实验报告

实验题目：Lab9　　　日期：2018 年 11 月 23 日

姓名:刘紫檀　　　学号:PB17000232　　成绩:____________

实验目的：

1-1 采用门级建模方式设计 1bi 全加器，定义门电路模型时采用参数化声明方式将基本门（与、或、非）延时设为 2 个时间单位（#2），全加器模块调用门电路时，传入新的延时参数：反相器（#1）、与门（#3）、或门（#3），编写仿真文件进行仿真——2 分

2-1 设计 8bit 增/减计数器，通过设置综合属性，不使用 DSP48 单元，下载到开发板——代码 2 分，下载 2 分

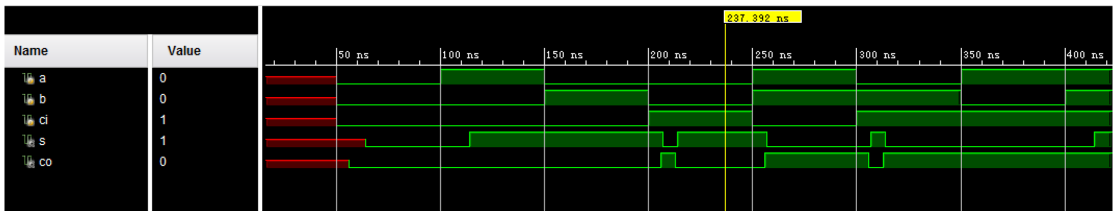
3-1 设计一精确到 0.1s 的计时器，在七段数码管上以 M.SS.f 的格式显示，带有复位和使能功能——代码 2 分，下载 2 分

截图：

```
`timescale 1ns / 1ps
//1-1 全加器代码

module fulladder #(parameter INV_DELAY = 2, AND_DELAY = 2, OR_DELAY = 2) (
    input a,
    input b,
    input ci,
    output s,
    output co
    );
    //s=a xor b xor ci
    wire a_xor_b;
    xor #(INV_DELAY+AND_DELAY+OR_DELAY) (a_xor_b, a, b);
    xor #(INV_DELAY+AND_DELAY+OR_DELAY) (s, a_xor_b, ci);
    //co=ab+(a xor b)ci
    wire a_b;
    wire a_xor_b_ci;
    and #(AND_DELAY) (a_b, a, b);
    and #(AND_DELAY) (a_xor_b_ci, a_xor_b, ci);
    or #(OR_DELAY) (co, a_b, a_xor_b_ci);

endmodule
```

```
`timescale 1ns / 1ps
//仿真文件

module tb();
    reg a;
    reg b;
    reg ci;
    wire s;
    wire co;

    fulladder #(1, 3, 3) DUT(a, b, ci, s, co);
    initial begin
    #50 a = 0; b = 0; ci = 0;
    #50 a = 1; b = 0; ci = 0;
    #50 a = 0; b = 1; ci = 0;
    #50 a = 0; b = 0; ci = 1;
    #50 a = 1; b = 1; ci = 0;
    #50 a = 0; b = 1; ci = 1;
    #50 a = 1; b = 0; ci = 1;
    #50 a = 1; b = 1; ci = 1;
   end
endmodule
```

