd7) begin
                            cstate <= DATA;
```

```
                    if(`SCL_HIG) begin
                        num <= num+1'b1;
                        case (num)
                            4'd0: read_data[7] <= sda;
                            4'd1: read_data[6] <= sda;
                            4'd2: read_data[5] <= sda;
                            4'd3: read_data[4] <= sda;
                            4'd4: read_data[3] <= sda;
                            4'd5: read_data[2] <= sda;
                            4'd6: read_data[1] <= sda;
                            4'd7: read_data[0] <= sda;
                            default: ;
                            endcase

        //                      read_data[4'd7-num] <= sda;
                        end
    //                  else if(`SCL_NEG) read_data <=
{read_data[6:0],read_data[7]};
                    end
                else if((`SCL_LOW) && (num==4'd8)) begin
                    num <= 4'd0;
                    cstate <= ACK4;
                    end
                else cstate <= DATA;
            end
        else if(!sw1_r) begin
                sda_link <= 1'b1;
                if(num<=4'd7) begin
                    cstate <= DATA;
                    if(`SCL_LOW) begin
                        sda_link <= 1'b1;
                        num <= num+1'b1;
                        case (num)
                            4'd0: sda_r <= db_r[7];
                            4'd1: sda_r <= db_r[6];
                            4'd2: sda_r <= db_r[5];
                            4'd3: sda_r <= db_r[4];
                            4'd4: sda_r <= db_r[3];
                            4'd5: sda_r <= db_r[2];
                            4'd6: sda_r <= db_r[1];
                            4'd7: sda_r <= db_r[0];
                            default: ;
                            endcase
```

```
                                        //     sda_r <= db_r[4'd7-num];
                                               end
                 //                     else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
                                        end
                                  else if((`SCL_LOW) && (num==4'd8)) begin
                                               num <= 4'd0;
                                               sda_r <= 1'b1;
                                               sda_link <= 1'b0;
                                               cstate <= ACK4;
                                        end
                                  else cstate <= DATA;
                            end
                      end
                 ACK4: begin
                            if(/*!sda*/`SCL_NEG) begin
//                                 sda_r <= 1'b1;
                                   cstate <= STOP1;
                                   end
                            else cstate <= ACK4;
                      end
                 STOP1:     begin
                            if(`SCL_LOW) begin
                                        sda_link <= 1'b1;
                                        sda_r <= 1'b0;
                                        cstate <= STOP1;
                                   end
                            else if(`SCL_HIG) begin
                                        sda_r <= 1'b1;
                                        cstate <= STOP2;
                                   end
                            else cstate <= STOP1;
                      end
                 STOP2:     begin
                            if(`SCL_NEG)    begin     sda_link <= 1'b0;    scl_link <= 1'b0;    end
                            else if(cnt_5ms==20'h3fffc)      begin cstate <= IDLE;
cnt_5ms<=20'h00000;    iic_done<=1 ; end
                            else begin cstate <= STOP2      ;       cnt_5ms<=cnt_5ms+1 ; end
                      end
                 default: cstate <= IDLE;
                 endcase
end

