实验报告

实验题目：Lab 8　　日期：2018 年 11 月 16 日

姓名:刘紫檀 学号:PB17000232　 成绩:____________

实验目的：

——了解 IP 核例化和使用流程

——掌握时钟管理 IP 和计数器 IP 的使用

——学会自行设计计数器及生成脉冲信号

——掌握数码管动态扫描显示原理和具体实现

截图：

```
//实验 8-1-1 代码
`timescale 1ns / 1ps

module main_pulse(
    input clk_in1,
    input reset,
    input enable,
    output locked,
    output sig
    );
    wire clkabc;
    clk_wiz_0 clk_module (
      // Clock out ports
      .clk_out1(clkabc),     // output clk_out1
      // Status and control signals
      .reset(reset), // input reset
      .locked(locked),       // output locked
     // Clock in ports
      .clk_in1(clk_in1));      // input clk_in1
      pulse_gen
p_gen(.clk_out1(clkabc), .sig(sig), .enable(enable), .reset(reset));
endmodule
```

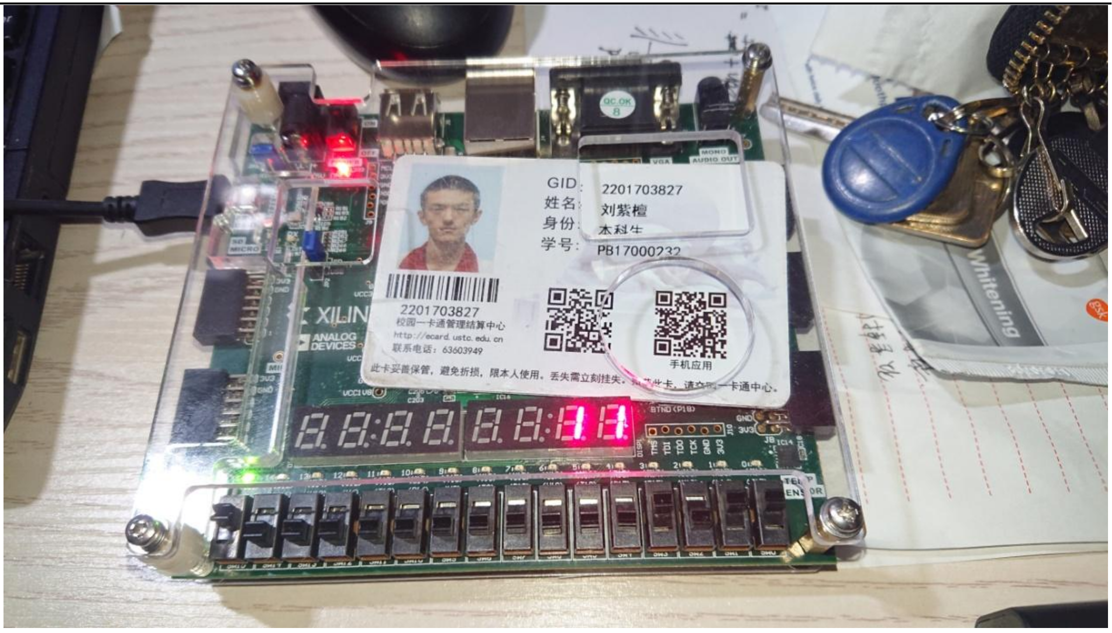
```
module pulse_gen(
    input clk_out1,
    input enable,
    output reg sig,
    input reset
    );
reg [31:0] count;  //Can't be in procedural statements

always @ (posedge clk_out1 or posedge reset)
begin
    if (reset) begin
        sig <= 1'b0;
        count <= 32'd5000000;
    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'd5000000;
        end else count <= count - 1;
    end
end
endmodule
```

