EE332 Lab2 Exercise4

仇琨元 / Qiu Kunyuan
Dept. Electric and Electronic Engineering of SUSTech
11913019@mail.sustech.edu.cn

I.Project Introduction

The purpose of this lab is to design and realize a three-digit decimal counter. The design has some important functions, including an asynchronous reset function, a synchronous load function to set a start number of the counting, the function of increasing 1 for every 1 second and function of displaying the counting results on the 12 LEDs. The asynchronous reset function is realized by using a push button to initialize the counter to 000. The synchronous load function is realized by using a switch to turn on or turn off the load mode and use 12 switches to set the start number.

II.System Design

II.A.Prelab Exercise

The FPGA board provides only a 100MHz clock source, which is 10^8 times faster than the required clock. So I have designed a component of a frequency divider that slow down the output by 10^8 times for a clock input. The conceptual diagram of the frequency divider is in 图 1:

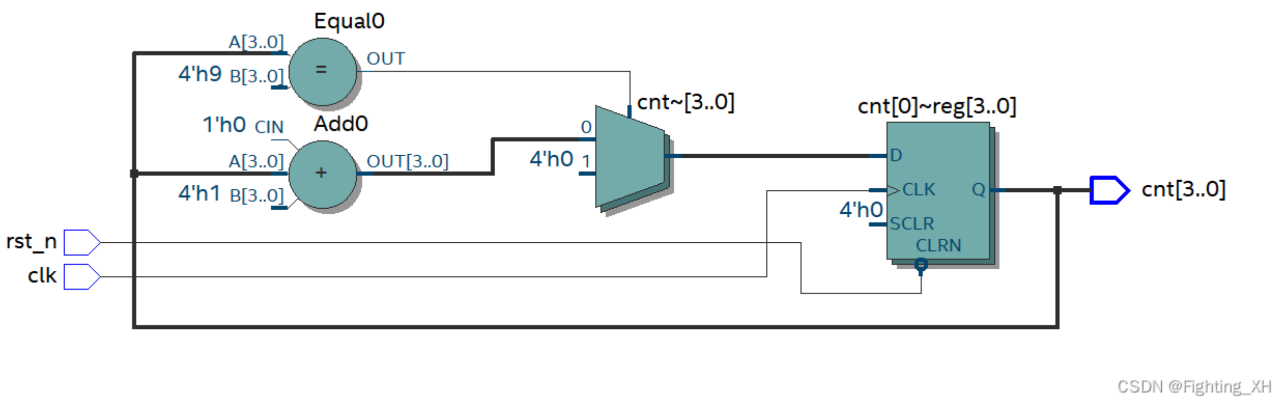

图 1 Conceptual Diagram of Counter based Divider

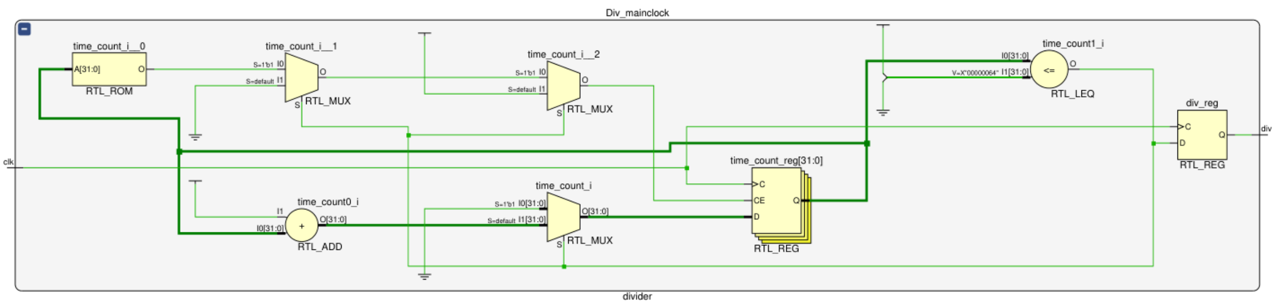

图 2 Elaborated Schematic of Divider

The component of frequency divider has one input `clk` and one output `div`. Because the clock source of FPGA board is 100MHz and what we need is the clock of 1Hz, the upper limit of the counter is set to the frequency of the main clock, which is `05F5_E100` in hex.

The behavioral simulation results indicated that the uncertain state at the powering up will not vanish spontaneously with the assign inside the block, rather than staying at such unstability. Therefore, an exterior or interior boot time reset operation is mandatory. To avoid of using exterior reset signal, an synthesizable `initial` block is introduced to ensure the initial state of the counter register bank.