assign sda = sda_link ? sda_r:1'bz;
assign dis_data = read_data;
```

```
//--------------------------------------------

endmodule
```

5.Downloading to The Board

(1) Lock the Pins

Signal Name	Port Description	Network Label	FPGA Pin
clk	System clock, 50 MHz	C10_50MCLK	U22
rst_n	Reset, high by default	KEY1	M4
sm_db[0]	Segment a	SEG_PA	K26
sm_db [1]	Segment b	SEG_PB	M20
sm_db [2]	Segment c	SEG_PC	L20
sm_db [3]	Segment d	SEG_PD	N21
sm_db [4]	Segment e	SEG_PE	N22
sm_db [5]	Segment f	SEG_PF	P21
sm_db [6]	Segment g	SEG_PG	P23
sm_db [7]	Segment h	SEG_DP	P24
sm_cs1_n	Segment 2	SEG_3V3_D0	R16
sm_cs2_n	Segment 1	SEG_3V3_D1	R17
data[0]	Switch input	GPIO_DIP_SW0	N8
data[1]	Switch input	GPIO_DIP_SW1	M5
data[2]	Switch input	GPIO_DIP_SW2	P4
data[3]	Switch input	GPIO_DIP_SW3	N4
data[4]	Switch input	GPIO_DIP_SW4	U6
data[5]	Switch input	GPIO_DIP_SW5	U5
data[6]	Switch input	GPIO_DIP_SW6	R8
data[7]	Switch input	GPIO_DIP_SW7	P8
sw1	Write EEPROM	KEY2	L4
sw2	Read EEPROM	KEY3	L5
scl	EEPROM clock	I2C_SCl	R20
sda	EEPROM data line	I2C_SDA	R21

(2) After the program is downloaded to the board, press the **Up** push button PB2 to write the 8-bit value represented by SW7~SW0 to EEPROM. Then press the **Return** button PB3 to read the value from the written position. Observe the value displayed on the segment decoders on the develop board and the value written in the 8'h03 register of the EEPROM address (SW7~SW0) (Here, it writes to 8'h37 address). The read value is displayed on the segment decoders. See Fig 12. 3.

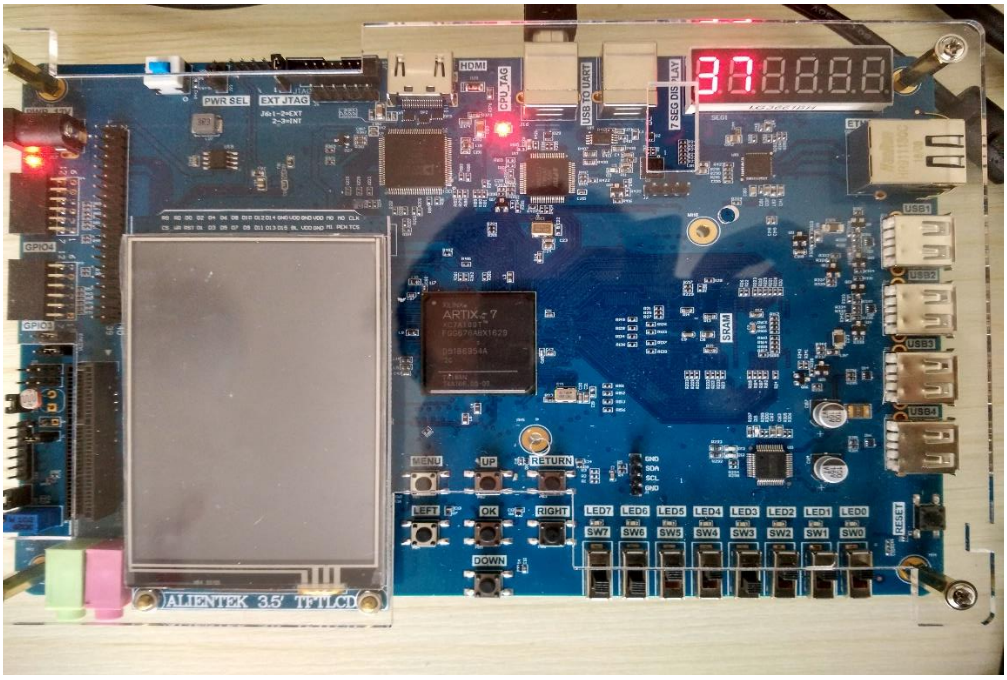


Fig 12. 3 Demonstration of the develop board

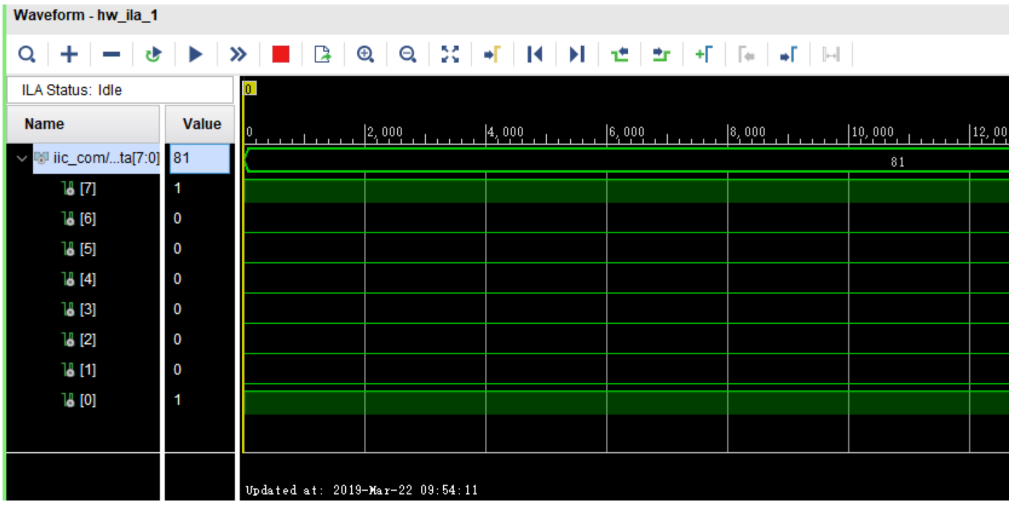


Fig 12. 4 ILA demonstration

6.More to Practice

(1) Try to write to eeprom multiple non-contiguous addresses and read them. Prepare for the next experiment.

Experiment 13 AD, DA Experiment

1.Experiment Objective

Since in the real world, all naturally occurring signals are analog signals, and all that are read and processed in actual engineering are digital signals. There is a process of mutual conversion between natural and industrial signals (digital-to-analog conversion: DAC, analog-to-digital conversion: ADC). The purpose of this experiment is twofold:

(1) Learning the theory of AD conversion

(2) Read the value of AD acquisition from PCF8591, and convert the value obtained into actual value, display it with segment decoders

2.Experiment Implement

(1) Perform analog-to-digital conversion using the ADC port of the chip and display the collected voltage value through the segment decoders.

(2) Board downloading verification for comparison

(3) Introduction to PCF8591: The PCF8591 uses the IIC bus protocol to communicate with the controller (FPGA). Please refer to the previous experiment for the contents of the IIC bus protocol. The first four bits of the device address are 1001, and the last three bits are determined by the actual circuit connection (here the circuit is grounded, so the device address is 7'b1001000). The LSB is the read/write control signal. After sending the device address information and the read/write control word are done, the control word information is sent. The specific control word information is shown in Fig 13. 1.

Bit	Slave address							
	7 MSB	6	5	4	3	2	1	0 LSB
slave address	1	0	0	1	A2	A1	A0	R/$\overline{W}$

Fig 13. 1 PCF8591 Control address

Here, the experiment uses the DIP switch (SW1, SW0) input channel as the AD acquisition input channel. Configure the control information as (8’h40). For more details, refer to the datasheet of PCF8591.

SW1，SW0	Channel Selection	Collection Object
00	0	Photosensitive Resistor Voltage Value
01	1	Thermistor Voltage Value
10	2	Adjustable Voltage Value

3.Experiment Design

(1) Program design and review the top-down design method used before.

(2) The top-level entity is divided into three parts: the segment decoder driver part, the AD

sampling part of the PCF and the IIC serial port driver part.

IIC serial driver part code is as follows:

```
module iic_com (
                clk,rst_n,
                data,
                sw1,sw2,
                scl,sda,
                iic_done,
                dis_data
            );

input clk;      // 50MHz
input rst_n;
input sw1,sw2;
inout scl;
inout sda;
output reg [7:0] dis_data=8'h00;
input [7:0] data ;
output reg    iic_done =0 ;
reg   [7:0] data_tep;
reg   scl_link ;

reg     [19:0] cnt_5ms    ;
reg sw1_r,sw2_r;
reg[19:0] cnt_20ms;

always @ (posedge clk or negedge rst_n)
    if(!rst_n) cnt_20ms <= 20'd0;
    else cnt_20ms <= cnt_20ms+1'b1;

always @ (posedge clk or negedge rst_n)
    if(!rst_n) begin
            sw1_r <= 1'b1;
            sw2_r <= 1'b1;
        end
    else if(cnt_20ms == 20'hfffff) begin
            sw1_r <= sw1;
            sw2_r <= sw2;
        end
reg[2:0] cnt;
reg[8:0] cnt_delay;
reg scl_r;
reg [7:0] read_data_temp [15:0];
reg [11:0]   dis_data_temp ;
```

```
always @ (posedge clk )
dis_data_temp<=read_data_temp[0]+read_data_temp[1]+read_data_temp[2]+read_data_te
mp[3]+read_data_temp[4]+read_data_temp[5]+read_data_temp[6]+read_data_temp[7]+read
_data_temp[8]+read_data_temp[9]+read_data_temp[10]+read_data_temp[11]+read_data_te
mp[12]+read_data_temp[13]+read_data_temp[14]+read_data_temp[15];

always @ (posedge clk )
        dis_data <= dis_data_temp>>4 ;

integer i;
always @ (posedge clk or negedge rst_n)
    if(!rst_n) begin
                for (i=0;i<16;i=i+1)
                        read_data_temp[i]<=8'h00;
    end
    else if    (iic_done)    begin
                for (i=0;i<15;i=i+1)
                        read_data_temp[i+1]<=read_data_temp[i];
                        read_data_temp[0] <= read_data ;
                end
                else begin for (i=0;i<16;i=i+1)
                        read_data_temp[i]<=read_data_temp[i];
                end
always @ (posedge clk or negedge rst_n)
        if(!rst_n) cnt_delay <= 9'd0;
        else if(cnt_delay == 9'd499) cnt_delay <= 9'd0;
        else cnt_delay <= cnt_delay+1'b1;

always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) cnt <= 3'd5;
        else begin
                case (cnt_delay)
                        9'd124:    cnt <= 3'd1;     //cnt=1:scl
                        9'd249:    cnt <= 3'd2;     //cnt=2:scl
                        9'd374:    cnt <= 3'd3;     //cnt=3:scl
                        9'd499:    cnt <= 3'd0;     //cnt=0:scl
                        default: cnt<=3'd5;
                        endcase
                end
end

