FPGA I2C 接口实现

计科 210X 甘晴void 202108010XXX

报告内容将包括：

（1）如何描述组合电路、时序电路、状态机？如何编写 TestBench？

（2）ModelSim 工具的使用；

（3）EEPROM 读写代码分析；

（4）实验总结；

注意：其中（1）将在自定 FSM 中以实例的方式呈现，（2）将在（3）中提及。

附件包括：

（1）FSM_example（文件夹），这是自定 FSM 的文件

（2）I2C（文件夹），这里放的是 EEPROM 的相关文件，其子文件夹包括

I2C_Rtl 目录：存放 i2c.v（设计文件）；

I2C_Tes 目录：存放 i2c_tb.v（测试文件）和 EEPROM 的模型文件（在仿真时用模型文件代替实际芯片，M24XXX_Macro.v，M24XXX_Memory.v，M24XXX_Parameters.v，请在“资料”栏中下载）；

I2C_Sim 目录：工程目录，用于存放工程文件等；

一、自定 FSM 说明

1、状态描述

设计如下状态，表示单日活动轨迹：

S0：宿舍休息

S1：起床并吃早餐，吃完后有课则上课（a=1），无课自习，

S2：12 节上课，若 34 有课则继续去上课（b=1），否则自习

S3：12 节自习，若 34 有课则去上课（b=1），否则自习

S4：34 节上课，结束后去吃午餐

S5：34 节自习，结束后去吃午餐

S6：吃午餐，午餐后有工训课就上工训课（c=1），否则午休

S7：午休，若 56 节有课（d=1）则去上课，否则去图书馆阅读

S8：中午工训课，若 56 节有课（d=1）则去上课，否则去图书馆阅读

S9：56 节上课，若 78 有课则继续去上课（e=1），否则若晴天（f=1）则运动，雨天则自习

S10：56 节阅读，若 78 有课则继续去上课（e=1），否则若晴天（f=1）则运动，雨天则自习

S11：78 节上课，结束后去吃晚餐

S12：78 节运动，结束后去吃晚餐

S13：78 节自习（教学楼），结束后去吃晚餐

S14：晚餐，吃完后自习，若周五直接回寝室(g=1)

S15：教学楼晚自习，结束后直接回寝休息

输出 pos 为位置：

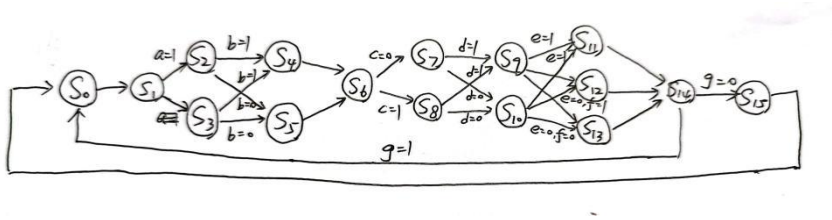
00：宿舍园区

01：教学楼

10：体育场

11：图书馆

状态图如下：



2、设计代码说明

（1）状态机描述代码如下：

```
//输入端口为控制信号和时钟信号，输出端口为 pos，表示位置
module fsm_example(
    input clk;
    input a,b,c,d,e,f,g,
    output reg [1:0] pos
)

//使用 4 位 16 进制数表示所有状态
parameter
s0=4'h0,s1=4'h1,s2=4'h2,s3=4'h3,s4=4'h4,s5=4'h5,s6=4'h6,s7=4'h7,s8=4
'h8,s9=4'h9,s10=4'ha,s11=4'hb,s12=4'hc,s13=4'hd,s14=4'he,s15=4'hf;
reg [3:0] state,next_state;
//下一状态判断
always @(*) begin
    case (state)
        s0: next_state=s1;
        s1: if(a) next_state=s2;
            else next_state=s3;
        s2: if(b) next_state=s4;
            else next_state=s5;
        s3: if(b) next_state=s4;
            else next_state=s5;
        s4: next_state=s6;
        s5: next_state=s6;
        s6: if(c) next_state=s8;
            else next_state=s7;
        s7: if(d) next_state=s9;
            else next_state=s10;
        s8: if(d) next_state=s9;
            else next_state=s10;
        s9: if(e) next_state=s11;
            else begin
```

```
                if(f) next_state=s12;
                else next_state=s13;
            end
        s10: if(e) next_state=s11;
            else begin
                if(f) next_state=s12;
                else next_state=s13;
            end
        s11: next_state=s14;
        s12: next_state=s14;
        s13: next_state=s14;
        s14: if(g) next_state=15;
             else next_state=s0;
        s15: next_state=s0;
        default: next_state=s0;
    endcase
end
//状态更新与输出
always @(posedge clk) state<=next_state;

always @(*) begin
    case (state)
        s0: pos=2'b00;
        s1: pos=2'b00;
        s2: pos=2'b01;
        s3: pos=2'b01;
        s4: pos=2'b01;
        s5: pos=2'b01;
        s6: pos=2'b00;
        s7: pos=2'b00;
        s8: pos=2'b01;
        s9: pos=2'b01;
        s10: pos=2'b11;
        s11: pos=2'b01;
        s12: pos=2'b10;
        s13: pos=2'b01;
        s14: pos=2'b00;
        s15: pos=2'b01;
        default: pos=2'b00;
    endcase
End

endmodule
```

