I2C

信安2101 孙照海 202109070105

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（b=1），无课自习，体温异常(T>37)则留在寝室休息

S2：12 节上课，若 34 有课则继续去上课（c=1），否则自习

S3：12 节自习，若 34 有课则去上课（c=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：吃午餐并午休，午休结束后 56 节上课

S7：56 节上课，课后若有会则开会（d=1），无会晴天（e=1）跑步，雨天自习

S8：78 节跑步，跑步后去吃晚餐

S9：78 节开会

S10：78 节自习

S11：晚餐，吃完后自习，若周五直接回寝室(F=1)

S12：晚自习，结束后若有快递（g=1）去取快递，否则直接回寝休息

S13：取快递，取完后回寝休息

输出 pos 为位置：

00：宿舍园区

01：教学楼

10：体育场

11：快递点

2、设计代码说明

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 fsm1(
    input clk,
    input a,
    input b,
    input c,
    input d,
    input e,
    input F,
    input g,
    input reg[6:0] T,
    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;
```

下一状态判断：

```
always @(*)
case(state)
s0:
  if(a) next_st=s1;
  else next_st=s0;
s1:
  if(T>=37) next_st=s0;
  else if(b) next_st=s2;
  else next_st=s3;
s2:
  if(c) next_st=s4;
  else next_st=s5;
s3:
  if(c) next_st=s4;
  else next_st=s5;
s4: next_st=s6;
s5: next_st=s6;
s6: next_st=s7;
s7:
  if(d) next_st=s9;
  else if(e) next_st=s8;
  else next_st=s10;
s8: next_st=s11;
s9: next_st=s11;
s10:next_st=s11;
s11:
    if(F) next_st=s0;
    else next_st=s12;
s12:
    if(g) next_st=s13;
    else next_st=s0;
s13:  next_st=s0;
default:  next_st=s0;
endcase
```

状态更新与输出：

```
always @(posedge clk) state<=next_st;

always @(*)
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'b01;
s8: pos=2'b10;
s9: pos=2'b01;
s10:  pos=2'b01;
s11:  pos=2'b00;
s12:  pos=2'b01;
s13:  pos=2'b11;
default:  pos=2'b00;
endcase

endmodule
```

test_bench 代码如下：

第一次从 s0 开始，设定状态变化为：

s0-> s1->s2->s4->s6->s8/s10(由随机产生的 e 决定)->s11->s12->s13->s0

第二次从 s0 开始，设置 T=37，状态在 s0 和 s1 之间转换

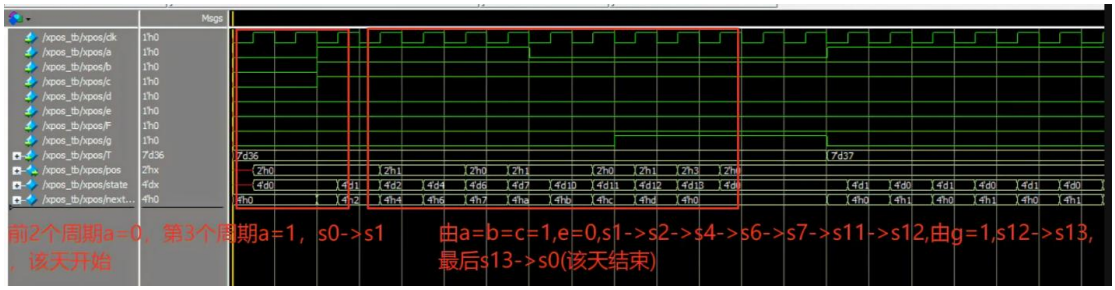
14 个周期后设置 Y=36，状态变化为：

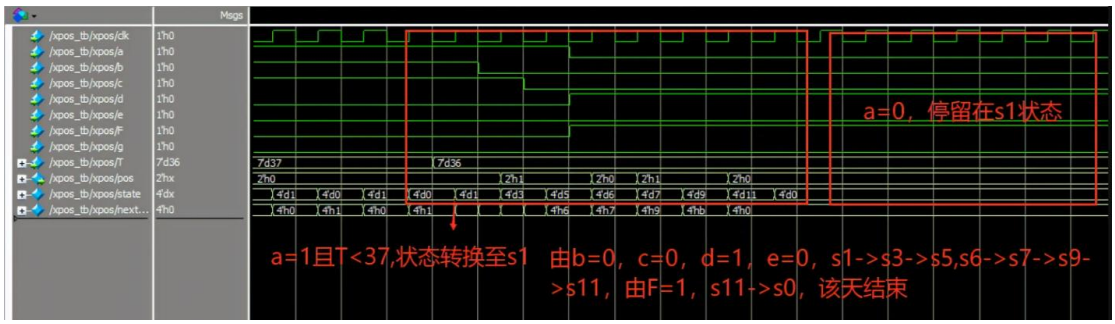
s0-> s1->s3->s5->s6->s9->s11->s0

此后设置 a=0，停留在 s0 状态