实验 1-1 仿真截图



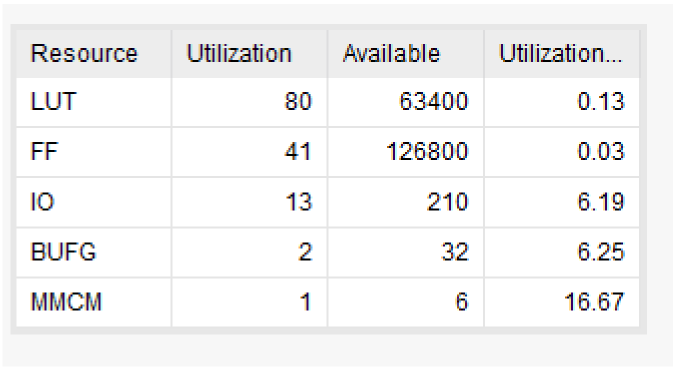
Name	Value
a	0
b	0
ci	1
s	1
co	0

```
//实验 2-1 代码
`timescale 1ns / 1ps
 (* use_dsp48 = "no" *)
module bcd_decoder(
    input [3:0] x,
    output reg [6:0] seg,
    output reg [6:0] seg_hi
    );

always @ (*)
begin
    case (x)
        4'd0: begin seg = 7'b1000000; seg_hi = 7'b1000000; end
        4'd1: begin seg = 7'b1111001; seg_hi = 7'b1000000; end
        4'd2: begin seg = 7'b0100100; seg_hi = 7'b1000000; end
        4'd3: begin seg = 7'b0110000; seg_hi = 7'b1000000; end
        4'd4: begin seg = 7'b0011001; seg_hi = 7'b1000000; end
        4'd5: begin seg = 7'b0010010; seg_hi = 7'b1000000; end
        4'd6: begin seg = 7'b0000010; seg_hi = 7'b1000000; end
        4'd7: begin seg = 7'b1111000; seg_hi = 7'b1000000; end
        4'd8: begin seg = 7'b0000000; seg_hi = 7'b1000000; end
        4'd9: begin seg = 7'b0010000; seg_hi = 7'b1000000; end
        4'd10: begin seg = 7'b1000000; seg_hi = 7'b1111001; end
        4'd11: begin seg = 7'b1111001; seg_hi = 7'b1111001; end
        4'd12: begin seg = 7'b0100100; seg_hi = 7'b1111001; end
        4'd13: begin seg = 7'b0110000; seg_hi = 7'b1111001; end
        4'd14: begin seg = 7'b0011001; seg_hi = 7'b1111001; end
        4'd15: begin seg = 7'b0010010; seg_hi = 7'b1111001; end
    endcase

end
endmodule


(* use_dsp48 = "no" *)
module main(
    input clk_in1,
    input reset,
    input enable,
    input dec,
    output locked,
    output [7:0] led
    );
    wire sig;  //flush signal
    wire sig_inc;
    wire counter_carry;

    wire clkabc;   //divided signal
    clk_wiz_0 clk_module (
         .clk_out1(clkabc),
         .reset(0),
         .locked(locked),
         .clk_in1(clk_in1));
    sec_gen s_gen(.clk_out1(clkabc), .sig(sig_inc), .enable(enable), .reset(0));
    c_counter_binary_0 counter (
      .CLK(sig_inc),          // input wire CLK
      .CE(enable),            // input wire
      .SCLR(reset),
      .UP(dec),
      .Q(led)              // output wire [3 : 0] Q
    );

endmodule
(* use_dsp48 = "no" *)
module sec_gen(
    input clk_out1,
    input enable,
    output reg sig,
    input reset
    );
reg [31:0] count;  //Can't be in procedural statements

always @ (posedge clk_out1 or posedge reset)
begin
    if (reset) begin
        sig <= 1'b0;
        count <= 32'd2500000;
    end
    else if (enable) begin
    //NOTE: if (enable) begin  WILL CAUSE BUG on reset=enable=1, causing count
undefined
        if (count == 0) begin
            sig <= ~sig;
            count <= 32'd2500000;
        end else count <= count - 1;
    end
end
endmodule


module c_counter_binary_0 #(parameter COUNT_SIZE = 233) (input CLK, input CE, input
SCLR, input UP, output reg [7:0] Q);
    always @ (posedge CLK) begin
        if (SCLR)
```

```
            Q <= 8'b00000000;
        else begin
            if (CE) begin
                if (Q >= COUNT_SIZE)
                    Q <=  8'b00000000;
                else if (Q == 0 && DOWN)
                    Q <= COUNT_SIZE;
                else begin
                    if (UP) begin
                        Q <= Q + 1;
                    end else begin
                        Q <= Q - 1;
                    end
                end
            end
        end
    end
endmodule
```

实验 2-1 下载照片

实验 2-1 Summary



Resource	Utilization	Available	Utilization...
LUT	80	63400	0.13
FF	41	126800	0.03
IO	13	210	6.19
BUFG	2	32	6.25
MMCM	1	6	16.67