（2）test_bench 代码如下：

```
`timescale 1ns/100ps
module xpos_tb();

parameter
s0=4'h0,s1=4'h1,s2=4'h2,s3=4'h3,s4=4'h4,s5=4'h5,s6=4'h6,s7=4'h
7,s8=4'h8,s9=4'h9,s10=4'ha,s11=4'hb,s12=4'hc,s13=4'hd,s14=4'he,
s15=4'hf;
reg [3:0] state,next_state;
reg a,b,c,d,e,f,g;
wire [1:0] pos;
pos xpos(clk,a,b,c,d,e,f,g,pos);
initial clk=0;
always #50 clk=~clk;
initial begin
    a=0;b=0;c=0;d=0;e=0;f=0;g=0;
    #1
    #400
    a=1;
    b=1;
    c=1;
    d=0;
    e=0;
    f=1;
    g=0;
    #600
    repeat(1024) @(posedge clk);
    $stop;
end
endmodule
```

第一次从 s0 开始，设定状态变化为：

s0-> s1->s2->s4->s6->s8->s10->s12->s14->s15->s0

二、EEPROM 读写代码设计及仿真

1、代码说明

输入：

clk，rstn 分别为时钟和复位信号

write_op：写命令，低电平有效

write_data：写数据

addr：地址

read_op：读命令，低电平有效

输出：

read_data：读到的数据

op_done：操作结束

I2C 协议信号：

scl：I2C 协议的 scl 信号
sda：I2C 协议的 sda 信号

```
`timescale 1ns / 1ps
module i2c(
    input clk,                  //时钟
    input rstn,              //复位
    input write_op,             //写操作
    input [7:0]write_data,          //写入的数据
    input read_op,              //读操作
    output reg [7:0]read_data,          //读出的数据
    input [7:0]addr,                //地址
    output op_done,             //操作结束
    output reg scl,             //scl
    inout sda                   //sda
);
```

使用 8 位 16 进制数表示所有状态，共 55 个：

```
14  //I2C状态
15  parameter IDLE =8'h00,
16            WAIT_WTICK0=8'h01,
17            WAIT_WTICK1=8'h02,
18            W_START=8'h03,
19            W_DEVICE7=8'h04,
20            W_DEVICE6=8'h05,
21            W_DEVICE5=8'h06,
22            W_DEVICE4=8'h07,
23            W_DEVICE3=8'h08,
24            W_DEVICE2=8'h09,
25            W_DEVICE1=8'h0a,
26            W_DEVICE0=8'h0b,
27            W_DEVACK=8'h0c,
28            W_ADDRES7=8'h0d,
29            W_ADDRES6=8'h0e,
30            W_ADDRES5=8'h0f,
31            W_ADDRES4=8'h10,
32            W_ADDRES3=8'h11,
33            W_ADDRES2=8'h12,
34            W_ADDRES1=8'h13,
35            W_ADDRES0=8'h14,
36            W_AACK=8'h15,
37            W_DATA7=8'h16,
38            W_DATA6=8'h17,
39            W_DATA5=8'h18,
40            W_DATA4=8'h19,
41            W_DATA3=8'h1a,
42            W_DATA2=8'h1b,
43            W_DATA1=8'h1c,
44            W_DATA0=8'h1d,
45            W_DACK=8'h1e,
```

```
46          WAIT_WTICK3=8'h1f,
47          R_START=8'h20,
48          R_DEVICE7=8'h21,
49          R_DEVICE6=8'h22,
50          R_DEVICE5=8'h23,
51          R_DEVICE4=8'h24,
52          R_DEVICE3=8'h25,
53          R_DEVICE2=8'h26,
54          R_DEVICE1=8'h27,
55          R_DEVICE0=8'h28,
56          R_DACK=8'h29,
57          R_DATA7=8'h2a,
58          R_DATA6=8'h2b,
59          R_DATA5=8'h2c,
60          R_DATA4=8'h2d,
61          R_DATA3=8'h2e,
62          R_DATA2=8'h2f,
63          R_DATA1=8'h30,
64          R_DATA0=8'h31,
65          R_NOACK=8'h32,
66          S_STOP=8'h33,
67          S_STOP0=8'h34,
68          S_STOP1=8'h35,
69          W_OPOVER=8'h36;
```

scl 周期是使用计数器对时钟周期计数实现的，一个 scl 周期是 30 个时钟周期，30*200k=6Mhz，为所使用的 FPGA 板的时钟频率。

```
reg [7:0]i2c,next_i;                        //当前状态，下一状态
reg [7:0]div_cnt;                       //计数器
wire scl_tick;
//计数
always @(posedge clk or negedge rstn)
if(!rstn) div_cnt <=8'd0;
else if((i2c==IDLE)|scl_tick) div_cnt <=8'd0;
else div_cnt<=div_cnt+1'b1;
//scl同步
wire scl_ls =(div_cnt==8'd0);               //scl low
wire scl_lc = (div_cnt==8'd7);              //scl low center
wire scl_hs =(div_cnt==8'd15);          //scl high
wire scl_hc = (div_cnt==8'd22);         //scl high center
assign scl_tick = (div_cnt==8'd29);         //一个周期结束
```

