FPGA I2C 接口实现

# 实验目的

1. 掌握状态机描述方法；
2. 能编写TestBench；
3. 能用ModelSim进行仿真；
4. 掌握I2C接口协议；
5. 能较好的理解参考设计文件和测试文件；

# 实验内容

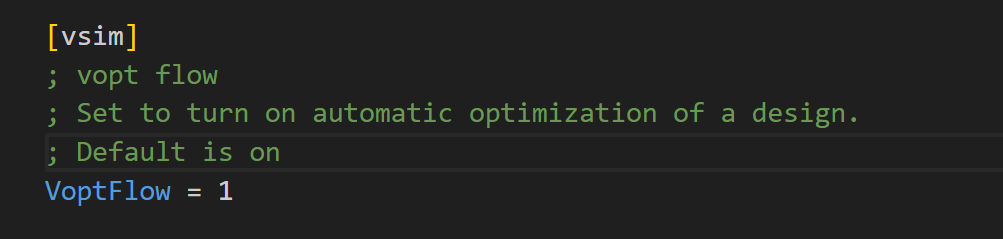
## 如何描述组合电路、时序电路、状态机？如何编写TestBench？

## ModelSim工具的使用步骤；

## 三．EEPROM读写代码分析；

## 四．实验总结；

# 实验过程



# 一．TestBench及ModelSim仿真

按示例文件编写fsm.v和fsm\_tb.v:

### 状态描述：

state0:

复位（reset = 1）时，处于state0；

当start = 1时，进入state1；

state1：

跳转至state2；

state2：

当skip3 = 1时，跳转至state0；

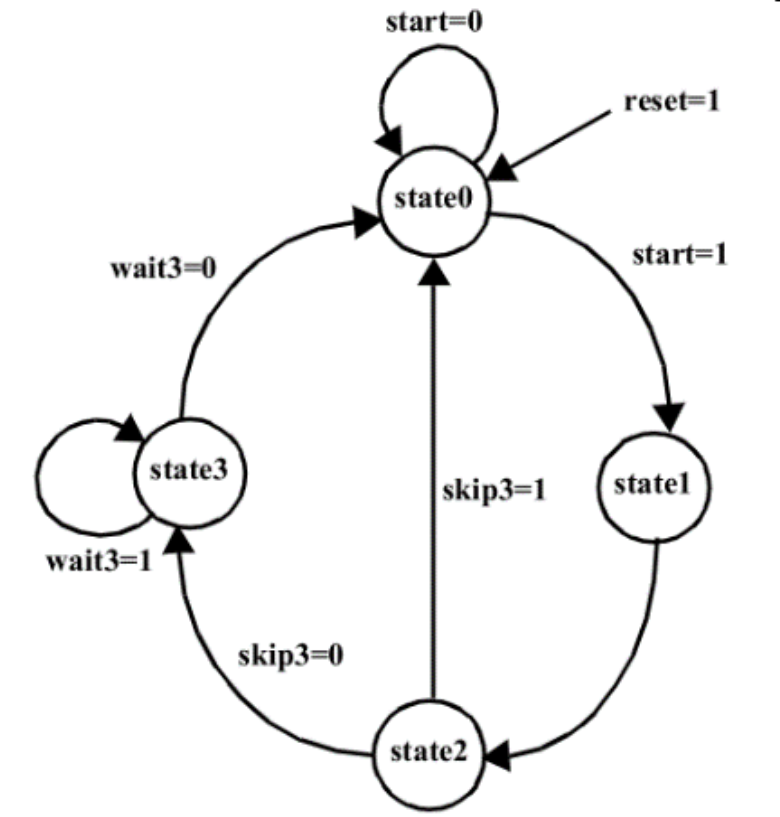
当skip3 = 0时，跳转至state3；

state3：

当wait3 = 0时，跳转至state0；

当wait3 = 1时，跳转至state3：

### 状态图：



### 状态机描述代码：

**fsm.v:**

**//所有管脚，包括时钟、输出**

module fsm(

input clk,

input reset,

input start,

input skip3,

input wait3,

output reg[2:0] zot

);

//状态定义，用state代替数字

parameter state0 = 2'h0,

state1 = 2'h1,

state2 = 2'h2,

state3 = 2'h3;

reg [1:0] state, next\_st;

//下一状态判断

always @ (\*)

case(state)

state0: if(start) next\_st = state1;

else next\_st = state0;

state1: next\_st = state2;

state2: if(skip3) next\_st = state0;

else next\_st = state3;

state3: if(wait3) next\_st = state3;

else next\_st = state0;

default: next\_st = state0;

endcase

//当前状态

always @ (posedge clk or posedge reset)

if(reset) state <= state0;

else state <= next\_st;

//输出信号

always @ (\*)

case(state)

state0: zot = 3'b000;

state1: zot = 3'b101;

state2: zot = 3'b111;

state3: zot = 3'b001;

default: zot = 3'b000;

endcase

endmodule

### 测试代码：

fsm\_tb.v:

//仿真时间单位、精度

`timescale 1 ns / 100 ps //time unit = 1ns

module fsm\_tb;

reg clk, reset, start, skip3, wait3;

wire [2:0] zot;

//实例化fsm为dut1

fsm dut1(clk, reset, start, skip3, wait3, zot);

//时钟信号的产生

always #50 clk = ~clk; //100 MHz

//reset信号产生

initial begin

clk = 0; reset = 0;

#1 reset = 1;

#200 reset = 0;

end

initial begin

start = 0; skip3 = 0; wait3 = 0;

#1

#300 start = 1;

//由 dut1.state3 改为 3'b001

wait(zot == 3'b001);

wait(zot == 3'b001);

#400 wait3 = 1;

#200 wait3 = 0;

//由 dut1.state3 改为 3'b001

wait(zot == 3'b001);

skip3 = 1;

//由 dut1.state2 改为 3'b111

wait(zot == 3'b111);

repeat (1024) begin

@(posedge clk); // 等待上升沿

skip3 = $random; // 每个周期赋值

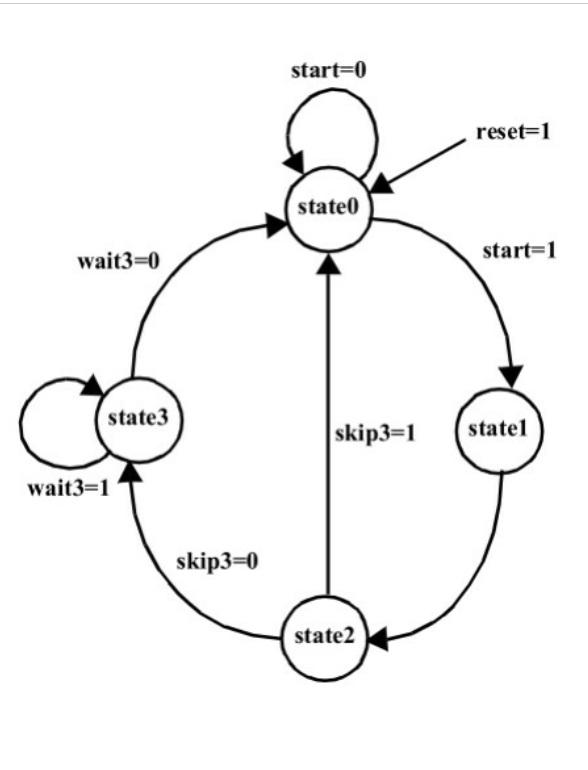
end

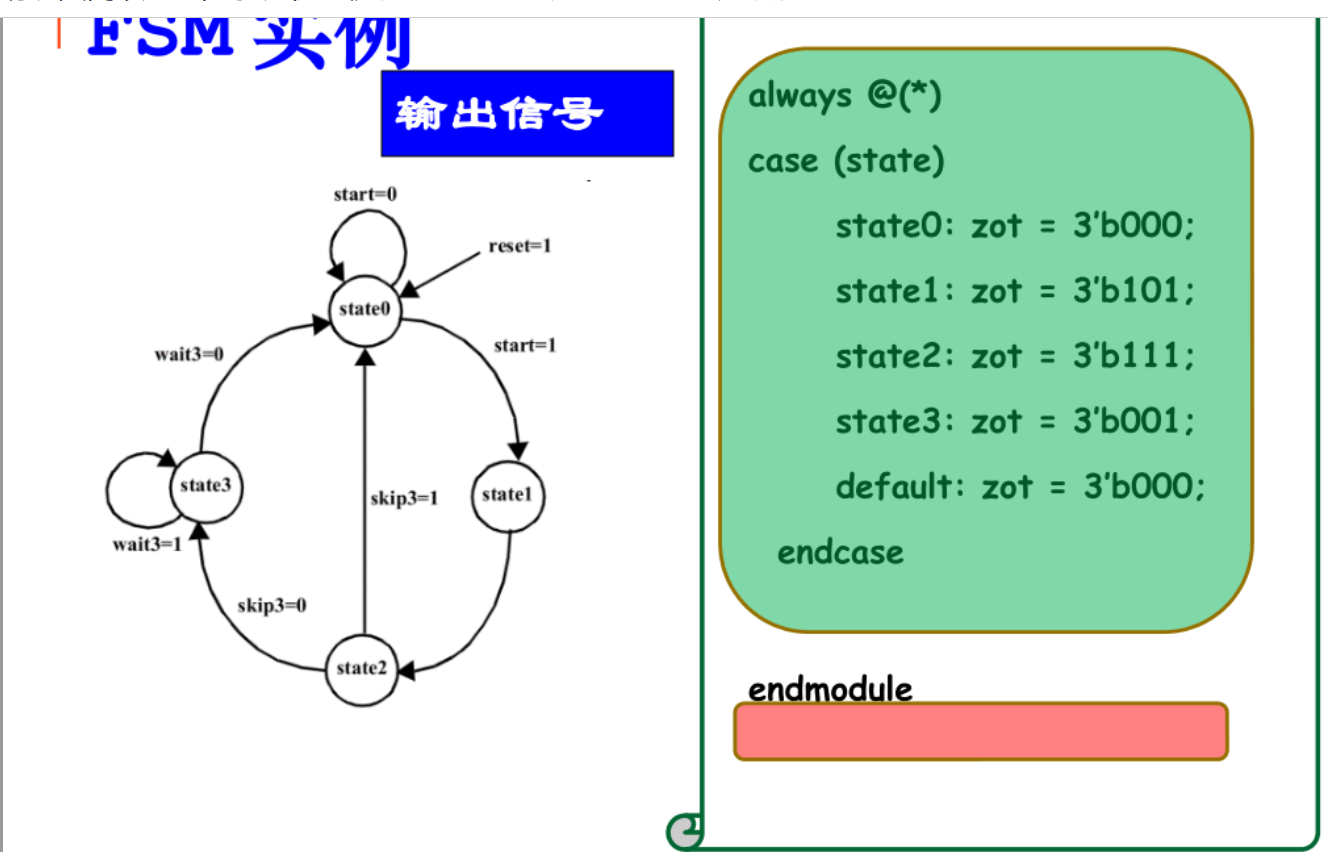
$stop;