实验 8-1-1 下载结果

```
//实验 8-1-2 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 main(
    input clk_in1,
    input reset,
    input enable,
    input [7:0] swt,
    output locked,
    output reg [6:0] seg,
    output reg [7:0] an
    );
    wire sig;
    wire [6:0] seg_temp_low;
    wire [6:0] seg_temp_hi;
    bcd_decoder( {swt[3], swt[2], swt[1], swt[0]}, seg_temp_low, seg_temp_hi);
    wire clkabc;
    clk_wiz_0 clk_module (
         // Clock out ports
         .clk_out1(clkabc),     // output clk_out1
         // Status and control signals
         .reset(reset), // input reset
         .locked(locked),       // output locked
        // Clock in ports
         .clk_in1(clk_in1));      // input clk_in1
    pulse_gen
p_gen(.clk_out1(clkabc), .sig(sig), .enable(enable), .reset(reset));

//NOTE: always @ (posedge sig or negedge sig) WILL CAUSE sig to be 1 all the time
//WHY??
always @ (clk_in1)
    begin
    if (sig) begin
        seg <= seg_temp_low;
        an <= 8'b11111110;
    end
    else begin
        seg <= seg_temp_hi;
        an <= 8'b11111101;
    end
end
endmodule

module pulse_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'd50000;
    end
    else if (enable) begin
    //NOTE: if (enable) begin  WILL CAUSE BUG on reset=enable=1, causing count
undefined
```

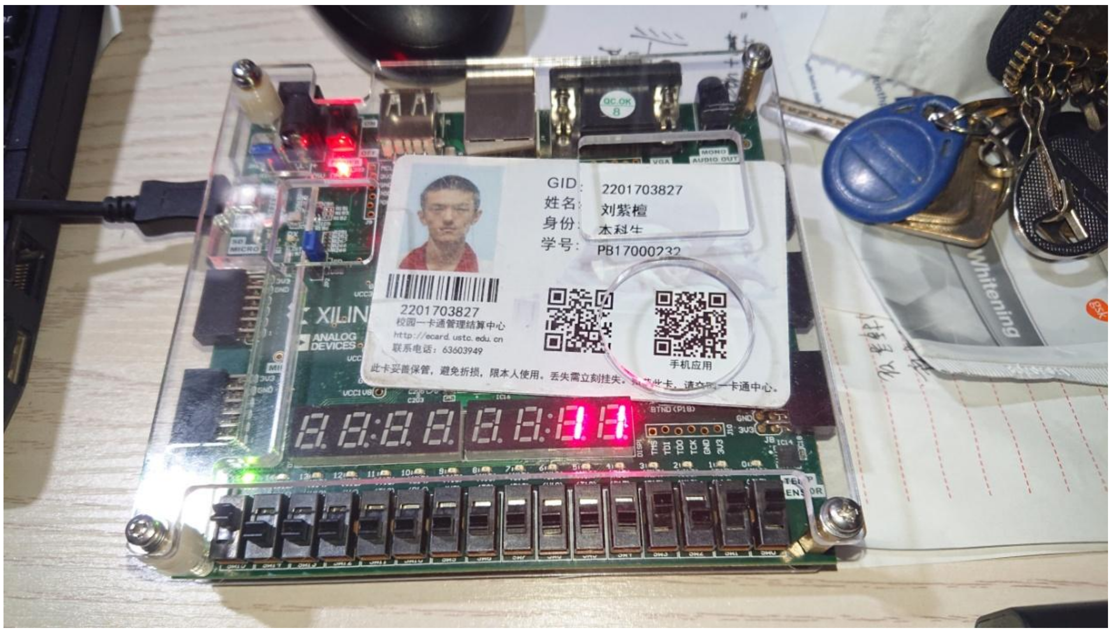
```
        if (count == 0) begin
            sig <= ~sig;
            count <= 32'd50000;
        end else count <= count - 1;
    end
end
endmodule
```