下一状态的更新：

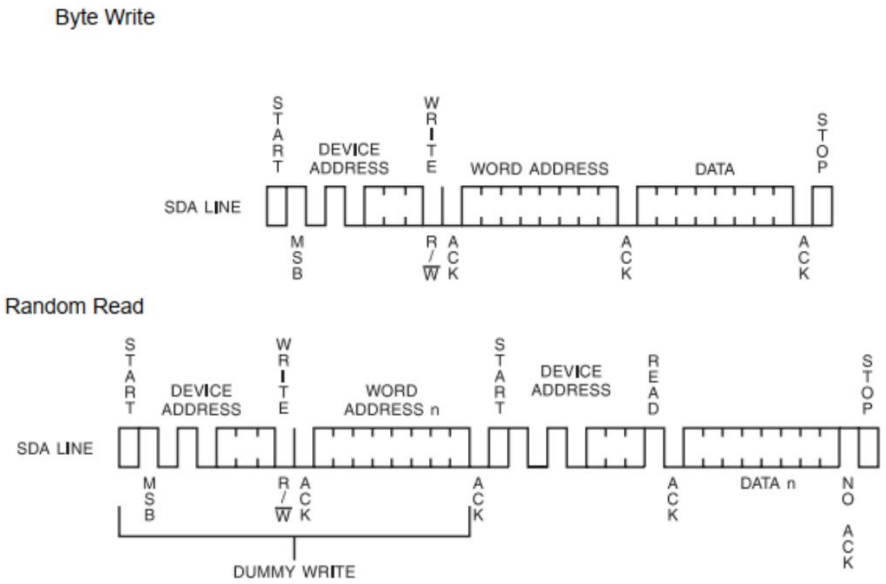
```
//状态
always @(posedge clk or negedge rstn)
if(!rstn) i2c <=0;
else i2c <= next_i;
```

使用 wr_op 和 rd_op 将输入信号 write_op，read_op 表示的读写命令用高电平表示：

```
//Byte Write : START + DEVICE +ACK + ADDR + ACK + DATA + ACK + STOP
//Random Read : START + DEVICE + ACK +ADDR + START + DEVICE + DATA + NO ACK + STOP
reg wr_op,rd_op;                    //读写操作

always @ (posedge clk or negedge rstn)
if(!rstn) wr_op <= 0;
else if (i2c==IDLE) wr_op <= ~write_op;
else if(i2c==W_OPOVER) wr_op <=1'b0;

always @(posedge clk or negedge rstn)
if(!rstn) rd_op <= 0;
else if (i2c==IDLE) rd_op <= ~read_op;
else if(i2c==W_OPOVER) rd_op <=1'b0;
```

下一状态判断，与状态图一致，时间为 scl_tick，即 scl 周期结束。

首先是在 scl=1 时，sda 由 1->0，开始数据传输，并先写入器件地址(10100000)和数据地址，然后根据 wr_op，rd_op 判断进行读还是写操作，写操作直接开始写入数据，读操作则需要重新写器件地址和数据地址，然后读取数据。



