PB15111604 金泽文

实验目的：

- 学习时序逻辑电路
- 学会用 verilog 语言设计时序逻辑电路
- 掌握计数器的电路结构
- 掌握数码管动态扫描显示原理

实验内容：

- 实现一个 8bit 十进制（BCD 码）计数器
  - 复位时计数值为 8’ h90
  - 复位后，计数器实现累加操作，步长为 1，逢 9 进 1，计数值达到 8’ h99 后，从 0 开始继续计数
  - 使能信号为 1 时正常计数，为 0 时暂停计数
  - 每 0.5 秒左右计数值加 1
  - 结果显示在 led 灯上（必做）
  - 在 isim 下进行仿真（选做）
  - 将结果显示在 7 段数码管的后两位上

具体实现：

用 div 模块实现分频，

用 cnt 模块实现计数

用 code 模块实现数码管和 led 管的处理

再用 top 模块调用以上子模块

实验结果：

实验要求的选做必做内容均已实现。

实验分析：

本次试验最大的问题在仿真处理上，由于第一次进行这么复杂的仿真，以及从未使用过 initial 语句，导致了后面的一些问题。

中间也出现了一些 bug，被及时清除。

意见建议：

这次试验检查进度太慢，实验难度突然增大是一个原因呢，大家对 verilog 语法理解不够也是个原因，两个助教出国也是一个原因。建议这种情况下强制大家进行足够的预习或事先完成。

附录：

代码：

```
module cnt(
input                clk,
input                rst_n,
input                cnt_en,
output  reg [7:0]    cnt_data
);
initial
   cnt_data = 10010000;
always@(posedge clk or negedge rst_n)
begin
    if(~rst_n)
        cnt_data <= 8'b1001_0000;
    else if(cnt_en)
    begin
         if(cnt_data[3:0] == 4'b1001)
            begin
               if(cnt_data[7:4] == 4'b1001)
                  cnt_data[7:0] <= 8'b0000_0000;
               else
                  begin
                  cnt_data[3:0] <= 4'b0000;
                  cnt_data[7:4] <= cnt_data[7:4] + 4'b1;
                  end
            end
         else
            cnt_data[3:0] <= cnt_data[3:0] + 1'b1;
    end
end

endmodule
```