The HDL code is provided as Code 1

II.B.Lab Exercise

The top level structure of the overall design is shown as 图 3:

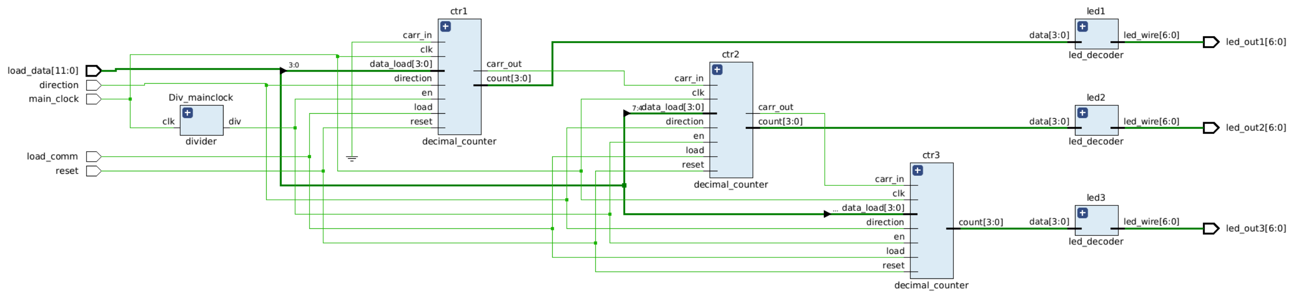

图 3　Top-level Schematics of the Design

While the 4-digit counter unit inside the top-level block are implemented with the following additional functionalities:

- Synchronous loading and resetting
- Asynchronous enable/hold line
- Reverse counting (up and down counting)
- Parameterized lower and upper bounds
- Full carriage and borrowing support

These features are enabled by expanding the combinational logic part of the counter unit with the corresponding arithmetic logics. For the loading and resetting features, the loaded data or zeros are directly connected to the register bank. For the reverse counting feature, there is one selection line that controls whether the registers are added or minus with 1 at each positive clock edge. For the carriage and borrowing supports, the carriage or borrowing input digit is summed up with the add/minus logic with respection to the current add or minus counting status.

The full elaborated schematic is shown as 图 4.

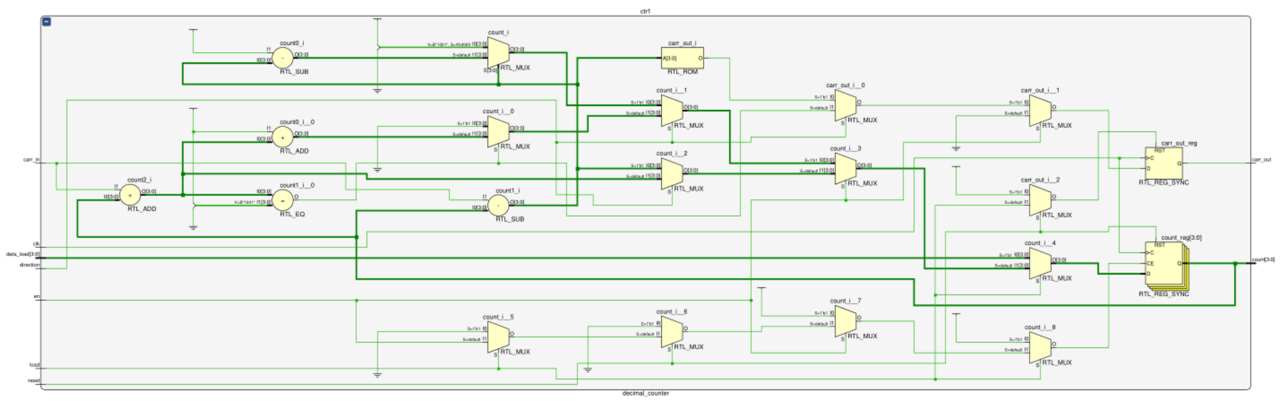

图 4　Elaborated Schematics of the Counter Unit

The HDL code of the counter unit is provided below:

III.Experiment Results

The following testbench are used to provide simulation stimulus. Since the output under 0.1MHz main clock frequency and the output under 1Hz main clock frequency should be identical for the same digital circuit, the simulation frequency is selected as 0.1MHz to minimize computational burdens.