以下是下一状态判断部分的代码

```
//下一状态判断
always@(*)
case (i2c)
    IDLE: begin next_i = IDLE;if(wr_op|rd_op) next_i = WAIT_WTICK0;end      //有读写操作跳出空闲状态

    //wait tick
    WAIT_WTICK0:begin next_i = WAIT_WTICK0;if(scl_tick) next_i=WAIT_WTICK1;end
    WAIT_WTICK1:begin next_i = WAIT_WTICK1;if(scl_tick) next_i = W_START;end

    //START:SCL=1,SDA=1->0(scl_lc)
    W_START:begin next_i=W_START;if(scl_tick) next_i=W_DEVICE7;end

    //DEVICE ADDRESS（1010_000_0(WRITE)）
    W_DEVICE7:begin next_i = W_DEVICE7;if(scl_tick) next_i=W_DEVICE6;end
    W_DEVICE6:begin next_i = W_DEVICE6;if(scl_tick) next_i=W_DEVICE5;end
    W_DEVICE5:begin next_i = W_DEVICE5;if(scl_tick) next_i=W_DEVICE4;end
    W_DEVICE4:begin next_i = W_DEVICE4;if(scl_tick) next_i=W_DEVICE3;end
    W_DEVICE3:begin next_i = W_DEVICE3;if(scl_tick) next_i=W_DEVICE2;end
    W_DEVICE2:begin next_i = W_DEVICE2;if(scl_tick) next_i=W_DEVICE1;end
    W_DEVICE1:begin next_i = W_DEVICE1;if(scl_tick) next_i=W_DEVICE0;end
    W_DEVICE0:begin next_i = W_DEVICE0;if(scl_tick) next_i=W_DEVACK;end

    //ACK
    W_DEVACK:begin next_i=W_DEVACK;if(scl_tick) next_i=W_ADDRES7;end
    //WORD ADDRESS
    W_ADDRES7 :begin next_i = W_ADDRES7;if(scl_tick) next_i=W_ADDRES6;end
    W_ADDRES6 :begin next_i = W_ADDRES6;if(scl_tick) next_i=W_ADDRES5;end
    W_ADDRES5 :begin next_i = W_ADDRES5;if(scl_tick) next_i=W_ADDRES4;end
    W_ADDRES4 :begin next_i = W_ADDRES4;if(scl_tick) next_i=W_ADDRES3;end
    W_ADDRES3 :begin next_i = W_ADDRES3;if(scl_tick) next_i=W_ADDRES2;end
    W_ADDRES2 :begin next_i = W_ADDRES2;if(scl_tick) next_i=W_ADDRES1;end
    W_ADDRES1 :begin next_i = W_ADDRES1;if(scl_tick) next_i=W_ADDRES0;end
    W_ADDRES0 :begin next_i = W_ADDRES0;if(scl_tick) next_i=W_AACK;end

    //ACK
    W_AACK:begin next_i = W_AACK;
                if(scl_tick&wr_op) next_i=W_DATA7;          //wr_op即写命令，开始写数据
                else if(scl_tick&rd_op) next_i=WAIT_WTICK3;         //rd_op读命令,则下一状态为WAIT_WTICK3
           end

    //WRITE DATA[7:0]
    W_DATA7:begin next_i=W_DATA7;if(scl_tick)next_i=W_DATA6;end
    W_DATA6:begin next_i=W_DATA6;if(scl_tick)next_i=W_DATA5;end
    W_DATA5:begin next_i=W_DATA5;if(scl_tick)next_i=W_DATA4;end
    W_DATA4:begin next_i=W_DATA4;if(scl_tick)next_i=W_DATA3;end
    W_DATA3:begin next_i=W_DATA3;if(scl_tick)next_i=W_DATA2;end
    W_DATA2:begin next_i=W_DATA2;if(scl_tick)next_i=W_DATA1;end
    W_DATA1:begin next_i=W_DATA1;if(scl_tick)next_i=W_DATA0;end
    W_DATA0:begin next_i=W_DATA0;if(scl_tick)next_i=W_DACK;end
```

```
    //ACK
    W_DACK:begin next_i=W_DACK; if(scl_tick) next_i=S_STOP;end

    //Current Address Read
    //START: SCL=1,SDA=1->0(scl_lc)

    WAIT_WTICK3:begin next_i=WAIT_WTICK3; if(scl_tick) next_i=R_START;end
    R_START:begin next_i=R_START; if(scl_tick)next_i=R_DEVICE7;end

    //DEVICE ADDRESS(1010_000_1(READ))
    R_DEVICE7:begin next_i=R_DEVICE7; if(scl_tick) next_i=R_DEVICE6;end
    R_DEVICE6:begin next_i=R_DEVICE6; if(scl_tick) next_i=R_DEVICE5;end
    R_DEVICE5:begin next_i=R_DEVICE5; if(scl_tick) next_i=R_DEVICE4;end
    R_DEVICE4:begin next_i=R_DEVICE4; if(scl_tick) next_i=R_DEVICE3;end
    R_DEVICE3:begin next_i=R_DEVICE3; if(scl_tick) next_i=R_DEVICE2;end
    R_DEVICE2:begin next_i=R_DEVICE2; if(scl_tick) next_i=R_DEVICE1;end
    R_DEVICE1:begin next_i=R_DEVICE1; if(scl_tick) next_i=R_DEVICE0;end
    R_DEVICE0:begin next_i=R_DEVICE0; if(scl_tick) next_i=R_DACK;end

    //ACK
    R_DACK:begin next_i=R_DACK;if(scl_tick) next_i=R_DATA7;end

    //READ DATA[7:0], SDA:input
    R_DATA7:begin next_i=R_DATA7;if(scl_tick) next_i=R_DATA6;end
    R_DATA6:begin next_i=R_DATA6;if(scl_tick) next_i=R_DATA5;end
    R_DATA5:begin next_i=R_DATA5;if(scl_tick) next_i=R_DATA4;end
    R_DATA4:begin next_i=R_DATA4;if(scl_tick) next_i=R_DATA3;end
    R_DATA3:begin next_i=R_DATA3;if(scl_tick) next_i=R_DATA2;end
    R_DATA2:begin next_i=R_DATA2;if(scl_tick) next_i=R_DATA1;end
    R_DATA1:begin next_i=R_DATA1;if(scl_tick) next_i=R_DATA0;end
    R_DATA0:begin next_i=R_DATA0;if(scl_tick) next_i=R_NOACK;end

    //NO ACK
    R_NOACK:begin next_i=R_NOACK;if(scl_tick) next_i=S_STOP;end

    //STOP
    S_STOP:begin next_i=S_STOP;if(scl_tick) next_i=S_STOP0;end
    S_STOP0:begin next_i=S_STOP0;if(scl_tick) next_i=S_STOP1;end
    S_STOP1:begin next_i=S_STOP1;if(scl_tick) next_i=W_OPOVER;end

    //WAIT write_op=0,read_op=0;
    W_OPOVER:begin next_i = W_OPOVER;if(d5ms_over)next_i=IDLE;end       //操作结束回到空闲状态
    default:begin next_i= IDLE;end
endcase
```

SCL 同步的实现：

```
202  //SCL
203  assign clr_scl=scl_ls&(i2c!=IDLE)&(i2c!=WAIT_WTICK0)&                  //clr_scl，scl置0信号
204                 (i2c != WAIT_WTICK1)&(i2c!=W_START)&(i2c!=R_START)
205                 &(i2c!=S_STOP0)&(i2c!=S_STOP1)&(i2c!=W_OPOVER);
206
207  always @(posedge clk or negedge rstn)
208  if(!rstn) scl <= 1'b1;                                  //复位，scl为高电平
209  else if(clr_scl) scl <= 1'b0;                              //scl 1->0
210  else if(scl_hs) scl <=1'b1;                             //scl 0->1
211
```