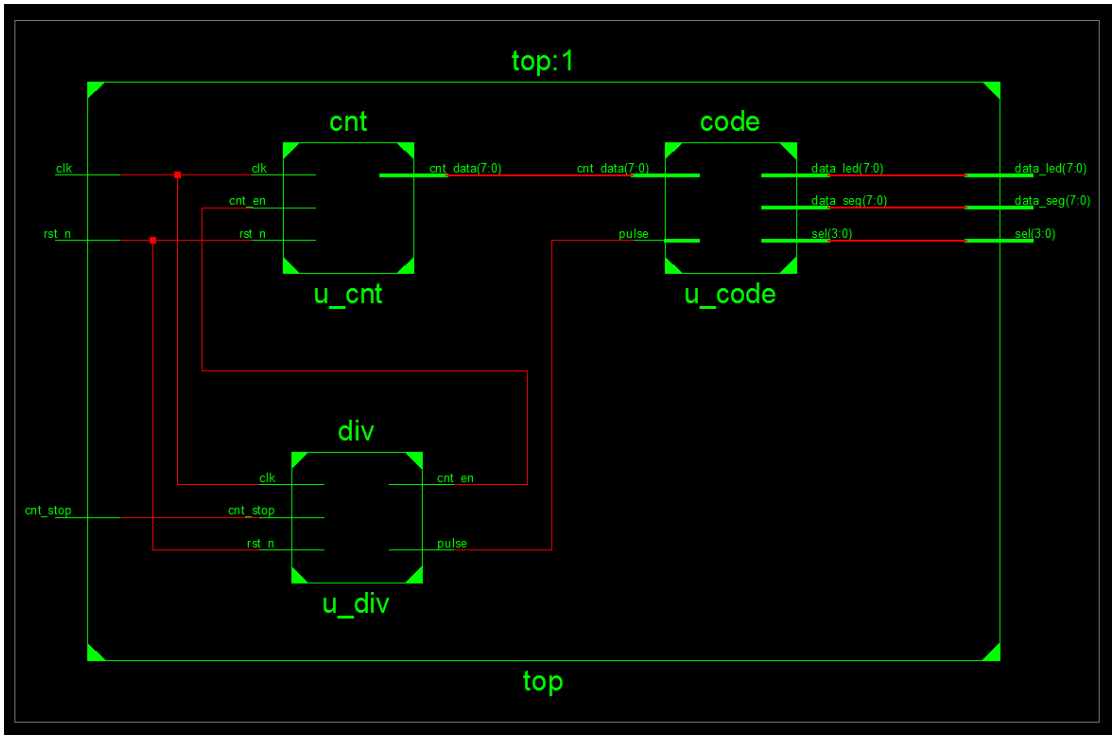
```
module div(
input       clk,
input       rst_n,
input       cnt_stop,
output  reg cnt_en,
output  reg pulse
);

reg     [31:0]  cnt_div;
initial
   cnt_en = 1;
initial
   pulse =1;
initial
   cnt_div = 32'b0;
always @(posedge clk)
begin
    if (cnt_div == 32'd49_999_999)
        cnt_div <= 32'd0;
    else
        cnt_div <= cnt_div + 32'd1;
end

always @(posedge clk or negedge rst_n) begin
    if (~rst_n)
        cnt_en <= 1'b0;
    else if(cnt_stop)
        ;
    else if (cnt_div==32'd49_999_999) begin
        cnt_en <= 1'b1;
    end
    else begin
        cnt_en <= 1'b0;
    end
end

always @(posedge clk) begin
    if (cnt_div[16]==1) begin
        pulse <= 1'b1;
    end
    else begin
        pulse <= 1'b0;
    end
end
endmodule