```
`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'h7,s8=4'h8,s9=4'h9,s10=4'ha,s11=4'hb,s12=4'hc,s13=4'hd;
reg [3:0] state,next_st;
reg [6:0] T;
reg a,clk,b,c,d,e,F,g;
wire [1:0] pos;
pos xpos(clk,a,b,c,d,e,F,g,T,pos);
initial clk=0;
always #50 clk=~clk;
initial
begin
  a=0;b=0;c=0;d=0;e=0;F=0;e=0;g=0;T=36;
  #1
  #200
  a=1;
  b=1;
  c=1;
  #500
  e=$random;
  a=0;
  #200
  g=1;
  #500
  g=0;
  a=1;
  T=37;
  #1400
  T=36;
  #100
  b=0;
  #100
  c=0;
  #100
  d=1;
  F=1;
  a=0;
  repeat(1024) @(posedge clk);
  $stop;
```

3、仿真波形说明





根据波形，可以验证状态转换与状态图一致，且 pos 输出正确。

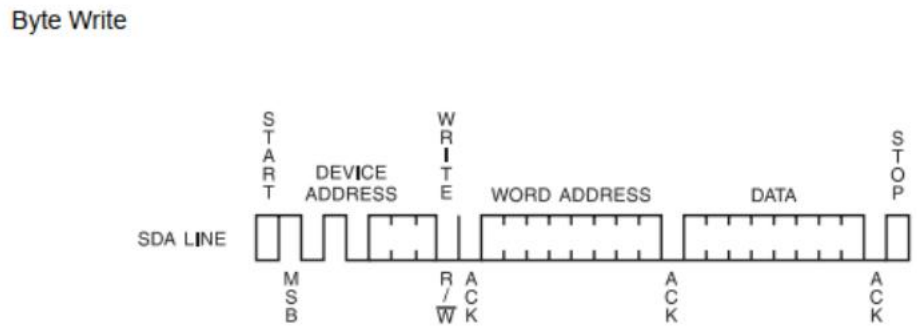
二、EEPROM 读写代码设计及仿真

I2C 通讯协议是一种简单、双向二线制同步串行总线，只需要两根线即可在连接于总线上的器件之间传送信息。输出信号：通常由当前状态和输入信号决定。

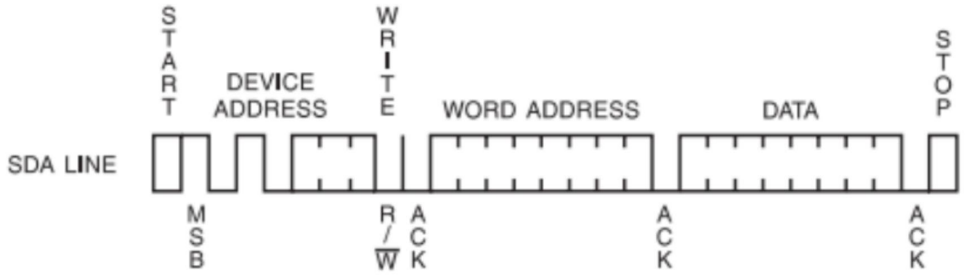
信号说明：

clk：时钟信号
rstn：复位信号，低有效；
write_op：写命令，高有效；当发出 write_op 命令，必须 等待 op_done 为高才可将 write_op 清零；
write_data[7:0]：写数据；
reade_op：读命令，高有效；当发出 read_op 命令，必须等待 op_done 为高才可将 read_op 清零；
read_data[7:0：读数据；
addr[7:0]：读写地址；
op_done：读写操作完成；
scl：I2C 协议的 SCL 信号；
sda：I2C 协议的 SDA 信号；