空闲，等待，操作结束，start 开始等状态下 SCL 都是高电平，因此不需要 clr_scl 对 SCL 清零。另外 clr_scl 只在 scl_ls（scl 的低电平开始）处才置 1，把 scl 清 0，在 15 个 clk 周期的 scl_hs 处,再把 scl 拉高，就实现了 SCL 周期。

SDA：

SDA 的控制信号声明，这些信号在对应的状态且 scl 在低电平的中心时置 1，根据

这些控制信号，在 SDA 上进行数据读写。而 i2c_reg 用来暂存数据。

```
//SDA
reg [7:0]i2c_reg;
assign start_clr = scl_lc &((i2c==W_START)|(i2c==R_START));            //在scl low center开始读写操作
assign ld_wdevice = scl_lc&(i2c==W_DEVICE7);                          //加载器件地址
assign ld_waddres = scl_lc&(i2c==W_ADDRES7);                          //加载数据地址
assign ld_wdata= scl_lc&(i2c==W_DATA7);                       //加载数据
assign ld_rdevice = scl_lc&(i2c==R_DEVICE7);                          //读操作的器件地址
assign noack_set = scl_lc&(i2c==R_NOACK);                        //读操作完毕
assign stop_clr = scl_lc&(i2c==S_STOP);
assign stop_set = scl_lc&((i2c==S_STOP0)|(i2c==WAIT_WTICK3));
```

使用信号 i2c_rlf 表示是否有读写操作，如果有，则 i2c_reg 将左移，一位一位处理数据。

```
222
223   assign i2c_rlf =scl_lc&(                                //有读写则i2c_rlf
224                           (i2c == W_DEVICE6)|
225                           (i2c == W_DEVICE5)|
226                           (i2c == W_DEVICE4)|
227                           (i2c == W_DEVICE3)|
228                           (i2c == W_DEVICE2)|
229                           (i2c == W_DEVICE1)|
230                           (i2c == W_DEVICE0)|
231                           (i2c == W_ADDRES6)|
232                           (i2c == W_ADDRES5)|
233                           (i2c == W_ADDRES4)|
234                           (i2c == W_ADDRES3)|
235                           (i2c == W_ADDRES2)|
236                           (i2c == W_ADDRES1)|
237                           (i2c == W_ADDRES0)|
238                           (i2c == W_DATA6)|
239                           (i2c == W_DATA5)|
240                           (i2c == W_DATA4)|
241                           (i2c == W_DATA3)|
242                           (i2c == W_DATA2)|
243                           (i2c == W_DATA1)|
244                           (i2c == W_DATA0)|
245                           (i2c == R_DEVICE6)|
246                           (i2c == R_DEVICE5)|
247                           (i2c == R_DEVICE4)|
248                           (i2c == R_DEVICE3)|
249                           (i2c == R_DEVICE2)|
250                           (i2c == R_DEVICE1)|
251                           (i2c == R_DEVICE0));
```

根据上述控制信号，将输入的特定数据保存到 i2c_reg

```
253   always@(posedge clk or negedge rstn)
254   if(!rstn) i2c_reg <= 8'hff;                                  //复位，高电平
255   else if(start_clr) i2c_reg <= 8'h00;                                 //开始读写，低电平
256   else if(ld_wdevice) i2c_reg <= {4'b1010,3'b000,1'b0};                        //10100000 写
257   else if(ld_waddres) i2c_reg <= addr;                                 //加载数据地址
258   else if(ld_wdata) i2c_reg <= write_data;                                 //加载写入的数据
259   else if(ld_rdevice) i2c_reg <= {4'b1010,3'b000,1'b1};                        //10100001 读
260   else if(noack_set) i2c_reg <= 8'hff;                                 //NOACK
261   else if(stop_clr) i2c_reg <= 8'h00;
262   else if(stop_set) i2c_reg <= 8'hff;
263   else if(i2c_rlf) i2c_reg <= {i2c_reg[6:0],1'b0};                             //左移
```

sda 输出使用 sda 使能信号 sda_en 控制，写器件地址，数据地址，写数据时使能信号为 1，接收 ACK 响应时使能为 0。sda 输出 i2c_reg 的最高位，即一位一位完成读或写。

```
assign sda_o = i2c_reg[7];                          //sda输出
assign clr_sdaen = (i2c==IDLE)|                     //sda使能置0信号
                   (scl_lc&(
                   (i2c==W_DEVACK)|
                   (i2c==W_AACK)|
                   (i2c==W_DACK)|
                   (i2c==R_DACK)|
                   (i2c==R_DATA7)));

assign set_sdaen = scl_lc&(                         //sda使能置1信号
                   (i2c==WAIT_WTICK0)|
                   (i2c==W_ADDRES7)|
                   (i2c==W_DATA7)|
                   (i2c==WAIT_WTICK3)|
                   (i2c==S_STOP)|
                   (i2c==R_NOACK));

reg sda_en;
always @(posedge clk or negedge rstn)
if(!rstn) sda_en <= 0;
else if (clr_sdaen) sda_en <=0;
else if(set_sdaen) sda_en <= 1'b1;

assign sda= sda_en?sda_o: 1'bz;                     //sda使能为1时sda可工作
```