```

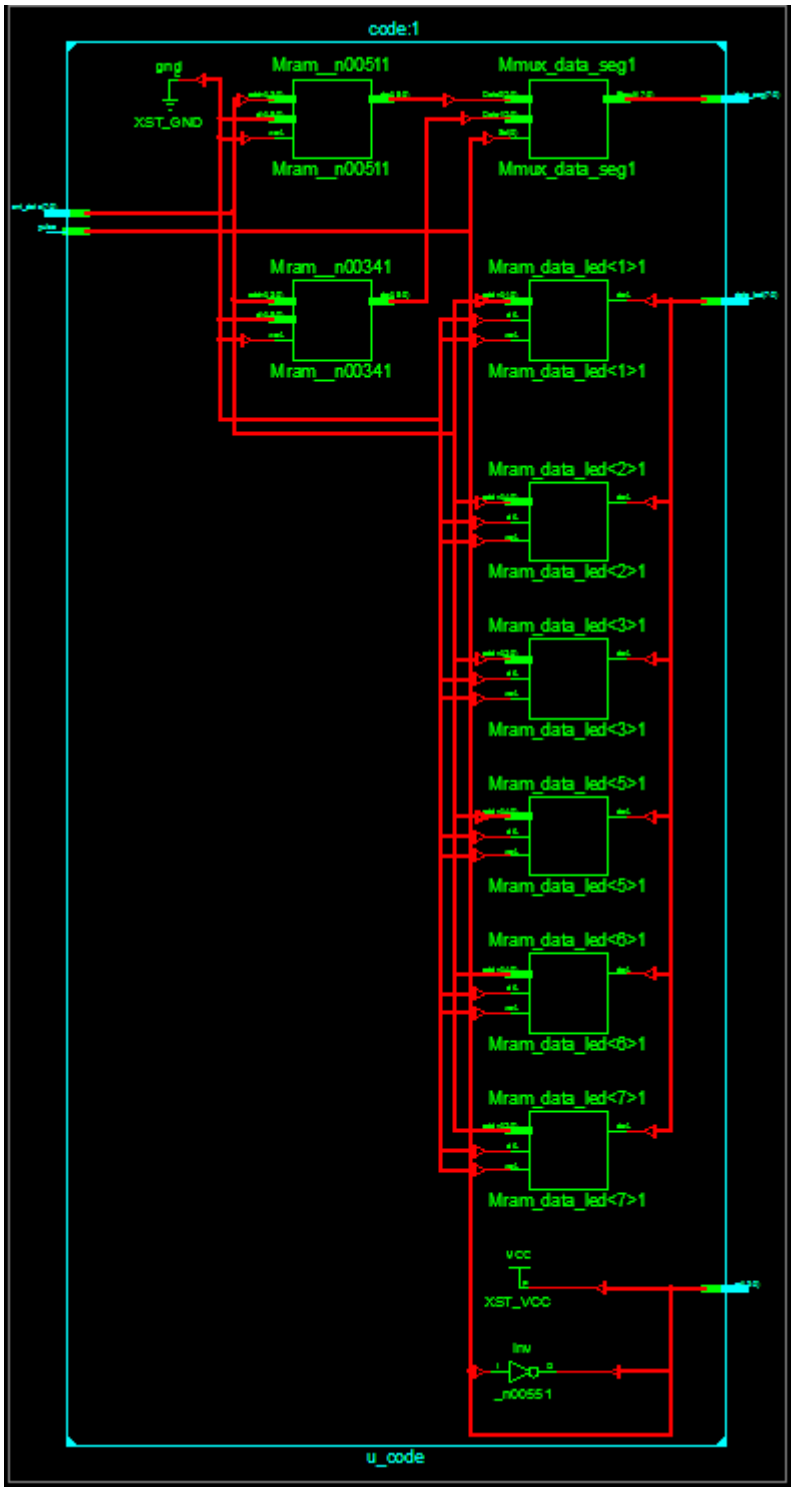
```
        4'd5:   data_seg = 8'b0100_1001;
        4'd6:   data_seg = 8'b0100_0001;
        4'd7:   data_seg = 8'b0001_1111;
        4'd8:   data_seg = 8'b0000_0001;
        4'd9:   data_seg = 8'b0000_1001;
        default:data_seg = 8'b1111_1111;
endcase
end

always@(*)
begin
    case(cnt_data[7:4])
        4'd0:   data_led[7:4] = 4'b0000;
        4'd1:   data_led[7:4] = 4'b0001;
        4'd2:   data_led[7:4] = 4'b0010;
        4'd3:   data_led[7:4] = 4'b0011;
        4'd4:   data_led[7:4] = 4'b0100;
        4'd5:   data_led[7:4] = 4'b0101;
        4'd6:   data_led[7:4] = 4'b0110;
        4'd7:   data_led[7:4] = 4'b0111;
        4'd8:   data_led[7:4] = 4'b1000;
        4'd9:   data_led[7:4] = 4'b1001;
        default:data_led[7:4] = 4'b0000;
endcase
end

always@(*)
begin
    case(cnt_data[3:0])
        4'd0:   data_led[3:0] = 4'b0000;
        4'd1:   data_led[3:0] = 4'b0001;
        4'd2:   data_led[3:0] = 4'b0010;
        4'd3:   data_led[3:0] = 4'b0011;
        4'd4:   data_led[3:0] = 4'b0100;
        4'd5:   data_led[3:0] = 4'b0101;
        4'd6:   data_led[3:0] = 4'b0110;
        4'd7:   data_led[3:0] = 4'b0111;
        4'd8:   data_led[3:0] = 4'b1000;
        4'd9:   data_led[3:0] = 4'b1001;
        default:data_led[3:0] = 4'b0000;
endcase
end

endmodule