end

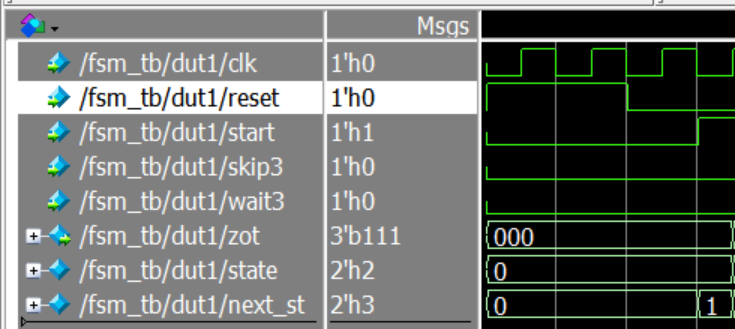
endmodule

### 仿真结果

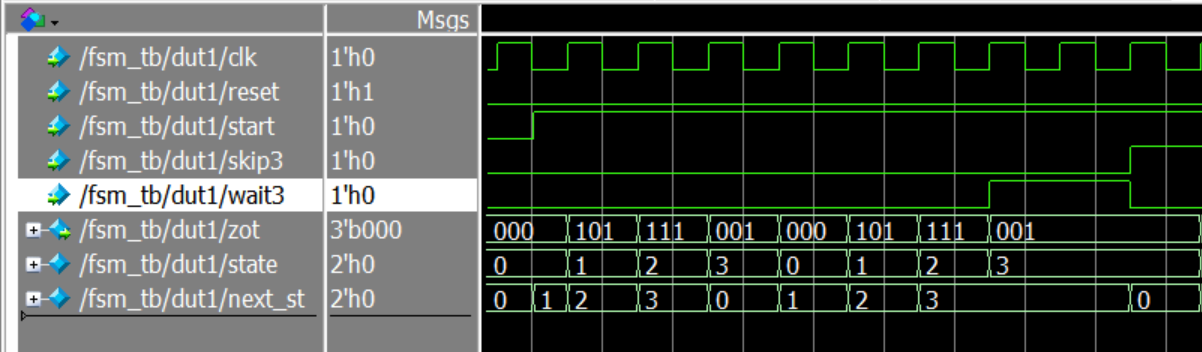




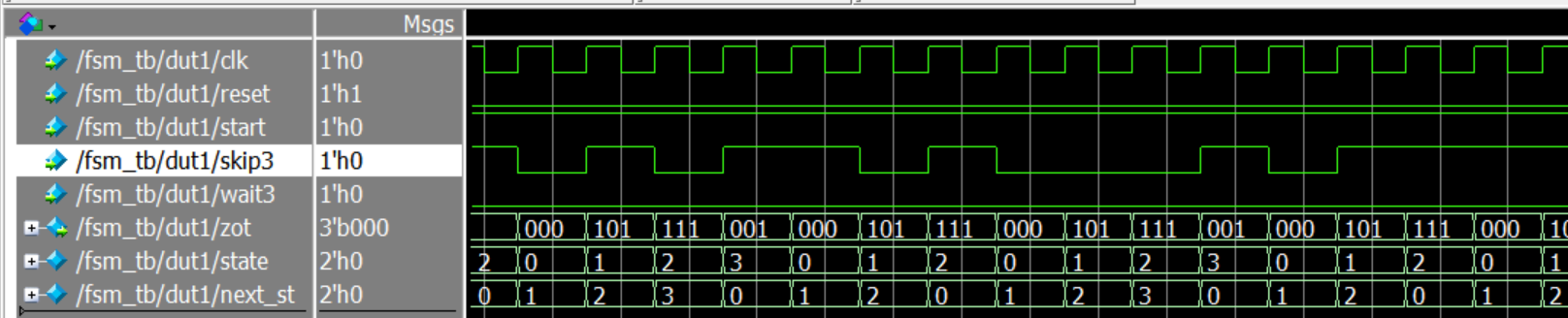
reset信号产生：



state3之后wait3的变化：



skip3随机赋值：



# 二．EEPROM 读写代码设计及仿真

### 代码说明：

输入：