实验 8-1-2 下载结果

```
//实验 8-2-1 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 main(
    input clk_in1,
    input reset,
    input enable,
    output locked,
    output reg [6:0] seg,
    output reg [7:0] an
    );
    wire sig;  //flush signal
    wire sig_inc;
    wire counter_carry;
    wire [3:0] low_bcd;
    wire [3:0] hi_bcd;
    wire [6:0] seg_temp_low;
    wire [6:0] seg_temp_hi;
    bcd_decoder low_decode(low_bcd, seg_temp_low);
    bcd_decoder hi_decode(hi_bcd, seg_temp_hi);

    wire clkabc;   //divided signal
    clk_wiz_0 clk_module (
         .clk_out1(clkabc),
         .reset(0),
         .locked(locked),
         .clk_in1(clk_in1));
    pulse_gen p_gen(.clk_out1(clkabc), .sig(sig), .enable(1), .reset(0));
    sec_gen s_gen(.clk_out1(clkabc), .sig(sig_inc), .enable(enable), .reset(0));
    c_counter_binary_0 low_counter (
      .CLK(sig_inc),          // input wire CLK
      .CE(enable),            // input wire CE
      .SCLR(reset),        // input wire SCLR
      .THRESH0(counter_carry),  // output wire THRESH0
      .Q(low_bcd)              // output wire [3 : 0] Q
    );
   c_counter_binary_1 hi_counter (
      .CLK(sig_inc),          // input wire CLK
      .SCLR(reset),        // input wire SCLR
      .CE(counter_carry),            // input wire CE
      .Q(hi_bcd)              // output wire [3 : 0] Q
    );

//NOTE: always @ (posedge sig or negedge sig) WILL CAUSE sig to be 1 all the time
//WHY??
always @ (clk_in1)
    begin
    if (sig) begin
        seg <= seg_temp_low;
        an <= 8'b11111110;
    end
    else begin
        seg <= seg_temp_hi;
        an <= 8'b11111101;
    end
end
endmodule

module pulse_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'd50000;
    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'd50000;
        end else count <= count - 1;
    end
end
endmodule


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
```