III.A.Pre-Synthesis Behavioral Simulation

The results of behavioral simulation with increasing 1 at frequency 0.1MHz is shown as

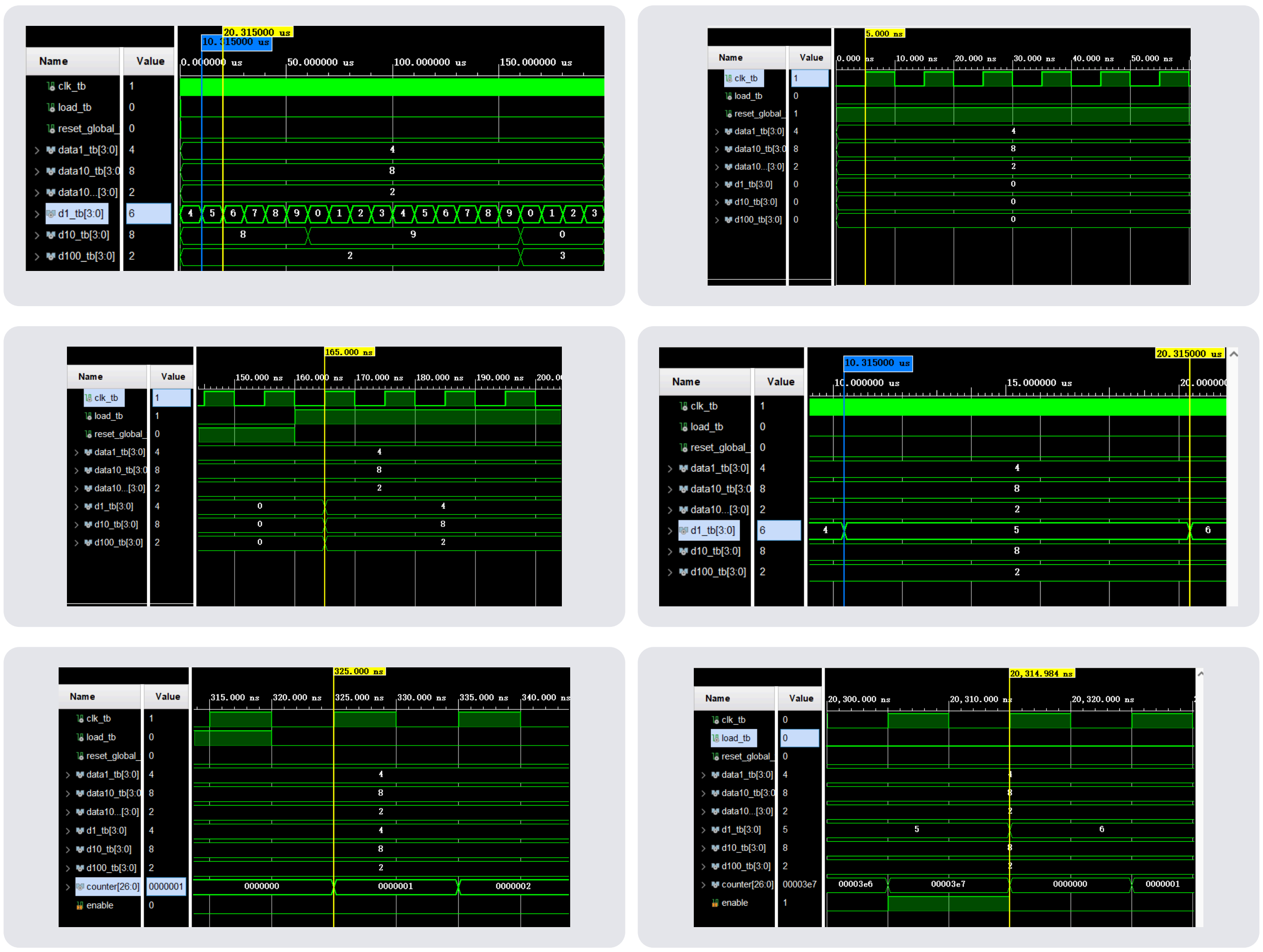

图 5 Details of Behavioral simulation result with increasing 1 by 10µs

From the details of Behavioral simulation result with increasing 1 by 10 µs, the result of simulation is in line with our expectation. When the logic value of `reset` is 1, the counter is initialized to `000`, which satisfy asynchronous reset function. The load number are loaded when both of `clk` and `load` are 1, which satisfy synchronous load function. The counter of frequency divider start counting at the first rising edge of `clk` after the logic value of `load` is 0. The whole decimal counter increase by 1 after 10 µs. What's more, the duration of high level of `enable` is the period of `clk`, which ensures the whole decimal counter increase by 1 after 10 µs.