clk：时钟信号

reset：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

addr[7:0]：读写地址

read\_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；

输出：

read\_data：读数据

op\_done：操作结束

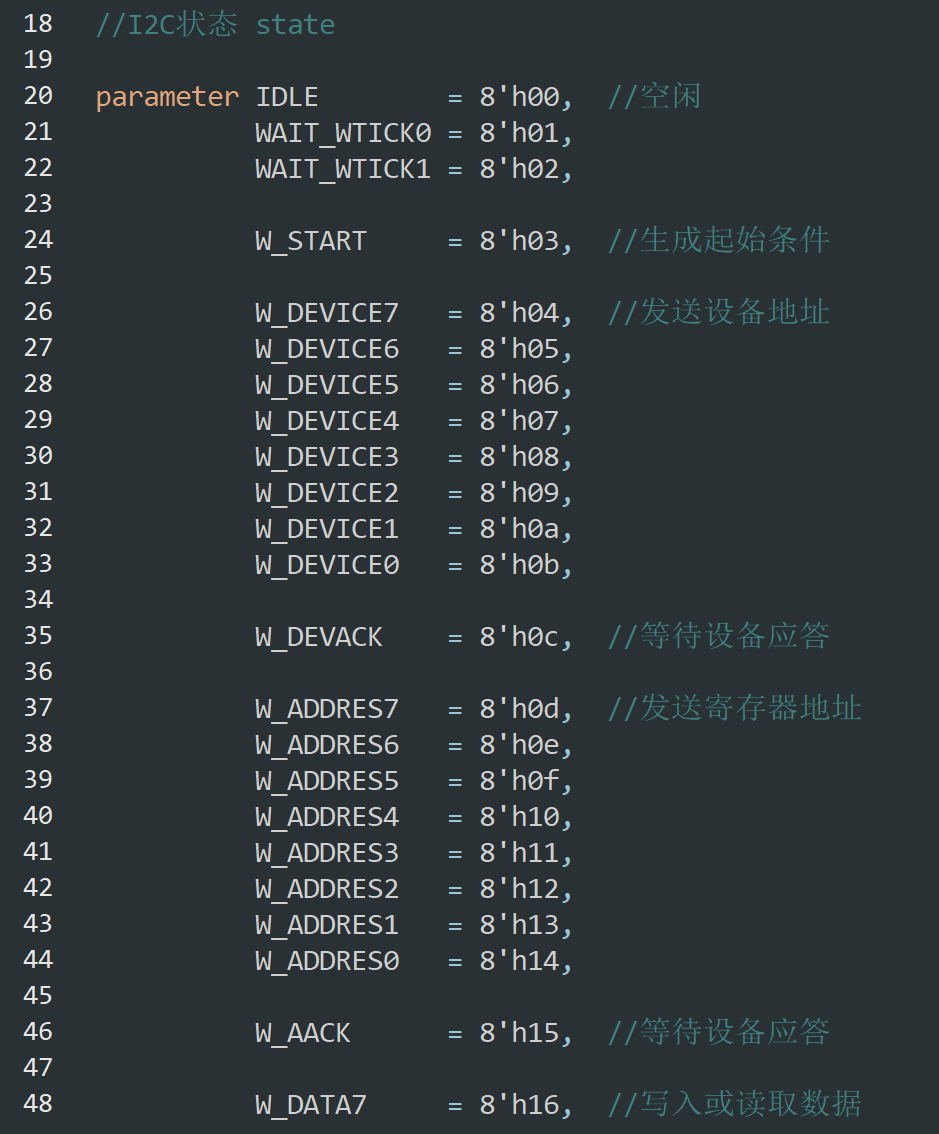