`define SCL_POS          (cnt==3'd0)          //cnt=0:scl
`define SCL_HIG          (cnt==3'd1)          //cnt=1:scl
```

```
`define SCL_NEG          (cnt==3'd2)          //cnt=2:scl
`define SCL_LOW          (cnt==3'd3)          //cnt=3:scl

always @ (posedge clk or negedge rst_n)
        if(!rst_n) data_tep <= 8'h00;
        else      data_tep<= data ;      //
always @ (posedge clk or negedge rst_n)
        if(!rst_n) scl_r <= 1'b0;
        else if(cnt==3'd0) scl_r <= 1'b1;     //scl
        else if(cnt==3'd2) scl_r <= 1'b0;     //scl
assign scl = scl_link?scl_r: 1'bz ;
                                `define    DEVICE_READ {7'h48,1'b1}
`define DEVICE_WRITE   {7'h48,1'b0}
`define    WRITE_DATA   8'b1000_0001
`define BYTE_ADDR       8'b0000_0011
reg[7:0] db_r;
reg[7:0] read_data;

parameter      IDLE        = 4'd0;
parameter      START1      = 4'd1;
parameter      ADD1        = 4'd2;
parameter      ACK1        = 4'd3;
parameter      ADD2        = 4'd4;
parameter      ACK2        = 4'd5;
parameter      START2      = 4'd6;
parameter      ADD3        = 4'd7;
parameter      ACK3        = 4'd8;
parameter      DATA        = 4'd9;
parameter      ACK4        = 4'd10;
parameter      STOP1       = 4'd11;
parameter      STOP2       = 4'd12;

reg[3:0] cstate;
reg sda_r;
reg sda_link;
reg[3:0] num;

always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
                cstate <= IDLE;
                sda_r <= 1'b1;
                scl_link <= 1'b1;
                sda_link <= 1'b1;
                num <= 4'd0;
```

```
                read_data <= 8'b0000_0000;
                cnt_5ms     <=20'h00000 ;
                iic_done<=1'b0 ;
            end
        else
            case (cstate)
                IDLE:       begin
                        sda_link <= 1'b1;
                        scl_link <= 1'b1;
                        iic_done<=1'b0 ;
                        if(!sw1_r || !sw2_r) begin
                            db_r <= `DEVICE_WRITE;
                            cstate <= START1;
                            end
                        else cstate <= IDLE;
                    end
                START1: begin
                        if(`SCL_HIG) begin
                            sda_link <= 1'b1;
                            sda_r <= 1'b0;
                            cstate <= ADD1;
                            num <= 4'd0;
                            end
                        else cstate <= START1;
                    end
                ADD1:       begin
                        if(`SCL_LOW) begin
                                if(num == 4'd8) begin
                                        num <= 4'd0;
                                        sda_r <= 1'b1;
                                        sda_link <= 1'b0;
                                        cstate <= ACK1;
                                    end
                                else begin
                                        cstate <= ADD1;
                                        num <= num+1'b1;
                                        case (num)
                                            4'd0: sda_r <= db_r[7];
                                            4'd1: sda_r <= db_r[6];
                                            4'd2: sda_r <= db_r[5];
                                            4'd3: sda_r <= db_r[4];
                                            4'd4: sda_r <= db_r[3];
                                            4'd5: sda_r <= db_r[2];
                                            4'd6: sda_r <= db_r[1];
```

```
                                                4'd7: sda_r <= db_r[0];
                                                default: ;
                                                endcase
                                //          sda_r <= db_r[4'd7-num];
                                    end
                            end
//          else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
            else cstate <= ADD1;
        end
ACK1:       begin
            if(/*!sda*/`SCL_NEG) begin
                        cstate <= ADD2;
                        db_r <= {6'b0100_00,data_tep[1:0]};
                    end
            else cstate <= ACK1;
        end
ADD2:       begin
            if(`SCL_LOW) begin
                        if(num==4'd8) begin
                                    num <= 4'd0;
                                    sda_r <= 1'b1;
                                    sda_link <= 1'b0;
                                    cstate <= ACK2;

                                end
                        else begin
                                    sda_link <= 1'b1;
                                    num <= num+1'b1;
                                    case (num)
                                        4'd0: sda_r <= db_r[7];
                                        4'd1: sda_r <= db_r[6];
                                        4'd2: sda_r <= db_r[5];
                                        4'd3: sda_r <= db_r[4];
                                        4'd4: sda_r <= db_r[3];
                                        4'd5: sda_r <= db_r[2];
                                        4'd6: sda_r <= db_r[1];
                                        4'd7: sda_r <= db_r[0];
                                        default: ;
                                        endcase
                        //          sda_r <= db_r[4'd7-num];
                                    cstate <= ADD2;
                                end
                    end
//          else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
```

```
                else cstate <= ADD2;
            end
        ACK2:   begin
                if(/*!sda*/`SCL_NEG) begin
                    if(!sw1_r) begin
                            cstate <= DATA;
                            db_r <= data_tep;
                        end
                    else if(!sw2_r) begin
                            db_r <= `DEVICE_READ;
                            cstate <= START2;
                        end
                    end
                else cstate <= ACK2;
            end
        START2: begin
                if(`SCL_LOW) begin
                    sda_link <= 1'b1;
                    sda_r <= 1'b1;
                    cstate <= START2;
                    end
                else if(`SCL_HIG) begin
                    sda_r <= 1'b0;
                    cstate <= ADD3;
                    end
                else cstate <= START2;
            end
        ADD3:   begin
                if(`SCL_LOW) begin
                        if(num==4'd8) begin
                                num <= 4'd0;
                                sda_r <= 1'b1;
                                sda_link <= 1'b0;
                                cstate <= ACK3;
                            end
                        else begin
                                num <= num+1'b1;
                                case (num)
                                    4'd0: sda_r <= db_r[7];
                                    4'd1: sda_r <= db_r[6];
                                    4'd2: sda_r <= db_r[5];
                                    4'd3: sda_r <= db_r[4];
                                    4'd4: sda_r <= db_r[3];
                                    4'd5: sda_r <= db_r[2];
```