Then we test the simulation when the value in the corresponding position is more than 9. The number which will loaded is `0010 1100 0100`. The result of simulation is as below:

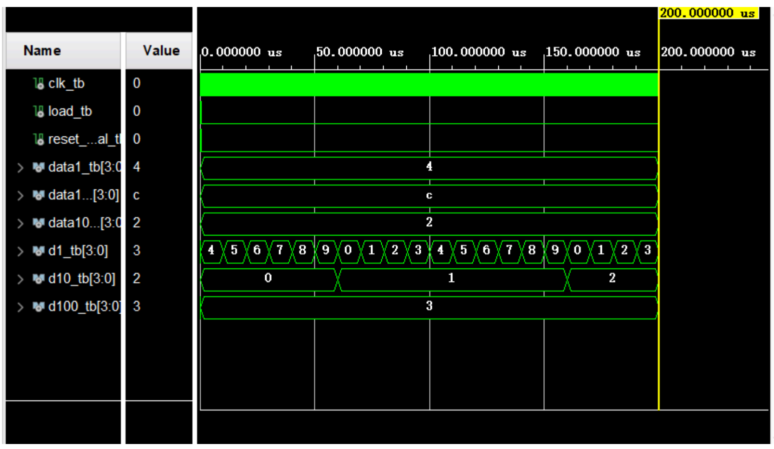

图 6 Behavioral simulation result for the value in the corresponding position is more than 9

The loaded number is `304`, which is in line with our expectation.

III.B.Post-Synthesis Functional Simulation

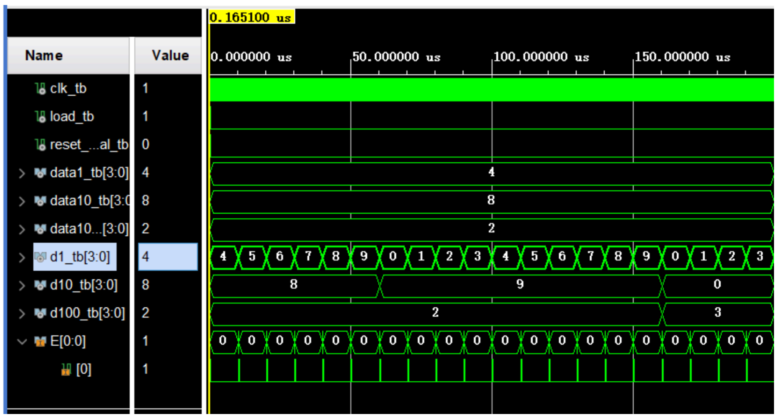

图 7　Post-Synthesis functional simulation result

The post-synthesis functional simulation is also in line with our design and the result of behavioral simulation. However, the result of post-synthesis timing simulation is a little different from that of behavioral simulation.

It is obviously that there exists a delta delay in the beginning, which is different from behavioral simulation and postsynthesis functional simulation. The value of delta delay is 1.340 ns. What's more, the numbers of input are not loaded until about 4.303 ns after the rising edge of `load`. The reason behind this is that although multiple signal assignment statements are executed concurrently in simulated time and are referred to as concurrent signal assignment statements, a small delay——delta delay will be automatically set in order to make a logical sequence of signals transmission.