I2C 协议信号：

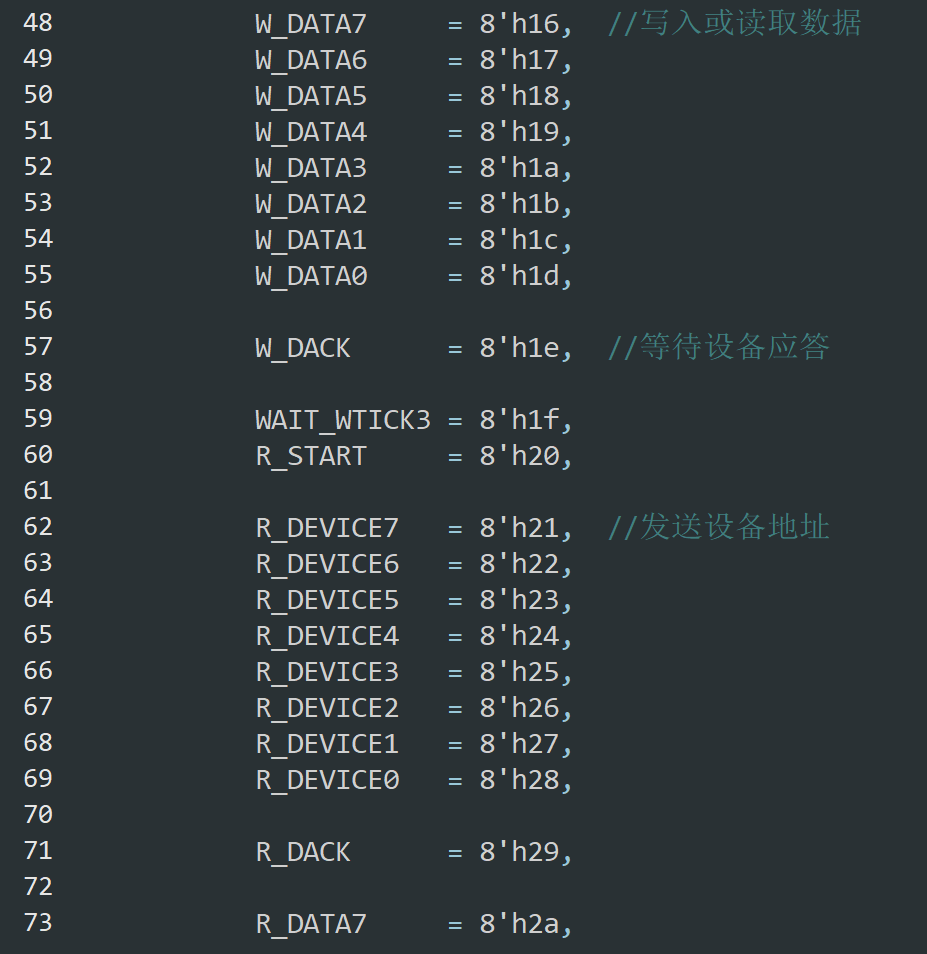
scl：I2C协议的SCL信号；

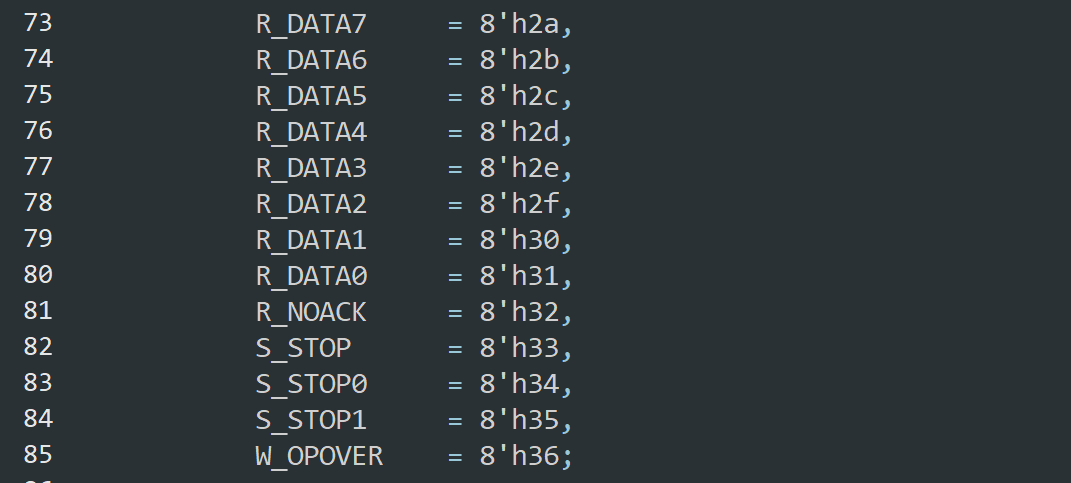
sda：I2C协议的SDA信号；



使用 8 位 16 进制数表示所有状态，共 55 个：





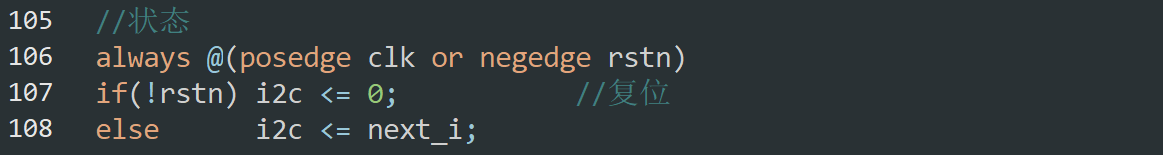


使用计数器对时钟周期计数实现scl周期，一个scl周期是30个时钟周期；