```
                                        4'd6: sda_r <= db_r[1];
                                        4'd7: sda_r <= db_r[0];
                                        default: ;
                                        endcase

                                //      sda_r <= db_r[4'd7-num];
                                        cstate <= ADD3;
                                end
                        end
                //      else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
                        else cstate <= ADD3;
                end
        ACK3:   begin
                        if(/*!sda*/`SCL_NEG) begin
                                cstate <= DATA;
                                sda_link <= 1'b0;
                        end
                        else cstate <= ACK3;
                end
        DATA:   begin
                        if(!sw2_r) begin
                                if(num<=4'd7) begin
                                        cstate <= DATA;
                                        if(`SCL_HIG) begin
                                                num <= num+1'b1;
                                                case (num)
                                                        4'd0: read_data[7] <= sda;
                                                        4'd1: read_data[6] <= sda;
                                                        4'd2: read_data[5] <= sda;
                                                        4'd3: read_data[4] <= sda;
                                                        4'd4: read_data[3] <= sda;
                                                        4'd5: read_data[2] <= sda;
                                                        4'd6: read_data[1] <= sda;
                                                        4'd7: read_data[0] <= sda;
                                                        default: ;
                                                        endcase

                        //                      read_data[4'd7-num] <= sda;
                                                end
                //                      else            if(`SCL_NEG)            read_data               <=
{read_data[6:0],read_data[7]};
                                        end
                                else if((`SCL_LOW) && (num==4'd8)) begin
                                        num <= 4'd0;
```

```
                                    cstate <= ACK4;
                                    end
                              else cstate <= DATA;
                        end
                  else if(!sw1_r) begin
                              sda_link <= 1'b1;
                              if(num<=4'd7) begin
                                    cstate <= DATA;
                                    if(`SCL_LOW) begin
                                          sda_link <= 1'b1;
                                          num <= num+1'b1;
                                          case (num)
                                                4'd0: sda_r <= db_r[7];
                                                4'd1: sda_r <= db_r[6];
                                                4'd2: sda_r <= db_r[5];
                                                4'd3: sda_r <= db_r[4];
                                                4'd4: sda_r <= db_r[3];
                                                4'd5: sda_r <= db_r[2];
                                                4'd6: sda_r <= db_r[1];
                                                4'd7: sda_r <= db_r[0];
                                                default: ;
                                                endcase

                                    //    sda_r <= db_r[4'd7-num];
                                          end
            //                      else if(`SCL_POS) db_r <= {db_r[6:0],1'b0};
                                    end
                              else if((`SCL_LOW) && (num==4'd8)) begin
                                          num <= 4'd0;
                                          sda_r <= 1'b1;
                                          sda_link <= 1'b0;
                                          cstate <= ACK4;
                                    end
                              else cstate <= DATA;
                        end
            end
            ACK4: begin
                  if(/*!sda*/`SCL_NEG) begin
//                      sda_r <= 1'b1;
                        cstate <= STOP1;
                        end
                  else cstate <= ACK4;
            end
            STOP1:    begin
```

```
                    if(`SCL_LOW) begin
                              sda_link <= 1'b1;
                              sda_r <= 1'b0;
                              cstate <= STOP1;
                         end
                    else if(`SCL_HIG) begin
                              sda_r <= 1'b1;
                              cstate <= STOP2;
                         end
                    else cstate <= STOP1;
               end
          STOP2:    begin
                    if(`SCL_NEG)   begin     sda_link <= 1'b0;   scl_link <= 1'b0;   end
                    else     if(cnt_5ms==20'h3fffc)          begin     cstate     <=     IDLE;
cnt_5ms<=20'h00000;   iic_done<=1 ; end
                    else begin cstate <= STOP2     ;     cnt_5ms<=cnt_5ms+1 ; end
               end
          default: cstate <= IDLE;
          endcase
end

assign sda = sda_link ? sda_r:1'bz;

endmodule
```

Segment decoder driver part is as follow:

```
module led_seg7(
          clk,rst_n,
          dis_data,
          point1 ,
          sel,sm_db
     );

input clk;
input rst_n;

input[7:0] dis_data;
output   reg [5:0] sel;
output[6:0] sm_db;
output reg point1 ;

reg[11:0] cnt1;
reg[2:0] cnt;
```

```
reg [3:0] num;

wire en=(cnt1==12'hfff) ?1:0 ;
parameter       V_REF       = 12'd3300       ;

reg     [19:0]     num_t         ;

reg       [19:0]     num1 ;

always @ (posedge clk)
    num_t <= V_REF *dis_data ;

wire     [3:0]                    data0       ;
wire     [3:0]                    data1       ;
wire     [3:0]                    data2       ;
wire     [3:0]                    data3       ;
wire     [3:0]                    data4       ;
wire     [3:0]                    data5       ;

assign    data5 = num1 / 17'd100000;
assign    data4 = num1 / 14'd10000 % 4'd10;
assign    data3 = num1 / 10'd1000 % 4'd10 ;
assign    data2 = num1 /    7'd100 % 4'd10    ;
assign    data1 = num1 /    4'd10 % 4'd10      ;
assign    data0 = num1 %    4'd10;

always @(posedge clk or negedge rst_n) begin
        if(rst_n == 1'b0) begin
                num1 <= 20'd0;
        end
        else
                num1 <= num_t >> 4'd8;
end

always @ (posedge clk or negedge rst_n)
        if(!rst_n) cnt1 <= 4'd0;
        else cnt1 <= cnt1+1'b1;

parameter      seg0 = 7'h3f,
               seg1 = 7'h06,
               seg2 = 7'h5b,
               seg3 = 7'h4f,
               seg4 = 7'h66,
```

```
            seg5 = 7'h6d,
            seg6 = 7'h7d,
            seg7 = 7'h07,
            seg8 = 7'h7f,
            seg9 = 7'h6f,
            sega = 7'h77,
            segb = 7'h7c,
            segc = 7'h39,
            segd = 7'h5e,
            sege = 7'h79,
            segf  = 7'h71;

reg[6:0] sm_dbr;

always @ (*)
        case (num)
            4'h0: sm_dbr = seg0;
            4'h1: sm_dbr = seg1;
            4'h2: sm_dbr = seg2;
            4'h3: sm_dbr = seg3;
            4'h4: sm_dbr = seg4;
            4'h5: sm_dbr = seg5;
            4'h6: sm_dbr = seg6;
            4'h7: sm_dbr = seg7;
            4'h8: sm_dbr = seg8;
            4'h9: sm_dbr = seg9;
            4'ha: sm_dbr = sega;
            4'hb: sm_dbr = segb;
            4'hc: sm_dbr = segc;
            4'hd: sm_dbr = segd;
            4'he: sm_dbr = sege;
            4'hf: sm_dbr = segf;
            default: ;
            endcase

assign sm_db = sm_dbr;

always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        sel    <= 6'b000000;
        num <= 4'b0;
        cnt<=3'b000;
    end
    else begin
```