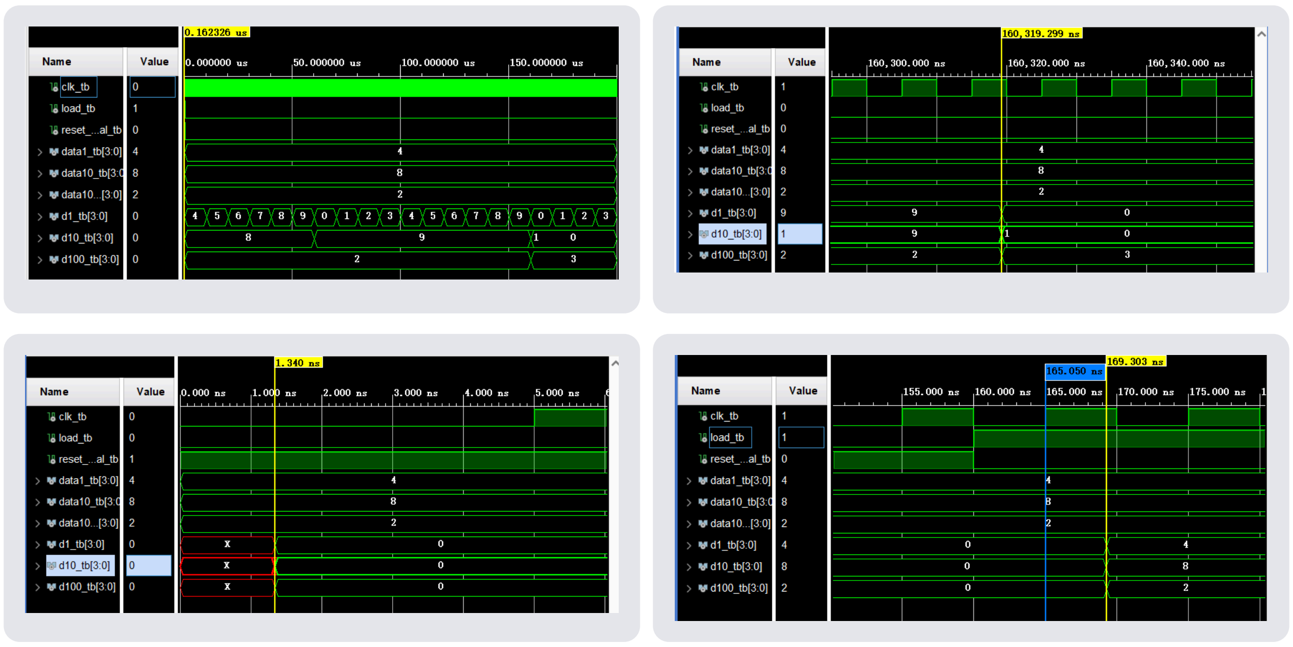

图 8　Post-implementation functional simulation result

As we can see, there is instantaneous change in `d10_tb`. One possible reason is that adding 1 in the combinational circuit when the output is `'d299`(decimal), `d1_tb` and `d10_tb` should change from 9 to 0. The binary code for 9 is 1001. The signal that changes the highest bit from 1 to 0 arrives before the signal that changes the lowest bit from 1 to 0. When the highest bit changes from 1 to 0, the value of the lowest bit is still 1 in that very short period of time. That's why there is instantaneous change in `d10_tb`.

III.C.Post-Synthesis Timing Simulation

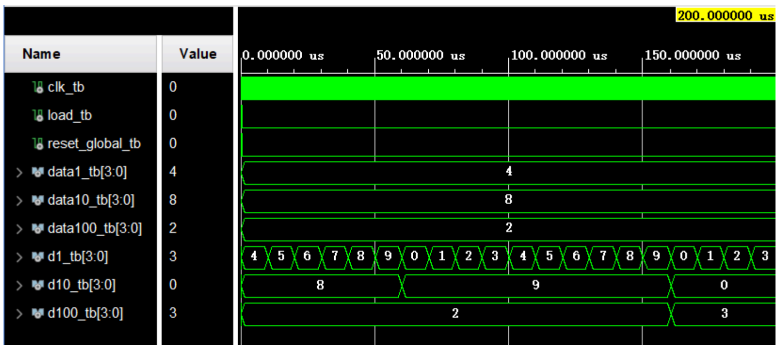

图 9　Post-Synthesis timing simulation result

The post-implementation functional simulation is also in line with our design and the result of behavioral simulation.