```

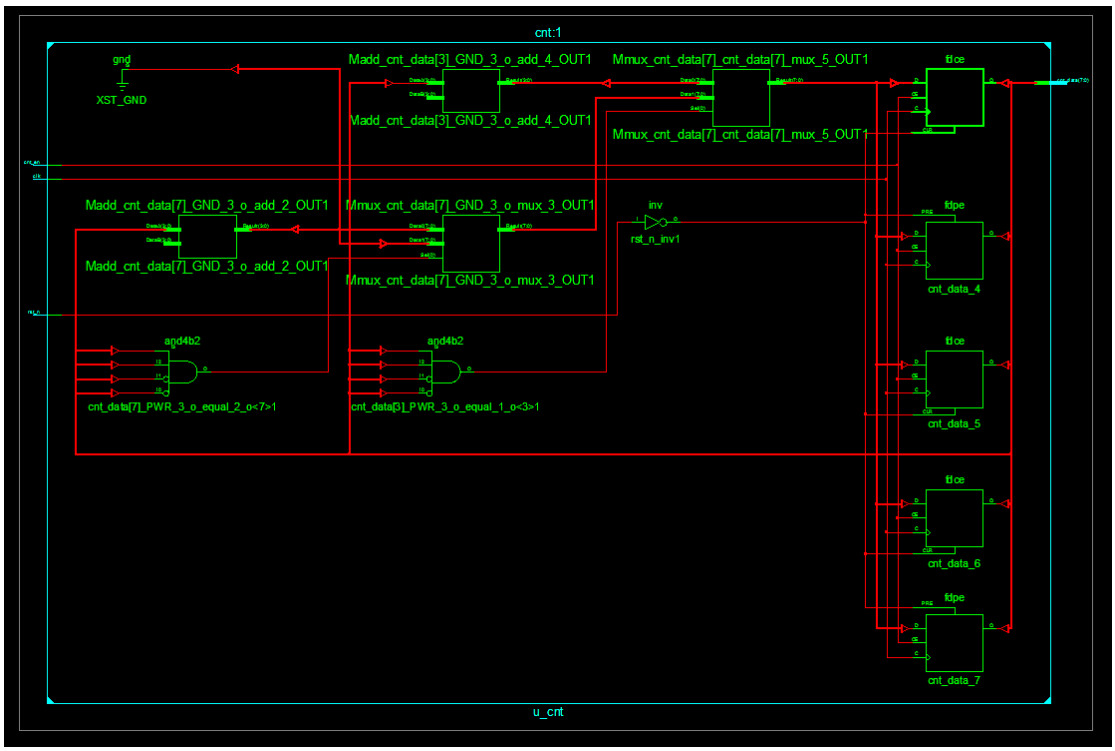
```
module code(
input           [7:0]  cnt_data,
input                  pulse,
output    reg   [3:0]  sel,
output    reg   [7:0]  data_seg,
output    reg   [7:0]  data_led
);
initial
   sel = 1110;

initial
   data_seg = 11111111;
initial
   data_led = 00000000;

always@(*)
begin
    if(pulse)
        sel <= 4'b1101;
    else
        sel <= 4'b1110;
end

always@(*)
begin
    if(pulse)
    case(cnt_data[7:4])
        4'd0:   data_seg = 8'b0000_0011;
        4'd1:   data_seg = 8'b1001_1111;
        4'd2:   data_seg = 8'b0010_0101;
        4'd3:   data_seg = 8'b0000_1101;
        4'd4:   data_seg = 8'b1001_1001;
        4'd5:   data_seg = 8'b0100_1001;
        4'd6:   data_seg = 8'b0100_0001;
        4'd7:   data_seg = 8'b0001_1111;
        4'd8:   data_seg = 8'b0000_0001;
        4'd9:   data_seg = 8'b0000_1001;
        default:data_seg = 8'b1111_1111;
    endcase
    else
    case(cnt_data[3:0])
        4'd0:   data_seg = 8'b0000_0011;
        4'd1:   data_seg = 8'b1001_1111;
        4'd2:   data_seg = 8'b0010_0101;
        4'd3:   data_seg = 8'b0000_1101;
        4'd4:   data_seg = 8'b1001_1001;
        4'd5:   data_seg = 8'b0100_1001;
```