```
case (cnt)
    3'd0 :begin
        sel      <= 6'b111111;
        num      <= data5 ;
        point1 <= 1'b1 ;
          if(en)
          cnt<=3'd1 ;

    end
    3'd1 :begin
        sel      <= 6'b111111;
        num      <= data4 ;
        point1 <=1'b1 ;
          if(en)
        cnt<=3'd2 ;

    end
    3'd2 :begin
        sel      <= 6'b111011;
        num      <= data3;
        point1 <= 1'b0 ;
        if(en)
      cnt<=3'd3 ;

    end
    3'd3 :begin
        sel      <= 6'b110111;
        num      <= data2;
        point1 <= 1'b1    ;
        if(en)
        cnt<=3'd4 ;

    end
    3'd4 :begin
        sel      <= 6'b101111;
        num      <= data1;
        point1 <=1'b1;

      if(en)
  cnt<=3'd5 ;

    end
    3'd5 :begin
```

```
                    sel       <= 6'b011111;
                    num       <= data0;
                    point1 <=1'b1;
                    if(en)
                    cnt<=3'd0 ;

                end
                default :begin
                    sel       <= 6'b000000;
                    num       <= 4'h0;
                    point1 <= 1'b1;
                end
            endcase

    end
end

endmodule
```

4.Downloading to The Board

(1) Lock the pins

SignaL Name	Port Description	Network Label	FPGA Pin
clk	System clock, 50 MHz	C10_50MCLK	U22
rst_n	Reset, high by default	KEY1	M4
sm_db[0]	Segment a	SEG_PA	K26
sm_db [1]	Segment b	SEG_PB	M20
sm_db [2]	Segment c	SEG_PC	L20
sm_db [3]	Segment d	SEG_PD	N21
sm_db [4]	Segment e	SEG_PE	N22
sm_db [5]	Segment f	SEG_PF	P21
sm_db [6]	Segment g	SEG_PG	P23
sm_db [7]	Segment h	SEG_DP	P24
data[0]	Switch input	GPIO_DIP_SW0	N8
data[1]	Switch input	GPIO_DIP_SW1	M5
data[2]	Switch input	GPIO_DIP_SW2	P4
data[3]	Switch input	GPIO_DIP_SW3	N4
data[4]	Switch input	GPIO_DIP_SW4	U6
data[5]	Switch input	GPIO_DIP_SW5	U5
data[6]	Switch input	GPIO_DIP_SW6	R8
data[7]	Switch input	GPIO_DIP_SW7	P8
scl	PCF8591 clock	ADDA_I2C_SCl	E20
sda	PCF8591 data line	ADDA_I2C_SDA	C19

sel[0]	Segment decoder position selection	SEG_3V3_D0	R16
sel[1]	Segment decoder position selection	SEG_3V3_D1	R17
sel[2]	Segment decoder position selection	SEG_3V3_D2	N18
sel[3]	Segment decoder position selection	SEG_3V3_D3	K25
sel[4]	Segment decoder position selection	SEG_3V3_D4	R25
sel[5]	Segment decoder position selection	SEG_3V3_D5	T24

(2) Testing by selecting SW0 and SW1 to change the measurement objects.

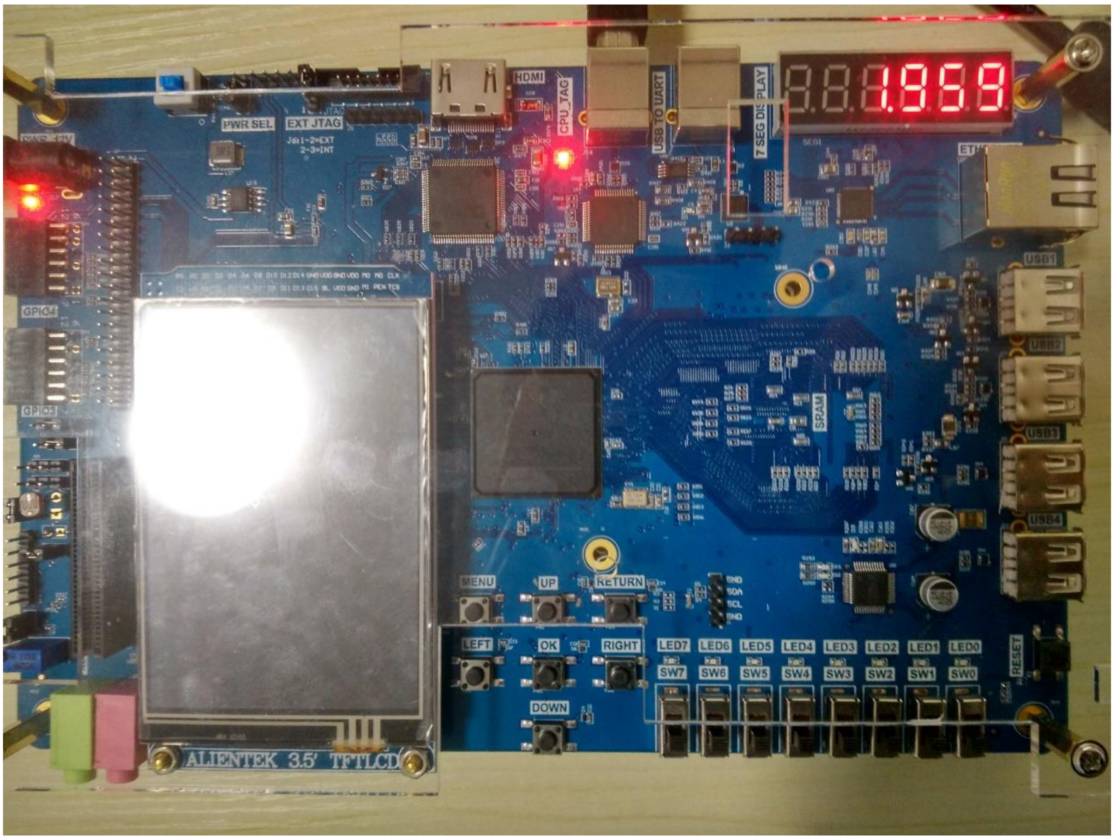


Fig 13. 2 Test result

Experiment 14 HDMI Graphic Display Experiment

1.Experiment Objective

(1) Learn about video timing
(2) Understand the register configuration of the ADV7511, reviewing the knowledge from experiment 12

2.Experiment Implement