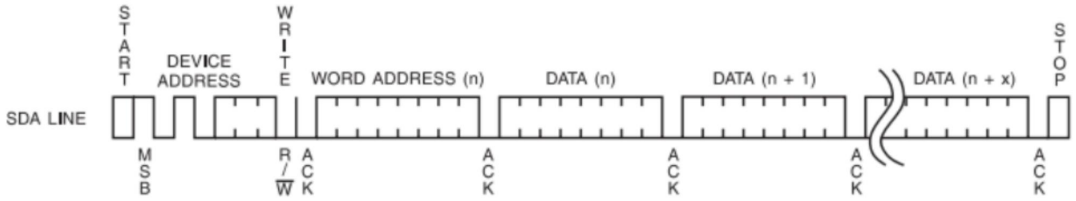
进行写入：



- **I2C** 设备单字节写操作时序如下图所示：

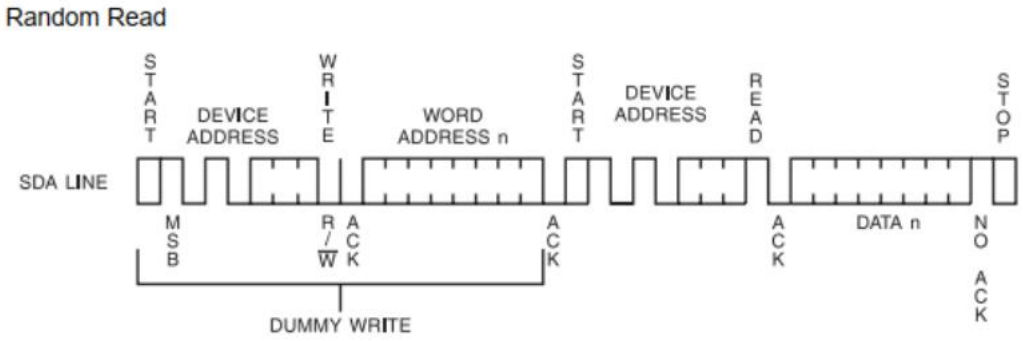


- 流程：
  - **START +** 器件地址（**8** 位，写 **/LSB=0**） **+ACK+** 地址（**8** 位） **+ACK+** 数据（**8** 位） **+ACK+STOP**
- **8** 位地址、数据均先发送第 **7bit(MSB),** 最后发送第 **0bit** （**LSB);**
  - **I2C** 页内多字节写操作时序如下图所示：

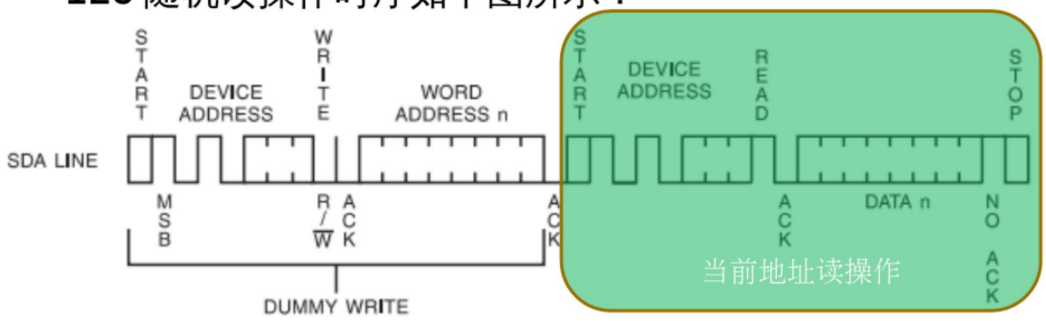


  - 流程：
    - **START +** 器件地址（**8** 位，写 **LSB=0**） **+ACK+** 地址（**8** 位） **+ACK+** 数据 **n** （**8** 位） **+ACK+** 数据 **(n+1)+ACK...+** 数据 **(n+x)+ACK+STOP**
  - 多字节写操作只能在页（**page**）内进行。 **AT24C02** 每页为 **8** 字节。
  - 当 **WORD ADDRESS=8'h00,** 连续写 **8** 个字节则依次将数据写入 **8'h00~8'h1f** 地址；

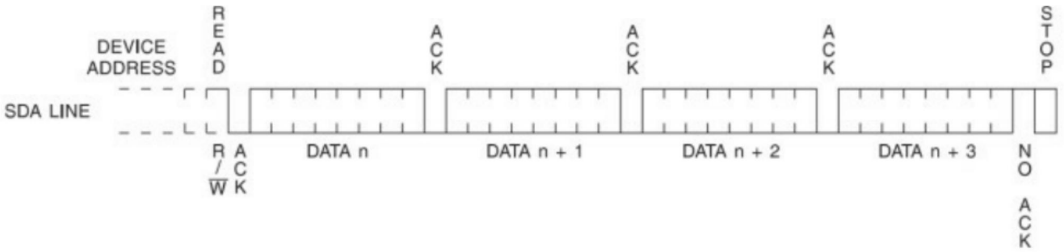
随机读操作:



- **I2C** 随机读操作时序如下图所示：



- 流程：
  - **START +** 器件地址（**8** 位，写 **/LSB=0**） **+ACK+** 地址（**8** 位） **+ACK+**
  - **START+** 器件地址（**8** 位，读 **/LSB=1**） **+ACK +** 数据（**8** 位，从机发出） **+NO ACK** （**SDA=1**） **+STOP**
- **DUMMY WRITE:** 用于写入读操作的地址。
- **I2C** 顺序读操作时序如下图所示：



- 顺序读操作就是从上一地址开始顺序读取。假如要读取 **n** 字节连续数据，只需写入要读取第一个字节数据的存储地址，就可以实现连续 **n** 字节数据的顺序读取。

代码分析:

端口声明:

```
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       )
);
```

状态:

```
parameter  IDLE       = 8'h00,
           WAIT_WTICK0= 8'h01,
           WAIT_WTICK1= 8'h02,
           W_START    = 8'h03,
           W_DEVICE7  = 8'h04,
           W_DEVICE6  = 8'h05,
           W_DEVICE5  = 8'h06,
           W_DEVICE4  = 8'h07,
           W_DEVICE3  = 8'h08,
           W_DEVICE2  = 8'h09,
           W_DEVICE1  = 8'h0a,
           W_DEVICE0  = 8'h0b,
           W_DEVACK   = 8'h0c,
           W_ADDRES7  = 8'h0d,
           W_ADDRES6  = 8'h0e,
           W_ADDRES5  = 8'h0f,
           W_ADDRES4  = 8'h10,
           W_ADDRES3  = 8'h11,
           W_ADDRES2  = 8'h12,
           W_ADDRES1  = 8'h13,
           W_ADDRES0  = 8'h14,
           W_AACK     = 8'h15,
           W_DATA7    = 8'h16,
           W_DATA6    = 8'h17,
           W_DATA5    = 8'h18,
           W_DATA4    = 8'h19,
           W_DATA3    = 8'h1a,
           W_DATA2    = 8'h1b,
           W_DATA1    = 8'h1c,
           W_DATA0    = 8'h1d,
           W_DACK     = 8'h1e,
           WAIT_WTICK3= 8'h1f,
           R_START    = 8'h20,
           R_DEVICE7  = 8'h21,
           R_DEVICE6  = 8'h22,
           R_DEVICE5  = 8'h23,
           R_DEVICE4  = 8'h24,
           R_DEVICE3  = 8'h25,
           R_DEVICE2  = 8'h26,
```