Number of BUFG/BUFGCTRL: 2

Number of Slice LUTs used: 80

Number of FF used: 41

Number of DSP48E1 slices used: 0

Number of IOs used: 13

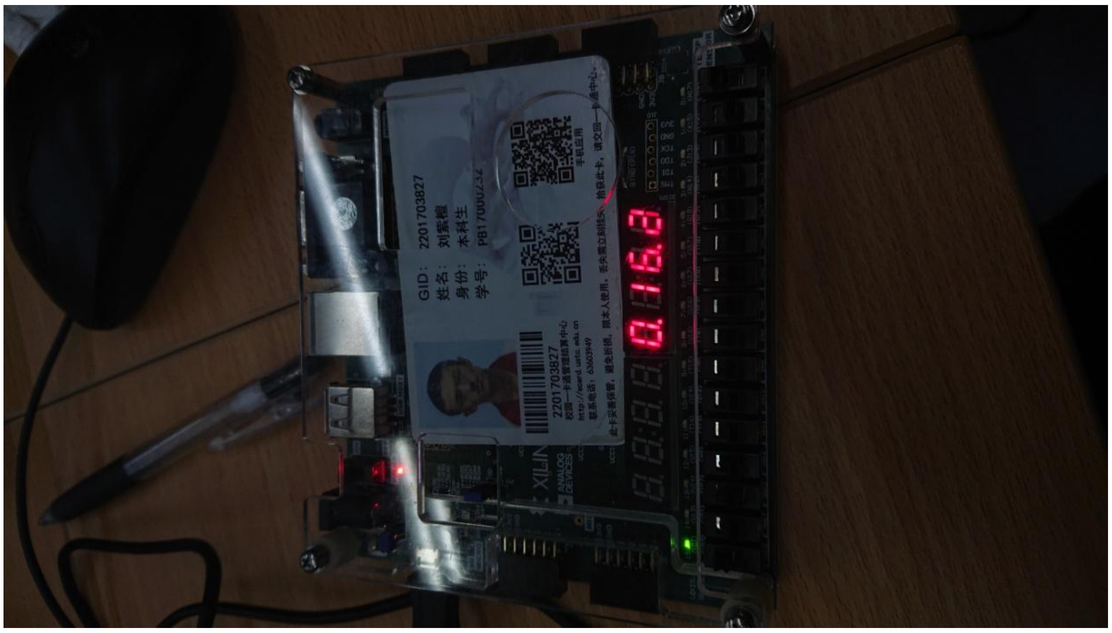
实验 3-1

```
//代码
`timescale 1ns / 1ps
module bcd_decoder(
    input [3:0] x,
    output reg [6:0] seg,
    output reg [6:0] seg_hi
    );

always @ (*)
begin
    case (x)
        4'd0: begin seg = 7'b1000000; seg_hi = 7'b1000000; end
        4'd1: begin seg = 7'b1111001; seg_hi = 7'b1000000; end
        4'd2: begin seg = 7'b0100100; seg_hi = 7'b1000000; end
        4'd3: begin seg = 7'b0110000; seg_hi = 7'b1000000; end
        4'd4: begin seg = 7'b0011001; seg_hi = 7'b1000000; end
        4'd5: begin seg = 7'b0010010; seg_hi = 7'b1000000; end
        4'd6: begin seg = 7'b0000010; seg_hi = 7'b1000000; end
        4'd7: begin seg = 7'b1111000; seg_hi = 7'b1000000; end
        4'd8: begin seg = 7'b0000000; seg_hi = 7'b1000000; end
        4'd9: begin seg = 7'b0010000; seg_hi = 7'b1000000; end
        4'd10: begin seg = 7'b1000000; seg_hi = 7'b1111001; end
        4'd11: begin seg = 7'b1111001; seg_hi = 7'b1111001; end
        4'd12: begin seg = 7'b0100100; seg_hi = 7'b1111001; end
        4'd13: begin seg = 7'b0110000; seg_hi = 7'b1111001; end
        4'd14: begin seg = 7'b0011001; seg_hi = 7'b1111001; end
        4'd15: begin seg = 7'b0010010; seg_hi = 7'b1111001; end
    endcase

end
endmodule


module carry_ff(input clk_div, input counter_carry, output reg carry_pulse);
    //Negedge of counter_carry detector
    //Skew and delay: 1 cycle (1/5000000 sec.)
    reg cc_prev;
    always @ (posedge clk_div) begin
       if (carry_pulse == 0 && cc_prev == 1)
           carry_pulse <= 1;
           cc_prev <= counter_carry;
       if (cc_prev == counter_carry)
           carry_pulse <= 0;
    end
endmodule


module main(
    input clk_in1,
    input reset,
    input enable,
    output locked,
    output reg [6:0] seg,
    output reg [7:0] an,
    output reg digit_dot
    );
    wire [1:0] sig;  //flush signal
    wire sig_inc; //increment signal
    wire clk_div; //5MHz Clock signal


    wire [3:0] low_bcd;
    wire [3:0] hi_bcd;
    wire [3:0] six_bcd;
    wire [3:0] five_bcd;

    wire [6:0] seg_temp_low;
    wire [6:0] seg_temp_hi;
    wire [6:0] seg_temp_six;
    wire [6:0] seg_temp_five;
    bcd_decoder low_decode(low_bcd, seg_temp_low);
    bcd_decoder hi_decode(hi_bcd, seg_temp_hi);
    bcd_decoder six_decode(six_bcd, seg_temp_six);
    bcd_decoder five_decode(five_bcd, seg_temp_five);

    //flush signal generator
    flush_gen f_gen(.clk_div(clk_div), .sig(sig), .reset(0));

    clk_wiz_0 c_clk(
       .clk_out1(clk_div),     // output clk_out1
       .reset(0), // input reset
       .locked(locked),       // output locked
       .clk_in1(clk_in1));      // input clk_in1