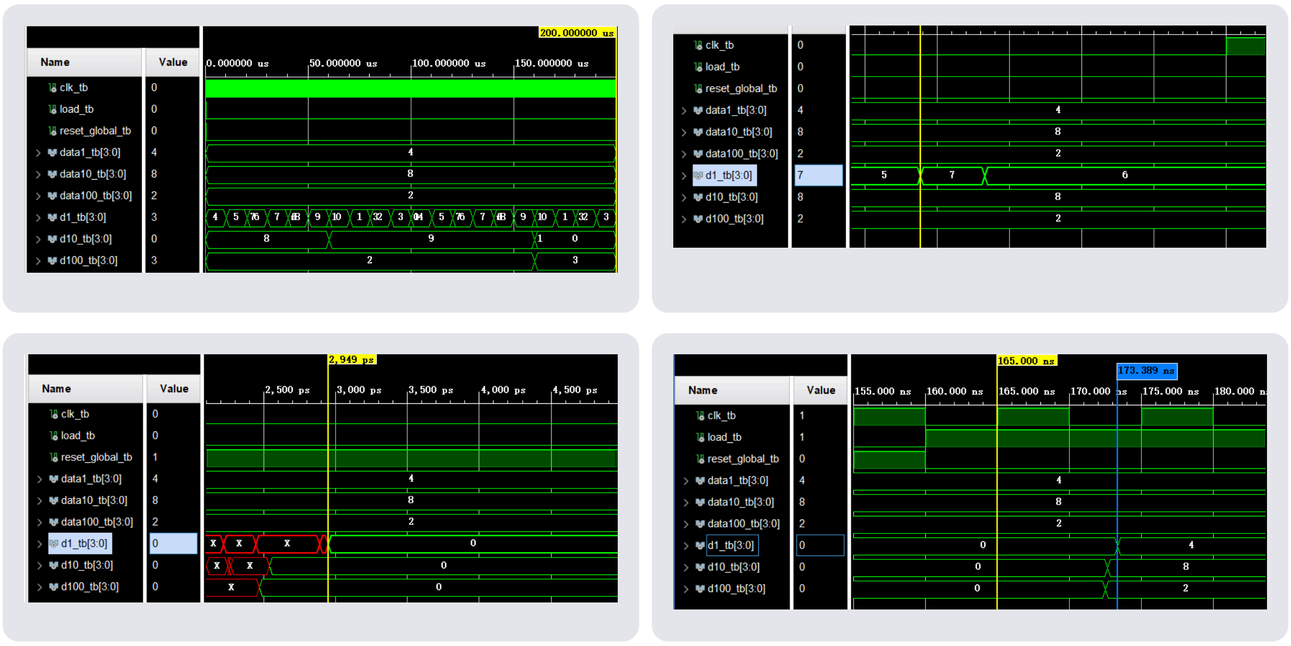

图 10　Post-implementtation timing simulation result

It is obviously that the delays at the beginning of `d1_tb`, `d10_tb` and `d100_tb` are different, which are almost twice as that in post-implementation timing simulation. What's more, the number of input `data1_tb` are not loaded until about 8.389ns after the rising edge of `load`, which are also almost twice as that in post-implementation timing simulation. That's due to layout, routing and different time delay of gate circuits in post-implementation timing simulation.

The schematic is very similar to our block diagram, which verify our code's correctness.

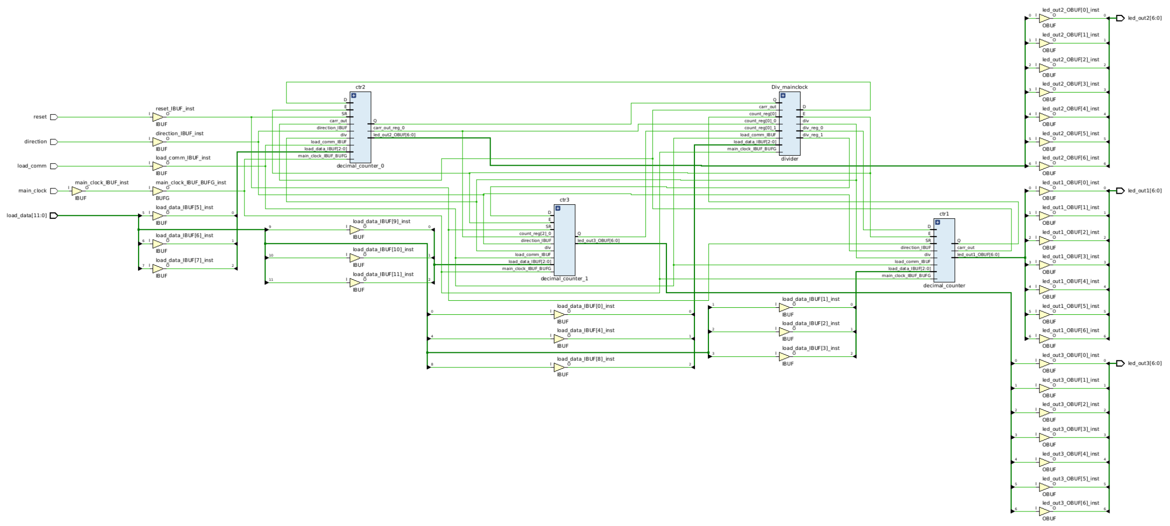

图 11　Post-Synthesis Schematic

III.D.Onboarding

In order to burn the program to FPGA, we need to write constraints file. The code for constraints file is shown in XDC file Code 5.

The corresponding layout of the hardware pins are shown below:

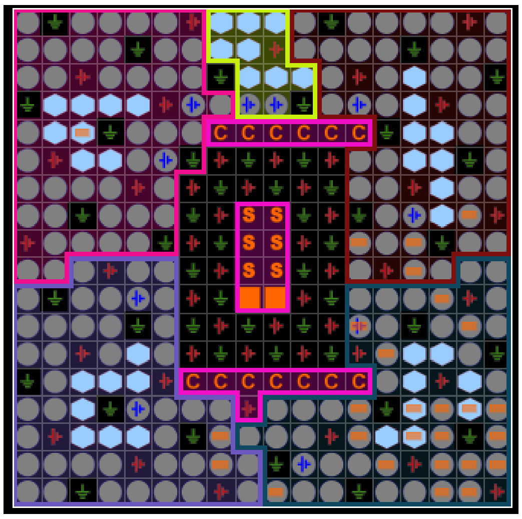

图 12 Post-Synthesis timing simulation result

And finally, we set the frequency to 1Hz and generate the bitstream and program to the board, the decimal counter result is correct.