(1) Image display processing has always been the focus of FPGA research. At present, the image display mode is also developing. The image display interface is also gradually transitioning from the old VGA interface to the new DVI or HDMI interface.
(2) Display the image using the HDMI interface of the development board.
(3) Download the program to the board for comparison.
(4) Introduction to HDMI: HDMI (High Definition Multimedia Interface) is a digital video/audio interface technology. It is a dedicated digital interface for image transmission. It can transmit audio and video signals at the same time.
(5) Introduction to ADV7511: The ADV7511 is a chip that converts FPGA digital signal to HDMI signal following VESA standard. For more details, see the related chip manual. Among them, “ADV7511 Programming Guide” and “ADV7511 Hardware Users Guide” are the most important. From Table 16 on page 27 of “ADV7511 Programming Guide”, the bit width and format type of RGB can be configured. Its registers can output the appropriate format according to needs after configuration.
(6) ADV7511 Register Configuration Description: The bus inputs D0-D3, D12-D15, and D24-D27 of the ADV7511 have no input, that is, RGB4:4:4, and each bit of data is in 8-bit mode. Directly set 0x15 [3:0] to 0x0. Set [5:4] of 0X16 to 11 and keep the default values for the other digits. 0x17[1] refers to the ratio of the length to the width of the image. It can be set to 0 or 1. The actual LCD screen will not change according to the data, but will automatically stretch the full screen mode according to the LCD's own settings. 0x18[7] is the way to start the color range stretching. The design is that RGB maps directly to RGB, so it can be disabled directly. 0XAF[1] is the setting of choosing either HDMI or DVI mode. The most direct point of HDMI over DVI is that HDMI can send digital audio data and encrypt data content. This experiment only needs to display the picture, and it can be set directly to DVI mode. 0XAF[7], set to 0 to turn off HDMI encryption. Due to GCCD, deep color encryption data is not applicable, so the GC option is turned off. 0xAF[7] is set to 0 to turn off the GC CD data.

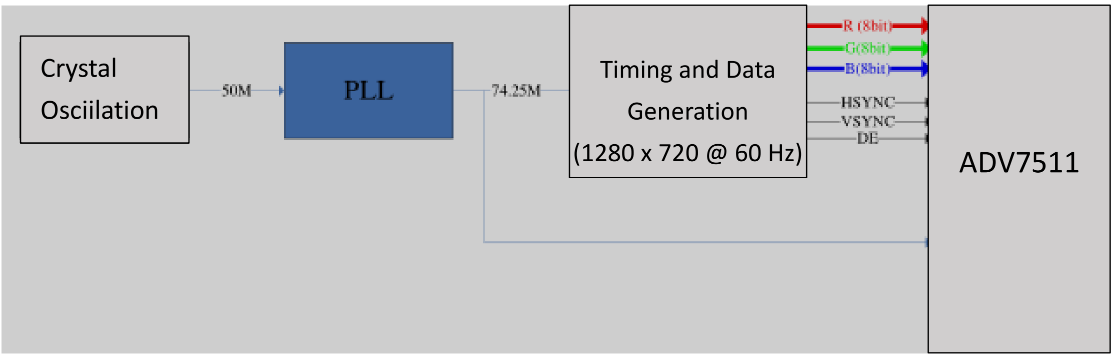


Fig 14. HDMI connection block diagram

3.Program Design

3.1 Schematics

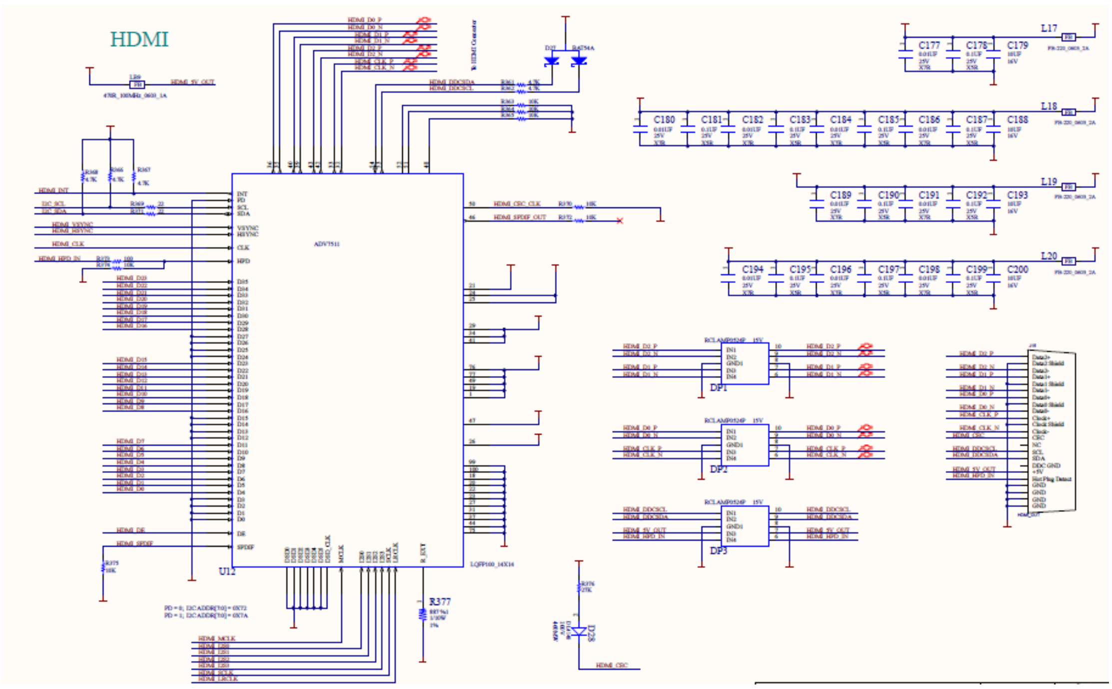


Fig 14. 2 Schematics of ADV7511

3.2 Main Code

(1) 1080p VGA main part of the timing generation program

```
// Set horizontal scanning parameter 1920*1080 60Hz VGA        Clock bit 130 MHz
//-----------------------------------------------------------//
parameter LinePeriod =2000;
parameter H_SyncPulse=12;
parameter H_BackPorch=40;
parameter H_ActivePix=1920;
```

```
    parameter H_FrontPorch=28;
    parameter Hde_start=52;
    parameter Hde_end=1972;
    //--------------------------------------------------------//