实验 8-2-1 下载结果

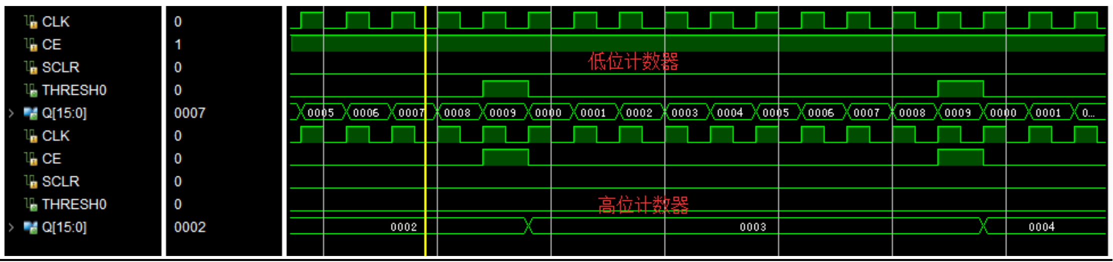
附加题：

1.对两位数的 BCD 计数器进行仿真，以确保功能正确

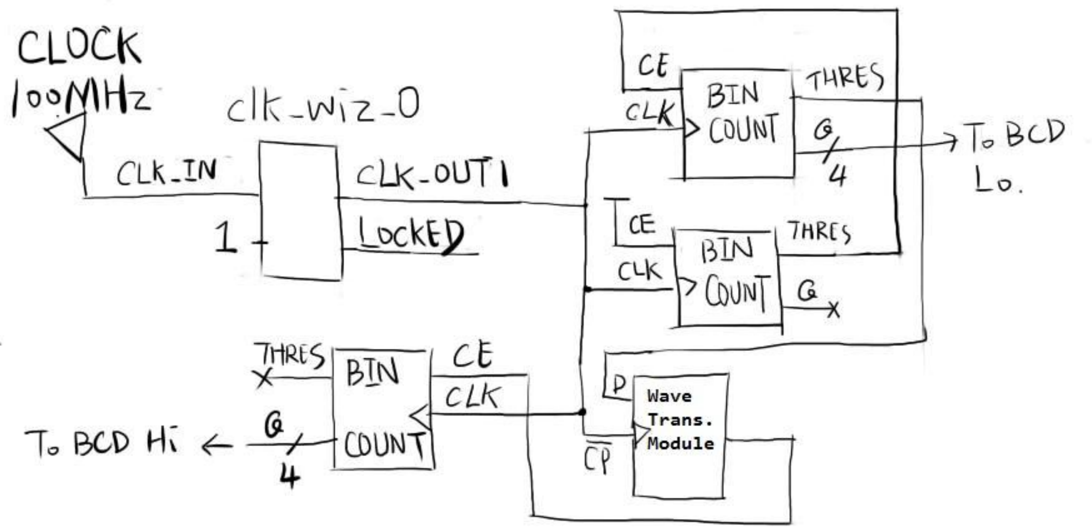
因为原时钟信号频率过高，直接写 100MHz CLK 的仿真的时间开销不可承受，所以将 module 单独提出来仿真，大概如下代码：

```
assign sig_inc = clk_in1;

    c_counter_binary_0 low_counter (
      .CLK(sig_inc),           // input wire CLK
      .CE(enable),             // input wire CE
      .SCLR(reset),         // input wire SCLR
      .THRESH0(counter_carry),   // output wire THRESH0
      .Q(low_bcd)              // output wire [3 : 0] Q
    );
   c_counter_binary_0 hi_counter (
      .CLK(sig_inc),           // input wire CLK
      .SCLR(reset),         // input wire SCLR
      .CE(counter_carry),             // input wire CE
      .Q(hi_bcd)             // output wire [3 : 0] Q
    );
```

利用原来的 testbench 可以得到如下结果：



2.如果 2.1 实验中十进制计数器使用 5MHz 时钟控制，用 1Hz 周期脉冲信号（每 5M 周期一个高电平）作为计数器时钟使能信号（CE），应如何设计？试画出电路图，并编写代码，下载验证



（图中“WAVE TRANS. MODULE”为一个“10”序列检测器，用来处理进位信号）

再实例化一个 IP 核，用作 CE 的 1/4999999 占空比 1Hz 产生信号。

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 main(
    input clk_in1,
    input reset,
    input enable,
    output locked,
    output reg [6:0] seg,
    output reg [7:0] an
    );
    wire sig;  //flush signal
    wire sig_inc; //increment signal
    wire clk_div; //5MHz Clock signal
    wire counter_carry;
    wire [3:0] low_bcd;
    wire [3:0] hi_bcd;
    wire [6:0] seg_temp_low;
    wire [6:0] seg_temp_hi;
```

```
bcd_decoder low_decode(low_bcd, seg_temp_low);
bcd_decoder hi_decode(hi_bcd, seg_temp_hi);


pulse_gen p_gen(.clk_out1(clk_div), .sig(sig), .enable(1), .reset(0));

clk_wiz_0 c_clk(
   // Clock out ports
   .clk_out1(clk_div),     // output clk_out1
   // Status and control signals
   .reset(0), // input reset
   .locked(locked),       // output locked
  // Clock in ports
   .clk_in1(clk_in1));      // input clk_in1

c_counter_pulsegen sig_gen (
  .CLK(clk_div),          // input wire CLK
  .CE(1),            // input wire CE
  .THRESH0(sig_inc)  // output wire THRESH0
);

c_counter_binary_0 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
);

//Negedge of counter_carry detector
//Skew and delay: 1 cycle (1/5000000 sec.)
reg cc_prev;
reg carry_pulse;
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

c_counter_binary_0 hi_counter (
  .CLK(clk_div),          // input wire CLK
```

```
      .SCLR(reset),        // input wire SCLR
      .CE(carry_pulse),            // input wire CE
      .Q(hi_bcd)              // output wire [3 : 0] Q
    );



    always @ (clk_div) begin
        if (sig) begin
            seg <= seg_temp_low;
            an <= 8'b11111110;
        end
        else begin
            seg <= seg_temp_hi;
            an <= 8'b11111101;
        end
    end
endmodule

module pulse_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'd500;
    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'd500;
        end else count <= count - 1;
    end
end
endmodule
```

下载截图：

实验总结：本次实验的附加题很有意思——所有计数器公用高速 5MHz 时钟，并且用 1 周期信号进位；我从中学到了很多。