     c_counter_pulsegen sig_gen (
         .CLK(clk_div),          // input wire CLK
         .CE(enable),            // input wire CE
         .THRESH0(sig_inc)  // output wire THRESH0
     );

    wire counter_carry;

    // .1s counter ( 0 ~ 9 )
    c_counter_dec low_counter (
      .CLK(clk_div),          // input wire CLK
      .CE(sig_inc),            // input wire CE
      .SCLR(reset),        // input wire SCLR
      .THRESH0(counter_carry),  // output wire THRESH0
      .Q(low_bcd)              // output wire [3 : 0] Q
    );

    wire carry_pulse_1;
    carry_ff c_ff_1
(.clk_div(clk_div),.counter_carry(counter_carry),.carry_pulse(carry_pulse_1));

    wire counter_carry_1;
    // 1s counter ( 0 ~ 9 )
   c_counter_dec hi_counter (
      .CLK(clk_div),          // input wire CLK
      .SCLR(reset),        // input wire SCLR
      .CE(carry_pulse_1),            // input wire CE
      .THRESH0(counter_carry_1),  // output wire THRESH0
      .Q(hi_bcd)              // output wire [3 : 0] Q
    );
    wire carry_pulse_2;
    carry_ff c_ff_2
(.clk_div(clk_div), .counter_carry(counter_carry_1), .carry_pulse(carry_pulse_2
));

    wire counter_carry_2;
    // 10s counter ( 0 ~ 9 )
   c_counter_six six_counter (
      .CLK(clk_div),          // input wire CLK
      .SCLR(reset),        // input wire SCLR
      .CE(carry_pulse_2),            // input wire CE
      .THRESH0(counter_carry_2),  // output wire THRESH0
      .Q(six_bcd)              // output wire [3 : 0] Q
    );

    wire carry_pulse_3;
    carry_ff c_ff_3
(.clk_div(clk_div), .counter_carry(counter_carry_2), .carry_pulse(carry_pulse_3
));

    wire counter_carry_3;
    // Min counter ( 0 ~ 4 )
   c_counter_five five_counter (
      .CLK(clk_div),          // input wire CLK
      .SCLR(reset),        // input wire SCLR
      .CE(carry_pulse_3),            // input wire CE
      .THRESH0(counter_carry_3),  // output wire THRESH0
      .Q(five_bcd)              // output wire [3 : 0] Q
    );


    always @ (clk_div) begin
        case (sig)
            2'b00: begin seg <= seg_temp_low;  an <= 8'b11111110; digit_dot <= 1;
end
            2'b01: begin seg <= seg_temp_hi; an <= 8'b11111101; digit_dot <= 0; end
            2'b10: begin seg <= seg_temp_six; an <= 8'b11111011; digit_dot <= 1;
end
            2'b11: begin seg <= seg_temp_five; an <= 8'b11110111; digit_dot <= 0;
end
        endcase
    end
```

```
endmodule

module flush_gen (input clk_div, input reset, output reg [1:0] sig);
    reg [31:0] count;
    always @ (posedge clk_div) begin
        if (reset)
            count <= 0;
        else if (count == 32'd5000) begin
                count <= 0;
                sig <= sig + 1;
             end else count <= count + 1;
    end
endmodule
```

实验 3-1 下载截图

（在刷新，快门没那么快，所以最后一位重影了）



实验总结：本次实验我熟练掌握了 IP Counter 的构建。