    //--------------------------------------------------------//
    parameter FramePeriod =1105;
    parameter V_SyncPulse=4;
    parameter V_BackPorch=18;
    parameter V_ActivePix=1080;
    parameter V_FrontPorch=3;
    parameter Vde_start=22;
    parameter Vde_end=1102;
  reg     [12 : 0]   x_cnt;
  reg     [10 : 0]   y_cnt;
  reg     [23 : 0] grid_data_1;
  reg     [23 : 0] grid_data_2;
  reg     [23 : 0] bar_data;
  reg     [3 :   0] vga_dis_mode;
  reg     [7 :   0]   vga_r_reg;
  reg     [7 :   0]   vga_g_reg;
  reg     [7 :   0]   vga_b_reg;
  reg hsync_r;
  reg vsync_r;
  reg hsync_de;
  reg vsync_de;
  reg [15:0] key1_counter;
  reg   rst ;
  wire [12:0]   bar_interval;

assign     bar_interval     = H_ActivePix[15: 3];

always @ (posedge vga_clk)
    rst<= !locked ;

always @ (posedge vga_clk)
        if(rst)      x_cnt <= 1;
        else if(x_cnt == LinePeriod) x_cnt <= 1;
        else x_cnt <= x_cnt+ 1;

//-----------------------------------------------------------------
//-----------------------------------------------------------------
always @ (posedge vga_clk)
   begin
```

```
            if(rst) hsync_r <= 1'b1;
            else if(x_cnt == 1) hsync_r <= 1'b0;
            else if(x_cnt == H_SyncPulse) hsync_r <= 1'b1;


              if(rst) hsync_de <= 1'b0;
            else if(x_cnt == Hde_start) hsync_de <= 1'b1;
            else if(x_cnt == Hde_end) hsync_de <= 1'b0;
       end

always @ (posedge vga_clk)
            if(rst) y_cnt <= 1;
            else if(y_cnt == FramePeriod) y_cnt <= 1;
            else if(x_cnt == LinePeriod) y_cnt <= y_cnt+1;
always @ (posedge vga_clk)
   begin
            if(rst) vsync_r <= 1'b1;
            else if(y_cnt == 1) vsync_r <= 1'b0;
            else if(y_cnt == V_SyncPulse) vsync_r <= 1'b1;

              if(rst) vsync_de <= 1'b0;
            else if(y_cnt == Vde_start) vsync_de <= 1'b1;
            else if(y_cnt == Vde_end) vsync_de <= 1'b0;
   end

assign    en    = hsync_de & vsync_de ;
always @(posedge    vga_clk)
     begin
        if ((x_cnt[4]==1'b1) ^ (y_cnt[4]==1'b1))
                         grid_data_1<= 24'h000000;
          else
                         grid_data_1<= 24'hffffff;

          if ((x_cnt[6]==1'b1) ^ (y_cnt[6]==1'b1))
                         grid_data_2<=24'h000000;
          else
                           grid_data_2<=24'hffffff;


       end
always @(posedge    vga_clk)
     begin
        if (x_cnt==Hde_start)
                         bar_data<= 24'hff0000;                    //Red strip
          else if (x_cnt==Hde_start + bar_interval)
```

```
                    bar_data<= 24'h00ff00;                  //Green strip
        else if (x_cnt==Hde_start + bar_interval*2)
                    bar_data<=24'h0000ff;                   //Blue strip
        else if (x_cnt==Hde_start + bar_interval*3)
                    bar_data<=24'hff00ff;                   //Purple strip
        else if (x_cnt==Hde_start + bar_interval*4)
                    bar_data<=24'hffff00;                   //Yellow strip
        else if (x_cnt==Hde_start + bar_interval*5)
                    bar_data<=24'h00ffff;                   //Light blue strip
        else if (x_cnt==Hde_start + bar_interval*6)
                    bar_data<=24'hffffff;                   //White strip
        else if (x_cnt==Hde_start + bar_interval*7)
                    bar_data<=24'hff8000;                   //Orange strip
        else if (x_cnt==Hde_start + bar_interval*8)
                    bar_data<=24'h000000;                   //Black strip
      end

always @(posedge vga_clk)
      if(rst) begin
            vga_r_reg<=0;
            vga_g_reg<=0;
            vga_b_reg<=0;
      end
    else
       case(vga_dis_mode)
            4'b0000:begin
                        vga_r_reg<=0;                             // all black
                   vga_g_reg<=0;
                   vga_b_reg<=0;
               end
               4'b0001:begin
                         vga_r_reg<=8'hff;                   // all white
                   vga_g_reg<=8'hff;
                   vga_b_reg<=8'hff;
               end
               4'b0010:begin
                      vga_r_reg<=8'hff;                 // all red
                   vga_g_reg<=0;
                   vga_b_reg<=0;
            end
             4'b0011:begin
                         vga_r_reg<=0;                          // all green
                   vga_g_reg<=8'hff;
                   vga_b_reg<=0;
```

```
            end
            4'b0100:begin
                        vga_r_reg<=0;                         // all blue
                    vga_g_reg<=0;
                    vga_b_reg<=8'hff;
                end
            4'b0101:begin
                     vga_r_reg<=grid_data_1[23:16];         // squre 1
                    vga_g_reg<=grid_data_1[15:8];
                    vga_b_reg<=grid_data_1[7:0];
            end
            4'b0110:begin
                        vga_r_reg<=grid_data_2[23:16];       // squre 2
                    vga_g_reg<=grid_data_2[15:8];
                    vga_b_reg<=grid_data_2[7:0];
                end
              4'b0111:begin
                     vga_r_reg<=x_cnt[12:5];                // horizontal gradient
                    vga_g_reg<=x_cnt[12:5];
                    vga_b_reg<=x_cnt[12:5];
                end
              4'b1000:begin
                        vga_r_reg<=y_cnt[10:3];               // vertical gradient
                    vga_g_reg<=y_cnt[10:3];
                    vga_b_reg<=y_cnt[10:3];
                end

              4'b1001:begin
                    vga_r_reg<=x_cnt[12:5];                // red horizontal gradient
                    vga_g_reg<=0;
                    vga_b_reg<=0;
                end
              4'b1010:begin
                        vga_r_reg<=0;                         //green horizontal gradient
                    vga_g_reg<=x_cnt[12:5];
                    vga_b_reg<=0;
                end
              4'b1011:begin
                        vga_r_reg<=0;                         //blue horizontal gradient
                    vga_g_reg<=0;
                    vga_b_reg<=x_cnt[12:5];
                end
              4'b1100:begin
                        vga_r_reg<=bar_data[23:16];               //colorful strips
```

```
                    vga_g_reg<=bar_data[15:8];
                    vga_b_reg<=bar_data[7:0];
               end
             default:begin
                        vga_r_reg<=8'hff;                    // all white
                    vga_g_reg<=8'hff;
                    vga_b_reg<=8'hff;
               end
          endcase;


   assign vga_hs = hsync_r;
   assign vga_vs = vsync_r;
   assign vga_r = (hsync_de & vsync_de)?vga_r_reg:8'h00;
   assign vga_g = (hsync_de & vsync_de)?vga_g_reg:8'b00;
   assign vga_b = (hsync_de & vsync_de)?vga_b_reg:8'h00;
```

(2) Main part of register configuration

Directly use the above experimental content for IIC interface configuration register. Here is mainly about the register configuration part