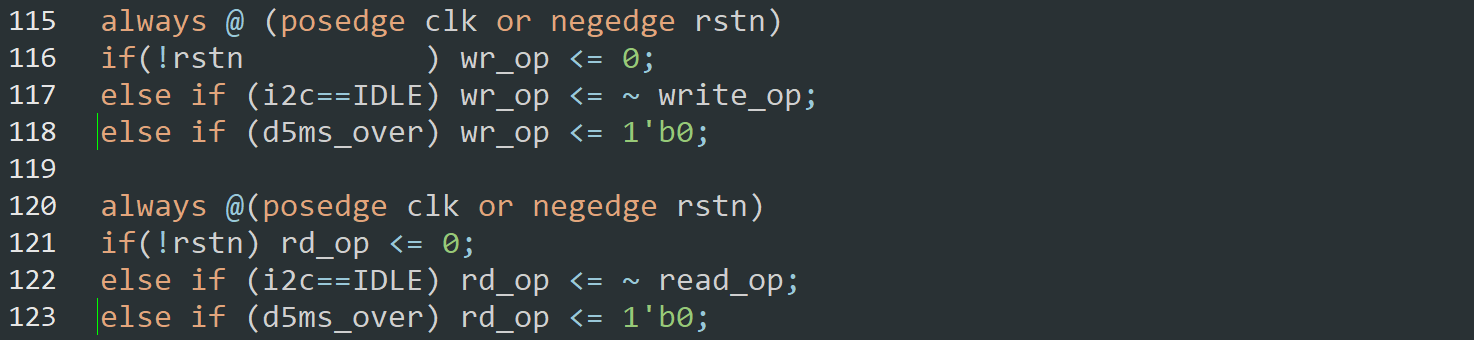
分出5个不同节拍，用于不同之处；



下一状态的更新：

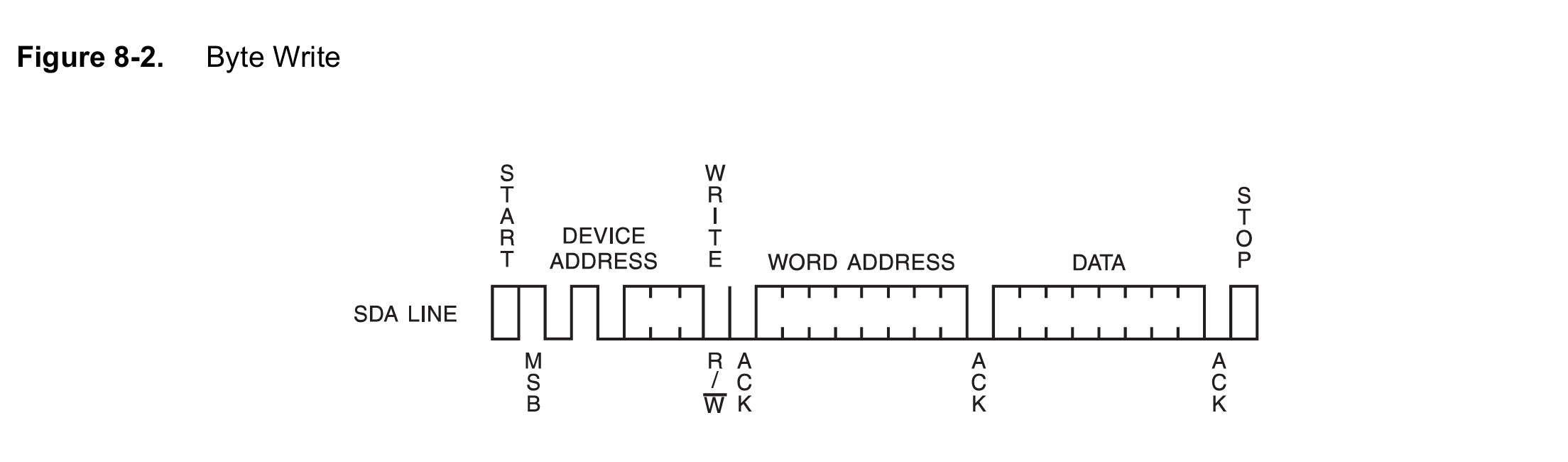


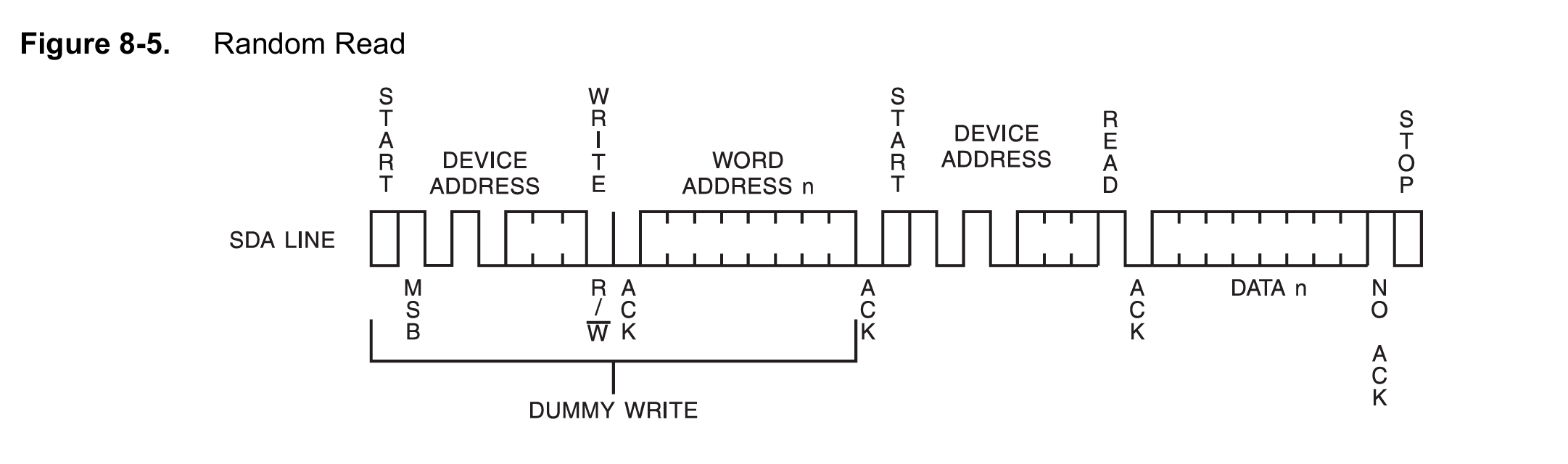
使用 wr\_op 和 rd\_op 将输入信号 write\_op，read\_op 表示的读写命令用高电平表示：