```
            R_DEVICE1   = 8'h27,
            R_DEVICE0   = 8'h28,
            R_DACK      = 8'h29,
            R_DATA7     = 8'h2a,
            R_DATA6     = 8'h2b,
            R_DATA5     = 8'h2c,
            R_DATA4     = 8'h2d,
            R_DATA3     = 8'h2e,
            R_DATA2     = 8'h2f,
            R_DATA1     = 8'h30,
            R_DATA0     = 8'h31,
            R_NOACK     = 8'h32,
            S_STOP      = 8'h33,
            S_STOP0     = 8'h34,
            S_STOP1     = 8'h35,
            W_OPOVER    = 8'h36;

reg [7:0] i2c, next_i/*synthesis preserve*/;
```

SCL 同步

```
//SCL
assign clr_scl = scl_ls                 &
                 (i2c != IDLE       ) &
                 (i2c != WAIT_WTICK0) &
                 (i2c != WAIT_WTICK1) &
                 (i2c != W_START    ) &
                 (i2c != R_START    ) &
                 (i2c != S_STOP0    ) &
                 (i2c != S_STOP1    ) &
                 (i2c != W_OPOVER   ) ;

always @(posedge clk or negedge rstn)
if(!rstn       ) scl <=1'b1;
else if(clr_scl) scl <=1'b0;
else if(scl_hs ) scl <=1'b1;
```

状态更新

```
always @(posedge clk or negedge rstn)
if(!rstn       ) scl <=1'b1;
else if(clr_scl) scl <=1'b0;
else if(scl_hs ) scl <=1'b1;
```

读写命令的判断

```
// write cycle(5ms)
// 6MHZ = 166ns, 5ms/166ns = 30120 = 0x75A8
reg [15:0] d5ms_cnt;
always @(posedge clk or negedge rstn)
if(!rstn             ) d5ms_cnt <= 0;
else if(i2c==W_OPOVER) d5ms_cnt <= d5ms_cnt+1'b1;
else                   d5ms_cnt <= 0;

assign d5ms_over = (rd_op) ? (d5ms_cnt=='h1f) : (d5ms_cnt=='h75A8);


endmodule
```

下一状态判断

```
always @(*)
case(i2c)

  IDLE       : begin next_i = IDLE       ; if(wr_op|rd_op) next_i = WAIT_WTICK0;end

  //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

  //DECIVE ADDRESS(1010_000)+WRITE(0)
  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[7:0]
  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;
                     else if(scl_tick&rd_op ) next_i = WAIT_WTICK3;
               end
```

```
151     //WRITE DATA[7:0]
152     W_DATA7     : begin next_i = W_DATA7     ; if(scl_tick) next_i = W_DATA6     ;end
153     W_DATA6     : begin next_i = W_DATA6     ; if(scl_tick) next_i = W_DATA5     ;end
154     W_DATA5     : begin next_i = W_DATA5     ; if(scl_tick) next_i = W_DATA4     ;end
155     W_DATA4     : begin next_i = W_DATA4     ; if(scl_tick) next_i = W_DATA3     ;end
156     W_DATA3     : begin next_i = W_DATA3     ; if(scl_tick) next_i = W_DATA2     ;end
157     W_DATA2     : begin next_i = W_DATA2     ; if(scl_tick) next_i = W_DATA1     ;end
158     W_DATA1     : begin next_i = W_DATA1     ; if(scl_tick) next_i = W_DATA0     ;end
159     W_DATA0     : begin next_i = W_DATA0     ; if(scl_tick) next_i = W_DACK      ;end
160
161     // ACK
162     W_DACK      : begin next_i = W_DACK      ; if(scl_tick) next_i = S_STOP      ;end
163
164
165     //Current Address Read
166     //START: SCL=1, SDA=1->0(scl_lc)
167     WAIT_WTICK3: begin next_i = WAIT_WTICK3; if(scl_tick) next_i = R_START     ;end
168     R_START     : begin next_i = R_START     ; if(scl_tick) next_i = R_DEVICE7   ;end
169
170     //DECIVE ADDRESS(1010_000)+READ(1)
171     R_DEVICE7   : begin next_i = R_DEVICE7   ; if(scl_tick) next_i = R_DEVICE6   ;end
172     R_DEVICE6   : begin next_i = R_DEVICE6   ; if(scl_tick) next_i = R_DEVICE5   ;end
173     R_DEVICE5   : begin next_i = R_DEVICE5   ; if(scl_tick) next_i = R_DEVICE4   ;end
174     R_DEVICE4   : begin next_i = R_DEVICE4   ; if(scl_tick) next_i = R_DEVICE3   ;end
175     R_DEVICE3   : begin next_i = R_DEVICE3   ; if(scl_tick) next_i = R_DEVICE2   ;end
176     R_DEVICE2   : begin next_i = R_DEVICE2   ; if(scl_tick) next_i = R_DEVICE1   ;end
177     R_DEVICE1   : begin next_i = R_DEVICE1   ; if(scl_tick) next_i = R_DEVICE0   ;end
178     R_DEVICE0   : begin next_i = R_DEVICE0   ; if(scl_tick) next_i = R_DACK      ;end
179
180     // ACK
181     R_DACK      : begin next_i = R_DACK      ; if(scl_tick) next_i = R_DATA7     ;end
182
183     //READ DATA[7:0], SDA:input
184     R_DATA7     : begin next_i = R_DATA7     ; if(scl_tick) next_i = R_DATA6     ;end
185     R_DATA6     : begin next_i = R_DATA6     ; if(scl_tick) next_i = R_DATA5     ;end
186     R_DATA5     : begin next_i = R_DATA5     ; if(scl_tick) next_i = R_DATA4     ;end
187     R_DATA4     : begin next_i = R_DATA4     ; if(scl_tick) next_i = R_DATA3     ;end
188     R_DATA3     : begin next_i = R_DATA3     ; if(scl_tick) next_i = R_DATA2     ;end
189     R_DATA2     : begin next_i = R_DATA2     ; if(scl_tick) next_i = R_DATA1     ;end
190     R_DATA1     : begin next_i = R_DATA1     ; if(scl_tick) next_i = R_DATA0     ;end
191     R_DATA0     : begin next_i = R_DATA0     ; if(scl_tick) next_i = R_NOACK     ;end
192
193     // NO ACK
194     R_NOACK     : begin next_i = R_NOACK     ; if(scl_tick) next_i = S_STOP      ;end
195
```