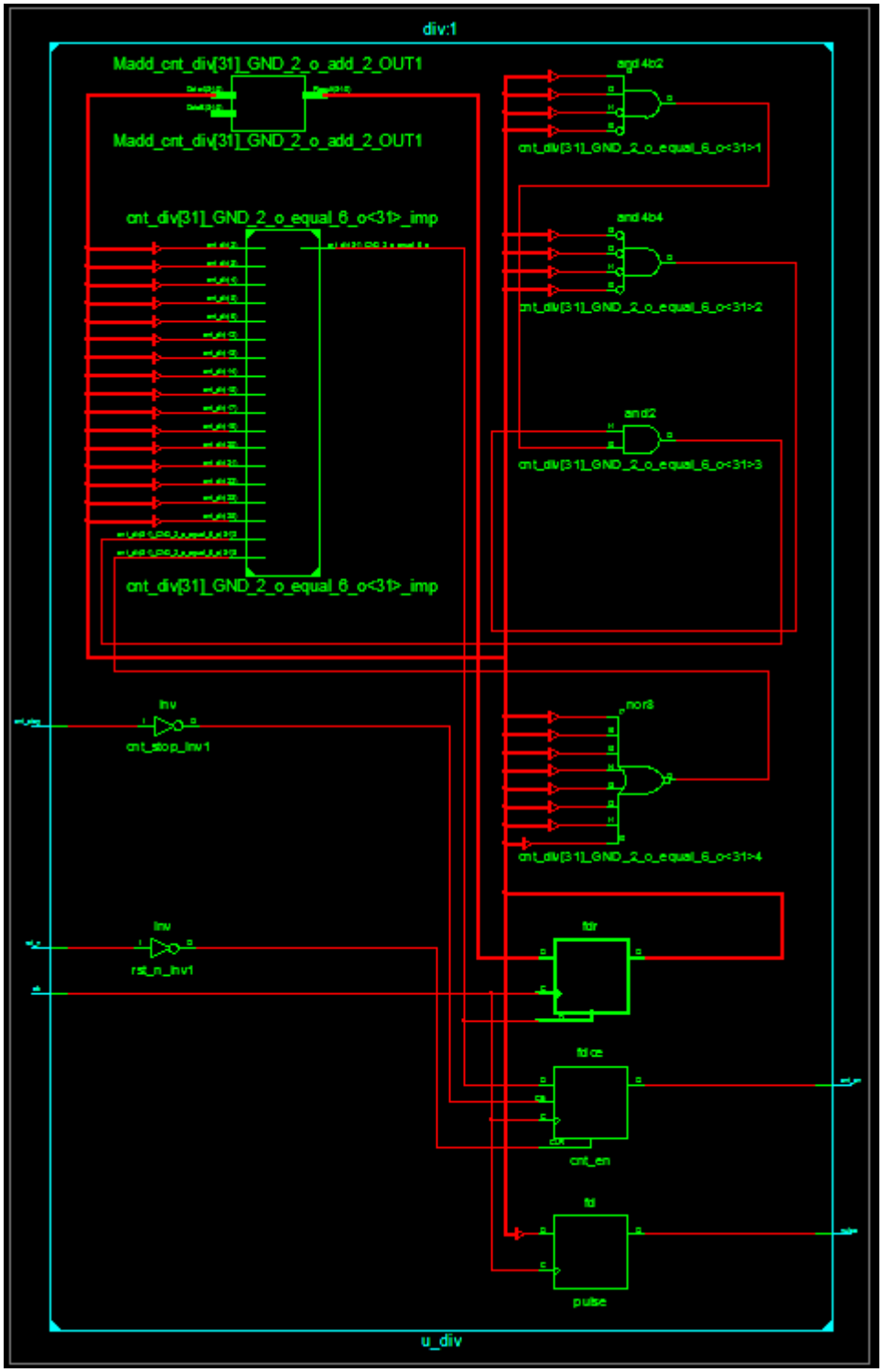
```
module top(
input        clk,
input        rst_n,
input        cnt_stop,
output   [3:0] sel,
output  [7:0]   data_led,
output  [7:0]   data_seg
);

wire                  cnt_en;
wire     [7:0]        cnt_data;
wire                  pulse;

div             u_div(
.clk           (clk     ),
.rst_n         (rst_n   ),
.cnt_stop      (cnt_stop),
.cnt_en        (cnt_en  ),
.pulse         (pulse)
);

cnt             u_cnt(
.clk           (clk     ),
.rst_n         (rst_n   ),
.cnt_en        (cnt_en  ),
.cnt_data      (cnt_data   )
);
code           u_code(
.cnt_data      (cnt_data   ),
.pulse         (pulse    ),
.sel           (sel     ),
.data_seg      (data_seg   ),
.data_led      (data_led   )
);

endmodule
```