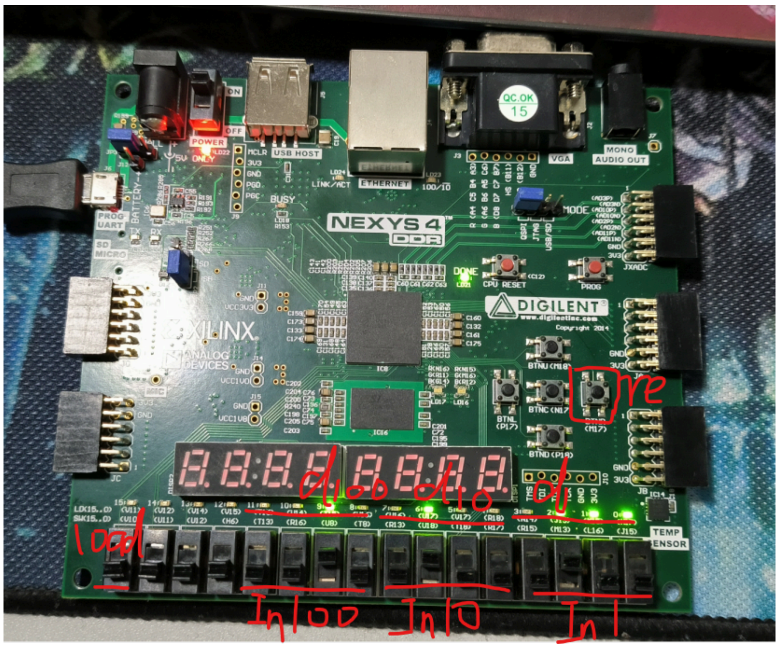

图 13 Post-Synthesis timing simulation result

IV.Conclusion and Analysis

The main part of this report is the process of digital design of three-digit decimal counter using VIVADO. There are some important points that need to be stressed again:

1. Synchronization means under the excitation of clock signal, in the rising or falling edge of the clock, the trigger carries out the corresponding operation.
2. Asynchrony means that the trigger performs corresponding operation at the rising or falling edge of the signal under the excitation signal.
3. The first segment of two-segment coding style is for a synchronous section or a synchronous section with asynchronous inputs. The first segment of two-segment coding style is for a combinational section.

V.Appendix / Code

V.A.Divider

```
module divider (
    input  wire clk,
    output reg  div
);

  parameter integer MAIN_FREQ = 32'd50_000;
  parameter integer PULSE_WIDTH = 100;

  reg [31:0] time_count = 32'd0;

  always @(posedge clk) begin
    if (time_count == (MAIN_FREQ-PULSE_WIDTH)) begin
      div <= 1'b1;
    end else if (time_count == MAIN_FREQ) begin
      div <= 1'b0;
      time_count <= 32'd0;
    end else begin
      time_count <= time_count + 32'd1;
      div <= div;
    end
  end

endmodule
```

Code 1 Divider

```
`timescale 1ns / 1ns

module decimal_counter (
    input             clk,        // 时钟信号
    input             reset,      // 异步复位信号
    input             en,         // 异步使能信号
    input             direction,  // 计数方向控制信号
    input             load,       // 同步置位使能信号
    input       [3:0] data_load,  // 同步置位数据输入
    input             carr_in,    // 进位/借位输入信号
    output reg [3:0] count,       // 计数器当前值
    output reg        carr_out    // 进位/借位输出信号
);

  // 参数定义
  parameter integer MAX_COUNT = 4'd9;  // 最大计数值
  parameter integer MIN_COUNT = 4'd0;  // 最小计数值

  initial begin
    count = MIN_COUNT; // 自启动条件
    carr_out = 0;
  end

  always @(posedge clk) begin
    if (reset) begin
      count <= MIN_COUNT;
      carr_out <= 1'b0;
    end else if (load) begin
      count <= data_load;
      carr_out <= 1'b0;
    end else if (en) begin
      if (direction) begin
        count = count - carr_in;  // 借位输入
        if (count == MIN_COUNT) begin
          count <= MAX_COUNT;
          carr_out <= 1'b1;
        end else begin
          count <= count - 4'b0001;
          carr_out <= 1'b0;
        end
      end else begin
        count = count + carr_in;  // 进位输入
        if (count == MAX_COUNT) begin
          count <= MIN_COUNT;
          carr_out <= 1'b1;
        end else begin
          count <= count + 4'b0001;
          carr_out <= 1'b0;
        end
      end
    end else begin
      carr_out <= 1'b0;
    end
  end

endmodule
```