读取数据时将数据读到 read_data

```
assign sda_wr = scl_hc &(                           //读数据
                (i2c==R_DATA7)|
                (i2c==R_DATA6)|
                (i2c==R_DATA5)|
                (i2c==R_DATA4)|
                (i2c==R_DATA3)|
                (i2c==R_DATA2)|
                (i2c==R_DATA1)|
                (i2c==R_DATA0));

always@(posedge clk or negedge rstn)
if(!rstn) read_data <= 0;
else if(sda_wr) read_data <= {read_data[6:0],sda};          //左移读入数据
```

最后使用 d5ms_count 计数时钟周期等待，使用时钟频率为 6Mhz，一个周期 166ns，等待约 1.36ms，然后重新开始完成新的读写命令。

```
//op_done
assign op_done = (i2c == W_OPOVER);                 //操作结束

//Write Cycle(5ms)
//6MHZ = 166ns,5ms/166ns = 31
reg [12:0] d5ms_cnt;
always @(posedge clk or negedge rstn)
if(!rstn)    d5ms_cnt <= 8'd0;
else if(i2c==IDLE) d5ms_cnt <= 8'd0;
else if(i2c==W_OPOVER) d5ms_cnt <= d5ms_cnt + 1'b1;

assign d5ms_over = (d5ms_cnt==13'h1FFF);

endmodule
```

2、TestBench 代码说明

模块声明与实例化：

```
`timescale 1ns / 1ps

`include "../I2C_Tes/M24XXX_Memory.v"
`include "../I2C_Rtl/i2c.v"

module i2c_tb();

reg clk;
reg rstn;
reg write_op;
reg [7:0] write_data;
reg read_op;
wire [7:0] read_data;
reg [7:0] addr;

wire scl;
wire sda;

pullup(sda);
i2c i2c_dut(
 .clk (clk),
 .rstn(rstn),
 .write_op(write_op),
 .write_data(write_data),
 .read_op(read_op),
 .read_data(read_data),
 .addr(addr),
 .op_done(op_done),
 .scl(scl),
 .sda(sda)
);
//EEPROM
M24XXX M24XXX_dut(
    .Ei(3'b0),
    .SDA(sda),
    .SCL(scl),
    .WC(1'b0),
    .VCC(1'b1)
);
```

根据时钟频率 6Mhz 设置周期 166ns，并对信号初始化：

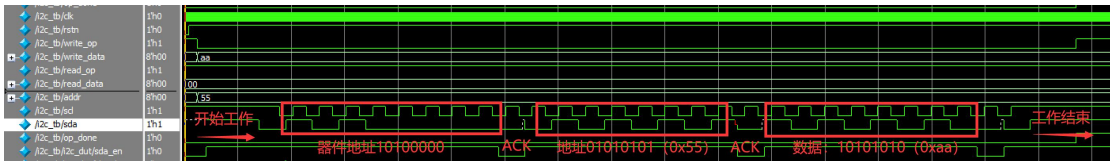
```
always #(166/2) clk = ~clk;                //6Mhz

initial
begin
    clk = 0;
    rstn = 0;
    write_op=1'b1;
    write_data=8'h00;
    read_op=1'b1;
    addr=0;

    repeat(5) @(posedge clk);
    rstn = 1'b1;
end
```

首先输入写命令信号，地址为 8‘h55，写入的数据为 8’haa。等待操作完成后，将 write_op 设为 1（高电平无效），输入读命令信号，读出地址 8’h55 中的数据，读出的数据应该为刚刚写入的 8’haa。

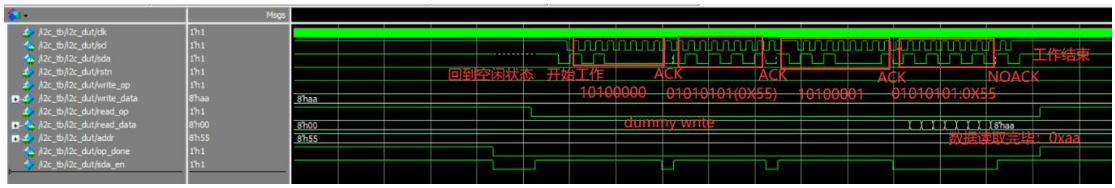
```
56  initial
57  begin
58      wait(rstn);
59      repeat(10) @(posedge clk);
60      write_op=1'b0;
61      addr = 8'h55;
62      write_data= 8'haa;
63
64      wait(op_done);
65      write_op=1'b1;
66      $display ($stime/1,"ns","Write:Addr(%h)=(%h)\n",addr,write_data);
67
68      wait(!op_done);
69      repeat(100)@(posedge clk);
70      read_op=1'b0;
71      addr = 8'h55;
72      wait(op_done);
73      read_op=1'b1;
74      $display ($stime/1,"ns","Read:Addr(%h)=(%h)\n",addr,read_data);
75
76      repeat(1000) @(posedge clk);
77      $stop;
78  end
79
80  endmodule
```

3、仿真波形说明（截图+文字标注）

首先是向地址为 8‘h55 处写入数据，sda 在 scl 为高电平时产生下降沿，表示开始工作，scl 开始翻转，依次写入器件地址，数据地址，以及数据 8’haa，并接收响应。最后 scl 为高电平，sda 产生上升沿，停止工作。