```
case( i )


          0:
           if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
           else begin isStart <= 2'b01; rData <= 8'h50; rAddr <= 8'h41; end

          1:
           if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
           else begin isStart <= 2'b01; rData <= 8'h10; rAddr <= 8'h41; end

          2:
           if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
           else begin isStart <= 2'b01; rData <= 8'h03; rAddr <= 8'h98; end

          3:
           if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
           else begin isStart <= 2'b01; rData <= 8'h03; rAddr <= 8'h9a; end

          4:
           if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
           else begin isStart <= 2'b01; rData <= 8'h30; rAddr <= 8'h9c; end
          5:
           if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
           else begin isStart <= 2'b01; rData <= 8'h01; rAddr <= 8'h9d; end
```

```
6:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rData <= 8'ha4; rAddr <= 8'ha2; end
7:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rData <= 8'ha4; rAddr <= 8'ha3; end

8:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rData <= 8'hd0; rAddr <= 8'he0; end

9:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rData <= 8'h00; rAddr <= 8'hf9; end
10:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rData <= 8'h20; rAddr <= 8'h15; end

11:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rData <= 8'h30; rAddr <= 8'h16; end

12:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rAddr <= 8'haf; rData <= 8'h02;end

13:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rAddr <= 8'h01; rData <= 8'h00;end

14:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rAddr <= 8'h02; rData <= 8'h18;end

15:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rAddr <= 8'h03; rData <= 8'h00;end

16:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
 else begin isStart <= 2'b01; rAddr <= 8'h0a; rData <= 8'h03;end

17:
 if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
```

```
        else begin isStart <= 2'b01; rAddr <= 8'h0b; rData <= 8'h6e;end

      18:
        if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
        else begin isStart <= 2'b01; rAddr <= 8'h0c; rData <= 8'hbd;end

      19:
        if( iic_done ) begin isStart <= 2'b00; i <= i + 1'b1; end
        else begin isStart <= 2'b01; rAddr <= 8'hd6; rData <= 8'hc0;end

    endcase
```

4.Download the Program to the Board to Test

(1) Lock the pins

Signal Name	Port Description	Network Label	FPGA Pin
clk_in	System clock, 50 MHz	C10_50MCLK	U22
rst_n	Reset, high by default	KEY1	M4
vga_hs	Horizontal synchronous signal	HDMI_HSYNC	C24
vga_vs	Vertical synchronous signal	HDMI_VSYNC	A25
en	Date valid	HDMI_DE	A24
vga_clk	Display clock	HDMI_CLK	B19
key1	Display effect toggle button	KEY2	L4
scl	ADV7511 configuration clock	I2C_SCL	R20
sda	ADV7511 configuration data	I2C_SDA	R21
vag_r[7]	Red output	HDMI_D23	F15
vag_r[6]	Red output	HDMI_D22	E16
vag_r[5]	Red output	HDMI_D21	D16
vag_r[4]	Red output	HDMI_D20	G17
vag_r[3]	Red output	HDMI_D19	E17
vag_r[2]	Red output	HDMI_D18	F17
vag_r[1]	Red output	HDMI_D17	C17
vag_r[0]	Red output	HDMI_D16	A17
vag_g[7]	Green output	HDMI_D15	B17
vag_g[6]	Green output	HDMI_D14	C18
vag_g[5]	Green output	HDMI_D13	A18
vag_g[4]	Green output	HDMI_D12	D19

vag_g[3]	Green output	HDMI_D11	D20
vag_g[2]	Green output	HDMI_D10	A19
vag_g[1]	Green output	HDMI_D9	B20
vag_g[0]	Green output	HDMI_D8	A20
vag_b[7]	Blue output	HDMI_D7	B21
vag_b[6]	Blue output	HDMI_D6	C21
vag_b[5]	Blue output	HDMI_D5	A22
vag_b[4]	Blue output	HDMI_D4	B22
vag_b[3]	Blue output	HDMI_D3	C22
vag_b[2]	Blue output	HDMI_D2	A23
vag_b[1]	Blue output	HDMI_D1	D21
vag_b[0]	Blue output	HDMI_D0	B24

(2) Comprehensive compilation and downloading the program to the board. Each time you press the **UP** button on the development board, you can see the different display effects on the computer monitor to switch. The effect is as follows:

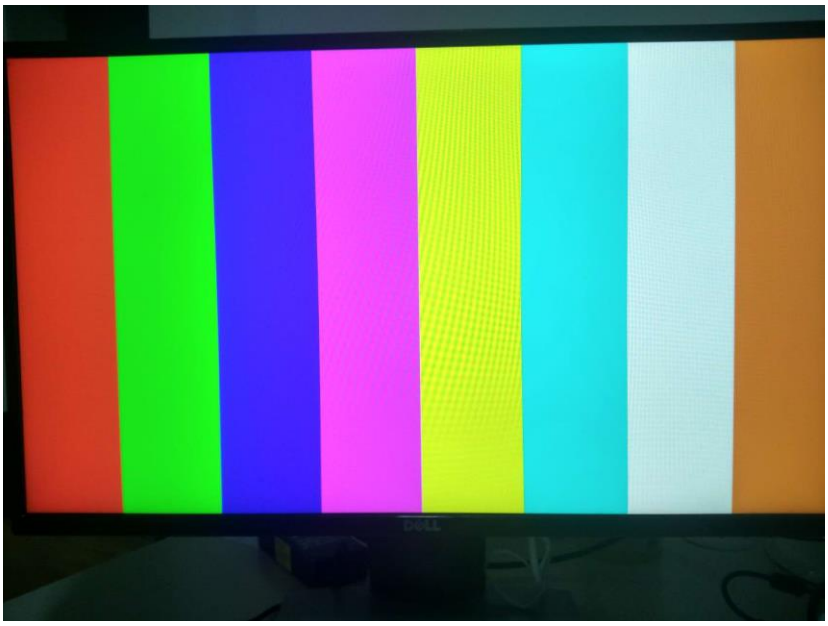


Fig 14. 3 HDMI display

References

(1) http://web.engr.oregonstate.edu/~traylor/ece474/beamer_lectures/verilog_operators.pdf

(2) https://www.utdallas.edu/~akshay.sridharan/index_files/Page5212.htm

(3) https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug908-vivado-programming-debugging.pdf

(4) https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug835-vivado-tcl-commands.pdf#nameddest=xwrite_cfgmem