```
196     // STOP
197     S_STOP      : begin next_i = S_STOP      ; if(scl_tick) next_i = S_STOP0     ;end
198     S_STOP0     : begin next_i = S_STOP0     ; if(scl_tick) next_i = S_STOP1     ;end
199     S_STOP1     : begin next_i = S_STOP1     ; if(scl_tick) next_i = W_OPOVER    ;end
200
201     // WAIT write_op=0,read_op=0;
202     W_OPOVER    : begin next_i = W_OPOVER    ; if(d5ms_over)next_i = IDLE        ;end
203     default     : begin next_i = IDLE        ;                                    end
204   endcase
205
206
```

```
207   //SCL
208   assign clr_scl = scl_ls              &
209                    (i2c != IDLE       ) &
210                    (i2c != WAIT_WTICK0) &
211                    (i2c != WAIT_WTICK1) &
212                    (i2c != W_START    ) &
213                    (i2c != R_START    ) &
214                    (i2c != S_STOP0    ) &
215                    (i2c != S_STOP1    ) &
216                    (i2c != W_OPOVER   ) ;
217
218   always @(posedge clk or negedge rstn)
219   if(!rstn       ) scl <=1'b1;
220   else if(clr_scl) scl <=1'b0;
221   else if(scl_hs ) scl <=1'b1;
222
```

SCL 同步实现

```
207  //SCL
208  assign clr_scl = scl_ls              &
209                   (i2c != IDLE       ) &
210                   (i2c != WAIT_WTICK0) &
211                   (i2c != WAIT_WTICK1) &
212                   (i2c != W_START    ) &
213                   (i2c != R_START    ) &
214                   (i2c != S_STOP0    ) &
215                   (i2c != S_STOP1    ) &
216                   (i2c != W_OPOVER   ) ;
217
218  always @(posedge clk or negedge rstn)
219  if(!rstn       ) scl <=1'b1;
220  else if(clr_scl) scl <=1'b0;
221  else if(scl_hs ) scl <=1'b1;
```

SDA