原理图：


top:1
cnt
code
clk
clk
cnt_data(7:0)
cnt_data(7:0)
data_led(7:0)
data_led(7:0)
cnt_en
data_seg(7:0)
data_seg(7:0)
rst_n
rst_n
pulse
sel(3:0)
sel(3:0)
u_cnt
u_code
div
clk
cnt_en
cnt_stop
cnt_stop
rst_n
pulse
u_div
top


code:1
gnd
XST_GND
Mram__n00511
Mram__n00511
Mmux_data_seg1
Mmux_data_seg1
Mram__n00341
Mram__n00341
Mram_data_led<1>1
Mram_data_led<1>1
Mram_data_led<2>1
Mram_data_led<2>1
Mram_data_led<3>1
Mram_data_led<3>1
Mram_data_led<5>1
Mram_data_led<5>1
Mram_data_led<6>1
Mram_data_led<6>1
Mram_data_led<7>1
Mram_data_led<7>1
vcc
XST_VCC
Inv
_n00551
u_code


cnt:1
gnd
XST_GND
Madd_cnt_data[3]_GND_3_o_add_4_OUT1
Madd_cnt_data[3]_GND_3_o_add_4_OUT1
Mmux_cnt_data[7]_cnt_data[7]_mux_5_OUT1
Mmux_cnt_data[7]_cnt_data[7]_mux_5_OUT1
ftce
Madd_cnt_data[7]_GND_3_o_add_2_OUT1
Madd_cnt_data[7]_GND_3_o_add_2_OUT1
Mmux_cnt_data[7]_GND_3_o_mux_3_OUT1
Mmux_cnt_data[7]_GND_3_o_mux_3_OUT1
inv
rst_n_inv1
fdpe
cnt_data_4
agd4b2
agd4b2
cnt_data[7]_PWR_3_o_equal_2_o<7>1
cnt_data[3]_PWR_3_o_equal_1_o<3>1
ftce
cnt_data_5
ftce
cnt_data_6
fdpe
cnt_data_7
u_cnt


div:1
Madd_cnt_div[31]_GND_2_o_add_2_OUT1
Madd_cnt_div[31]_GND_2_o_add_2_OUT1
cnt_div[31]_GND_2_o_equal_6_o<31>_imp
cnt_div[31]_GND_2_o_equal_6_o<31>_imp
and4b2
cnt_div[31]_GND_2_o_equal_6_o<31>1
and4b4
cnt_div[31]_GND_2_o_equal_6_o<31>2
and2
cnt_div[31]_GND_2_o_equal_6_o<31>3
nor8
cnt_div[31]_GND_2_o_equal_6_o<31>4
inv
cnt_stop_inv1
inv
rst_n_inv1
fdr
fdce
cnt_en
fd
pulse
u_div