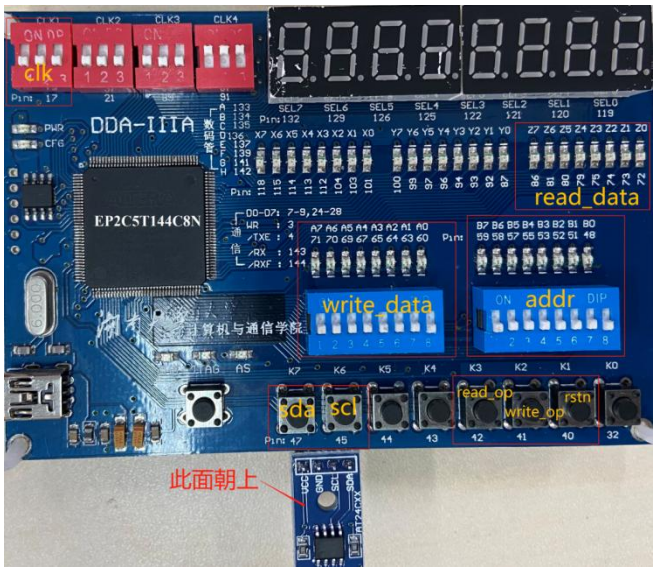


经过等待后开始读出 0x55 处的数据，先写入器件地址，数据地址（dummy write），然后再次写入器件地址，读出数据。



4、下载到板子并进行验证

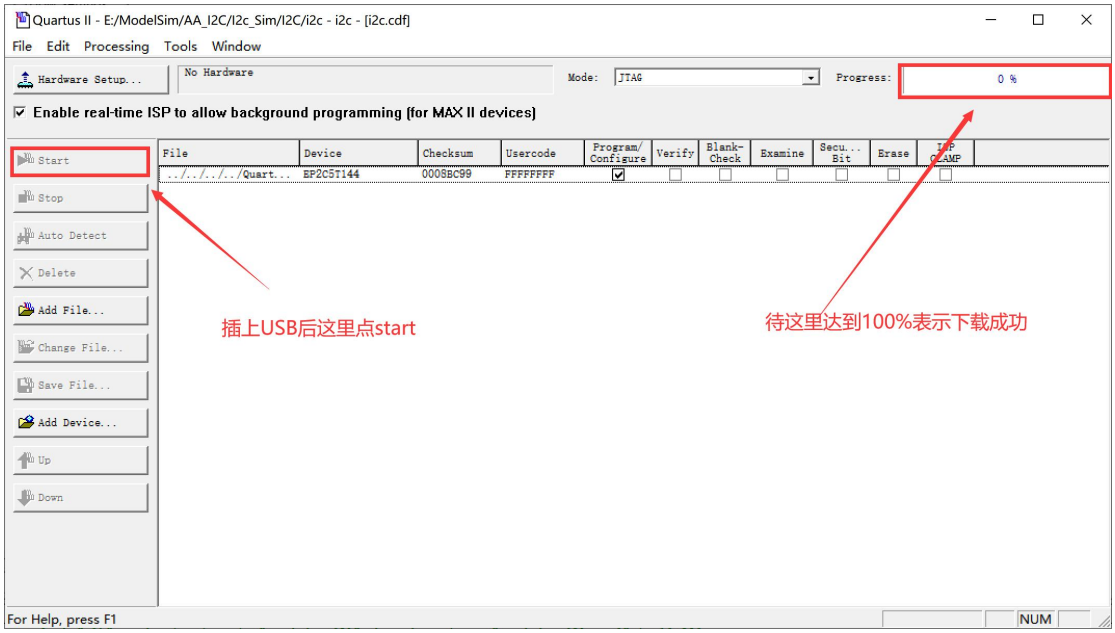
按照教师要求，利用 Quartus 分配管脚，按下图实物图指示分配管脚，Quartus 管脚分配截图如下。