```
223  //SDA
224  reg [7:0] i2c_reg;
225
226  assign start_clr  = scl_lc & ((i2c == W_START )|(i2c == R_START ));
227  assign ld_wdevice = scl_lc & (i2c == W_DEVICE7);
228  assign ld_waddres = scl_lc & (i2c == W_ADDRES7);
229  assign ld_wdata   = scl_lc & (i2c == W_DATA7  );
230  assign ld_rdevice = scl_lc & (i2c == R_DEVICE7);
231  assign noack_set  = scl_lc & (i2c == R_NOACK  );
232  assign stop_clr   = scl_lc & (i2c == S_STOP   );
233  assign stop_set   = scl_lc & ((i2c == S_STOP0 )|(i2c == WAIT_WTICK3));
234
235  assign i2c_rlf = scl_lc & (
236                   (i2c == W_DEVICE6 ) |
237                   (i2c == W_DEVICE5 ) |
238                   (i2c == W_DEVICE4 ) |
239                   (i2c == W_DEVICE3 ) |
240                   (i2c == W_DEVICE2 ) |
241                   (i2c == W_DEVICE1 ) |
242                   (i2c == W_DEVICE0 ) |
243                   (i2c == W_ADDRES6 ) |
244                   (i2c == W_ADDRES5 ) |
245                   (i2c == W_ADDRES4 ) |
246                   (i2c == W_ADDRES3 ) |
247                   (i2c == W_ADDRES2 ) |
248                   (i2c == W_ADDRES1 ) |
249                   (i2c == W_ADDRES0 ) |
250                   (i2c == W_DATA6   ) |
251                   (i2c == W_DATA5   ) |
252                   (i2c == W_DATA4   ) |
253                   (i2c == W_DATA3   ) |
254                   (i2c == W_DATA2   ) |
255                   (i2c == W_DATA1   ) |
256                   (i2c == W_DATA0   ) |
257                   (i2c == R_DEVICE6 ) |
258                   (i2c == R_DEVICE5 ) |
259                   (i2c == R_DEVICE4 ) |
260                   (i2c == R_DEVICE3 ) |
261                   (i2c == R_DEVICE2 ) |
262                   (i2c == R_DEVICE1 ) |
263                  (i2c == R_DEVICE0 ) );
```

```
always @(posedge clk or negedge rstn)
if(!rstn          ) i2c_reg <= 8'hff;
else if(start_clr ) i2c_reg <= 8'h00;       // "start"
else if(ld_wdevice) i2c_reg <= {4'b1010,3'b000,1'b0}; // DECIVE ADDRESS(1010_000)+WRITE(0)
else if(ld_waddres) i2c_reg <= addr;        // WORD ADDRESS[7:0]
else if(ld_wdata  ) i2c_reg <= write_data; // WRITE DATA[7:0]
else if(ld_rdevice) i2c_reg <= {4'b1010,3'b000,1'b1}; // DECIVE ADDRESS(1010_000)+READ(1)
else if(noack_set ) i2c_reg <= 8'hff;       // "no ack"
else if(stop_clr  ) i2c_reg <= 8'h00;       // "stop 0"
else if(stop_set  ) i2c_reg <= 8'hff;       // "stop 1"
else if(i2c_rlf   ) i2c_reg <= {i2c_reg[6:0],1'b0};

assign sda_o = i2c_reg[7];

assign clr_sdaen = (i2c == IDLE    ) |
                   (
                   scl_lc & (
                   (i2c == W_DEVACK) |
                   (i2c == W_AACK  ) |
                   (i2c == W_DACK  ) |
                   (i2c == R_DACK  ) |
                   (i2c == R_DATA7 ) ) ) ;

assign set_sdaen = scl_lc & (
                   (i2c == WAIT_WTICK0) |
                   (i2c == W_ADDRES7  ) |
                   (i2c == W_DATA7    ) |
                   (i2c == WAIT_WTICK3) |
                   (i2c == S_STOP     ) |
                   //(i2c == R_DATA7    ) |
                   (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;
```

操作结束，等待

```
// op_done
assign op_done = d5ms_over;

// Write Cycle(5ms)
// 6MHZ = 166ns, 5ms/166ns = 30120 = 0x75A8
reg [15:0] d5ms_cnt;
always @(posedge clk or negedge rstn)
if(!rstn              ) d5ms_cnt <= 0;
else if(i2c==W_OPOVER) d5ms_cnt <= d5ms_cnt+1'b1;
else                    d5ms_cnt <= 0;

assign d5ms_over = (rd_op) ? (d5ms_cnt=='h1f) : (d5ms_cnt=='h75A8);


endmodule

```