Code 2 Counter Unit

V.C.LED Decoder

```
`timescale 1ns / 1ns

module led_decoder (
    data,
    led_wire
);

  input [3:0] data;
  output reg [6:0] led_wire;

  initial begin
    led_wire = 7'b0;
  end

  always @(*) begin
    case (data)
      4'h0: begin
        led_wire = 7'h01;
      end
      4'h1: begin
        led_wire = 7'h4f;
      end
      4'h2: begin
        led_wire = 7'h12;
      end
      4'h3: begin
        led_wire = 7'h06;
      end
      4'h4: begin
        led_wire = 7'h4c;
      end
      4'h5: begin
        led_wire = 7'h24;
      end
      4'h6: begin
        led_wire = 7'h20;
      end
      4'h7: begin
        led_wire = 7'h0f;
      end
      4'h8: begin
        led_wire = 7'h00;
      end
      4'h9: begin
        led_wire = 7'h04;
      end
      4'ha: begin
        led_wire = 7'h08;
      end
      4'hb: begin
        led_wire = 7'h60;
      end
      4'hc: begin
        led_wire = 7'h31;
      end
      4'hd: begin
        led_wire = 7'h42;
      end
      4'he: begin
        led_wire = 7'h30;
      end
      4'hf: begin
        led_wire = 7'h38;
      end
      default: begin
        led_wire = 7'h7e;
      end
    endcase
  end


endmodule
```
Code 3 Divider

V.D.Testbench

```
`timescale 1ns / 1ns


module tb_all();

reg main_clock, rst, enable, direction, load, carr_in;
reg [11:0] data_load;
wire [20:0] count;
wire carr_out, divided_clock;
integer t_var, loop_var;

parameter integer MAIN_CLK=50;
parameter integer CTR_CLK_MULT=5;
parameter integer PULSE_WIDTH=50;
parameter integer T_DIR_INV = 16;
parameter integer T_LOAD = 48;
parameter integer T_RESET = 60;
parameter integer T_LOAD2 = 64;

divider #(
  .MAIN_FREQ (250),
  .PULSE_WIDTH (50)
  ) div_0 (
  .clk (main_clock),
  .div (divided_clock)
);

counter_top ctr_test1 (
  .main_clock (main_clock),
  .reset (rst),
  .direction (direction),
  .load_comm (load),
  .load_data (data_load),
  .led_out1 (count[6:0]),
  .led_out2 (count[13:7]),
  .led_out3 (count[20:14])
);

initial begin
  t_var=32'b0;
  loop_var=32'd0;
  main_clock=0;
  rst=1;
  #50;
  enable=1;
  rst=0;
  direction=0;
  carr_in=0;
  #10;
  load=1;
  data_load=4'h0;
  #10;
  load=0;
end

always begin
  #(MAIN_CLK/2);
  main_clock=~main_clock;
end

always @(posedge divided_clock) begin
  t_var=t_var+32'd1;
  if(t_var>=T_DIR_INV) begin
    direction=1;
  end
  if(t_var==(T_DIR_INV+T_LOAD2)) begin
    direction=0;
  end
end

endmodule
```

Code 4 Testbench

V.E.XDC Constraints

```
set_property PACKAGE_PIN E3 [get_ports clk]
set_property PACKAGE_PIN M17 [get_ports reset_global]
set_property PACKAGE_PIN V10 [get_ports load]

set_property PACKAGE_PIN J15 [get_ports data1[0]]
set_property PACKAGE_PIN L16 [get_ports data1[1]]
set_property PACKAGE_PIN M13 [get_ports data1[2]]
set_property PACKAGE_PIN R15 [get_ports data1[3]]

set_property PACKAGE_PIN R17 [get_ports data10[0]]
set_property PACKAGE_PIN T18 [get_ports data10[1]]
set_property PACKAGE_PIN U18 [get_ports data10[2]]
set_property PACKAGE_PIN R13 [get_ports data10[3]]

set_property PACKAGE_PIN T8 [get_ports data100[0]]
set_property PACKAGE_PIN U8 [get_ports data100[1]]
set_property PACKAGE_PIN R16 [get_ports data100[2]]
set_property PACKAGE_PIN T13 [get_ports data100[3]]

set_property PACKAGE_PIN H17 [get_ports d1[0]]
set_property PACKAGE_PIN K15 [get_ports d1[1]]
set_property PACKAGE_PIN J13 [get_ports d1[2]]
set_property PACKAGE_PIN N14 [get_ports d1[3]]

set_property PACKAGE_PIN R18 [get_ports d10[0]]
set_property PACKAGE_PIN V17 [get_ports d10[1]]
set_property PACKAGE_PIN U17 [get_ports d10[2]]
set_property PACKAGE_PIN U16 [get_ports d10[3]]

set_property PACKAGE_PIN V16 [get_ports d100[0]]
set_property PACKAGE_PIN T15 [get_ports d100[1]]
set_property PACKAGE_PIN U14 [get_ports d100[2]]
set_property PACKAGE_PIN T16 [get_ports d100[3]]
```

Code 5 XDC