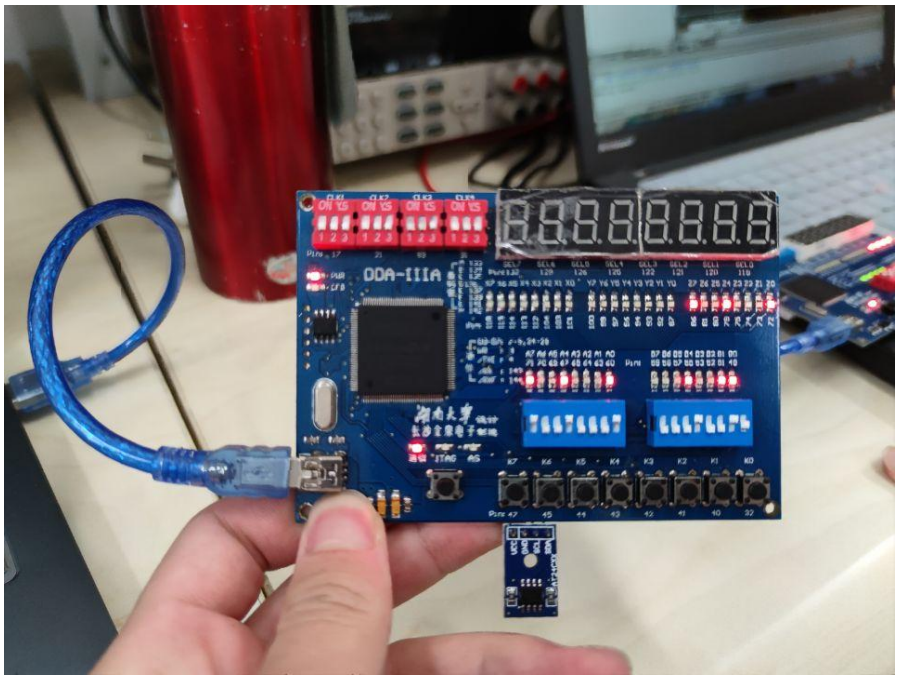
Quartus 管脚分配：

	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Group	Current Strength	PCB layer
1	addr[7]	Input	PIN_59	4	B4_N0	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
2	addr[6]	Input	PIN_58	4	B4_N1	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
3	addr[5]	Input	PIN_57	4	B4_N1	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
4	addr[4]	Input	PIN_55	4	B4_N1	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
5	addr[3]	Input	PIN_53	4	B4_N1	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
6	addr[2]	Input	PIN_52	4	B4_N1	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
7	addr[1]	Input	PIN_51	4	B4_N1	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
8	addr[0]	Input	PIN_48	4	B4_N1	3.3-V LVTTL (default)		addr[7..0]	24mA (default)	
9	clk	Input	PIN_17	1	B1_N0	3.3-V LVTTL (default)			24mA (default)	
10	op_done	Output	PIN_101	3	B3_N0	3.3-V LVTTL (default)			24mA (default)	
11	read_data[7]	Output	PIN_86	3	B3_N1	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
12	read_data[6]	Output	PIN_81	3	B3_N1	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
13	read_data[5]	Output	PIN_80	3	B3_N1	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
14	read_data[4]	Output	PIN_79	3	B3_N1	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
15	read_data[3]	Output	PIN_75	3	B3_N1	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
16	read_data[2]	Output	PIN_74	3	B3_N1	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
17	read_data[1]	Output	PIN_73	3	B3_N1	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
18	read_data[0]	Output	PIN_72	4	B4_N0	3.3-V LVTTL (default)		read_data[7..0]	24mA (default)	
19	read_op	Input	PIN_42	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
20	rstn	Input	PIN_40	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
21	scl	Output	PIN_45	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
22	sda	Bidir	PIN_47	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
23	write_data[7]	Input	PIN_71	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
24	write_data[6]	Input	PIN_70	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
25	write_data[5]	Input	PIN_69	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
26	write_data[4]	Input	PIN_67	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
27	write_data[3]	Input	PIN_65	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
28	write_data[2]	Input	PIN_64	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
29	write_data[1]	Input	PIN_63	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
30	write_data[0]	Input	PIN_60	4	B4_N0	3.3-V LVTTL (default)		write_data[7..0]	24mA (default)	
31	write_op	Input	PIN_41	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
32	<<new node>>									

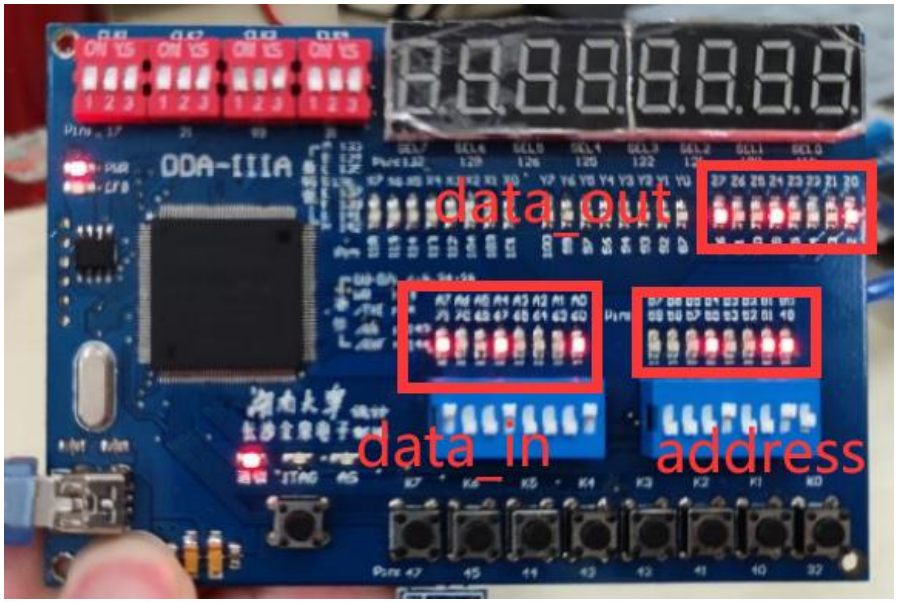
Quartus 下载：



下板验证：



下板验证解读：



三、实验总结

通过本次实验，我学习到了许多知识，同时也暴露了一些问题。

我详细学习了 verilog 语言的基本语法以及使用 verilog 语言描述时序逻辑和组合逻辑的方式。同时，我也了解了如何使用 verilog 编写有限状态机并练习编写了简单的有限状态机。了解了 test_bench 的编写以及熟悉了使用 ModelSim 进行波形仿真。理解了 I2C 接口协议以及 I2C 协议下 SCL，SDA 数据是如何传输的。我详细深入地研究了 I2C 协议，对于给出的参考代码能够基本理解，也能够对应波形仿真结果解释 I2C 协议的数据传输，最后将代码下载至 FPGA 开发板，验证了 I2C 协议正常工作。

但是，我对于实验中 I2C 协议的 verilog 实现的一些具体细节理解的还不够深刻，使用 verilog 语言编写有限状态机的能力还比较基础，还需要后续的练习和进一步的学习。