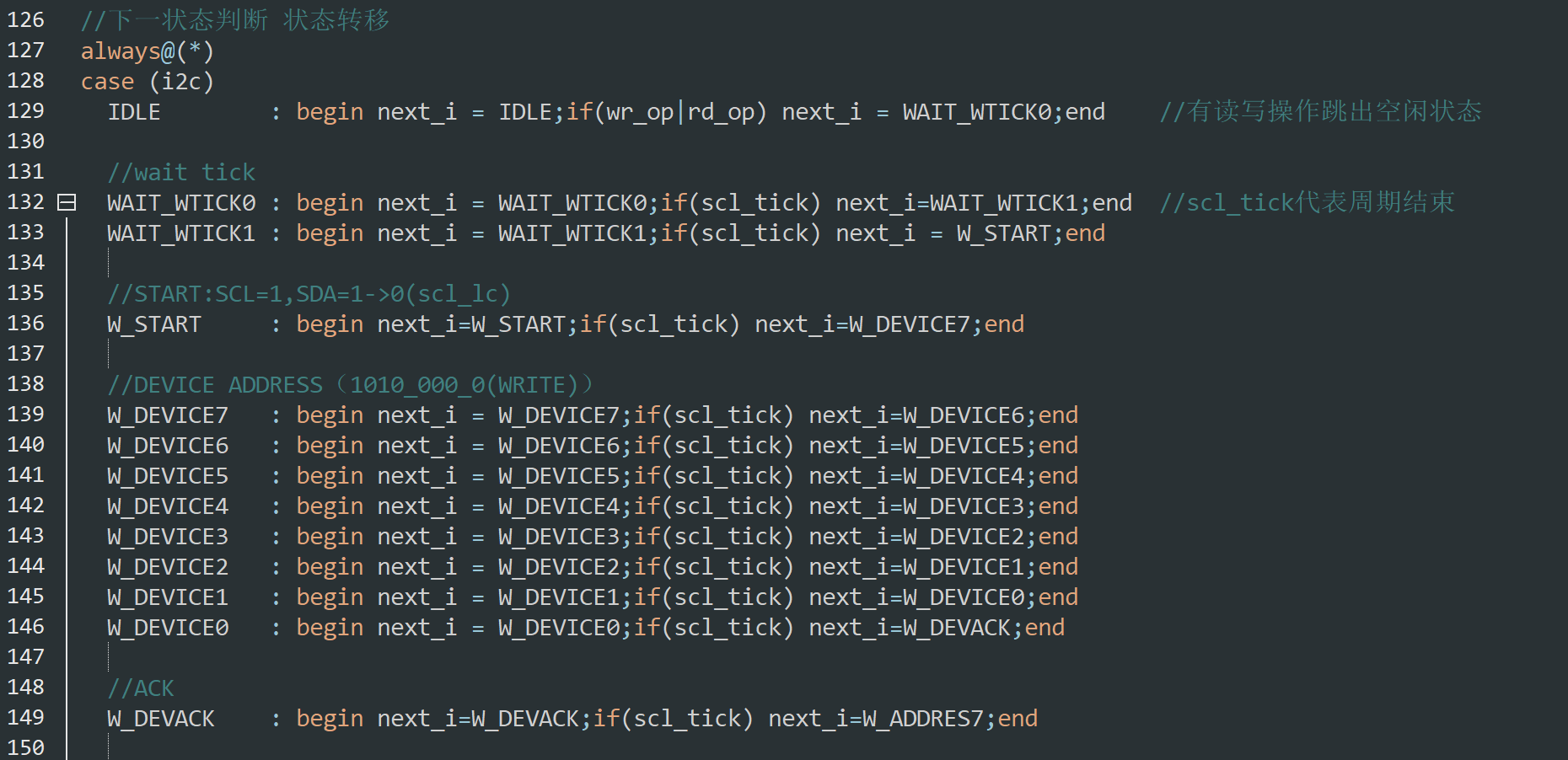
按状态图判断下一状态，scl\_tick为1时，scl周期结束；

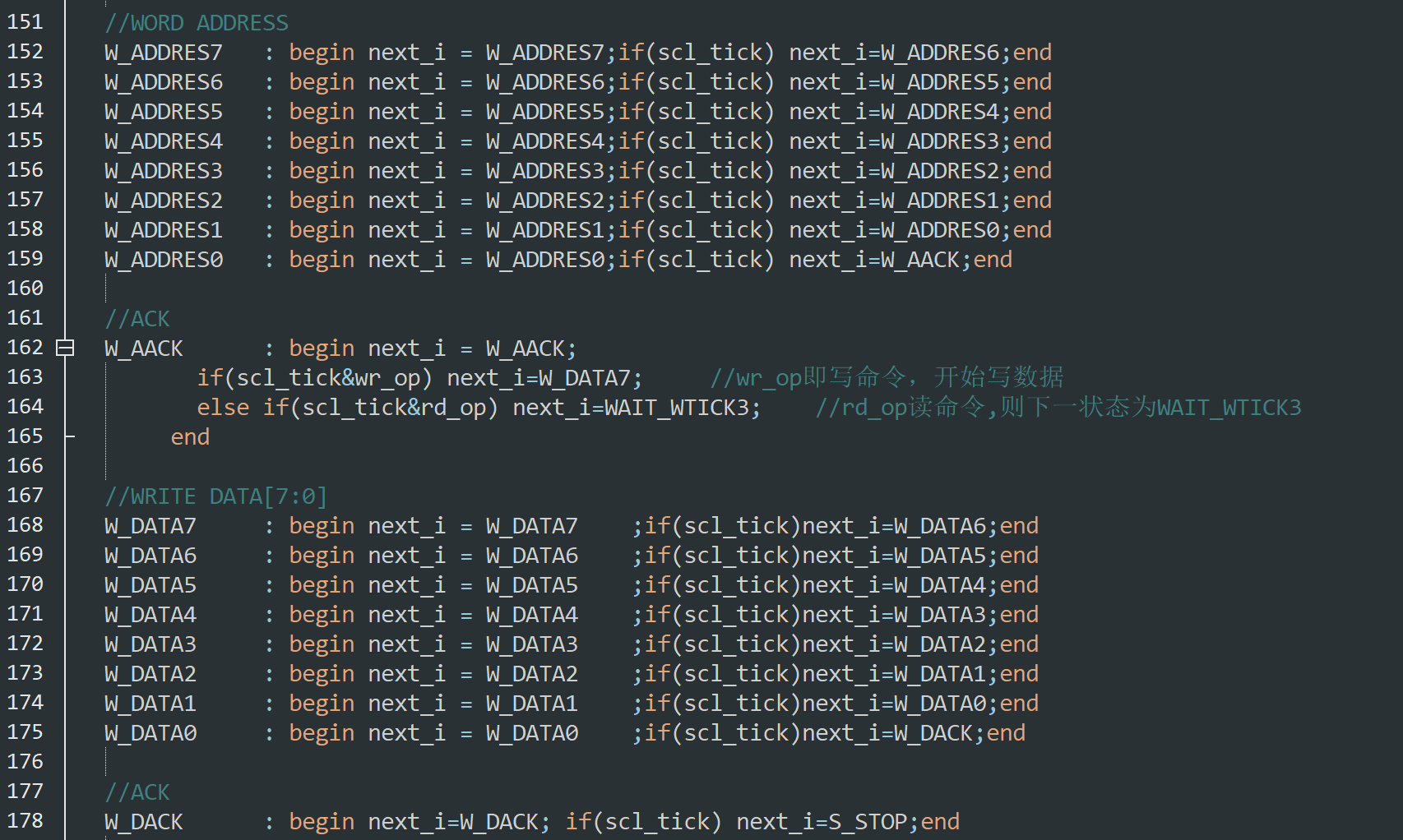
在 scl=1 时，sda 由 1->0，开始数据传输，并先写入器件地址(10100000)和数据地址，然后根据 wr\_op，rd\_op 判断进行读还是写操作，写操作直接开始写入数据，读操作则需要重新写器件地址和数据地址，然后读取数据。

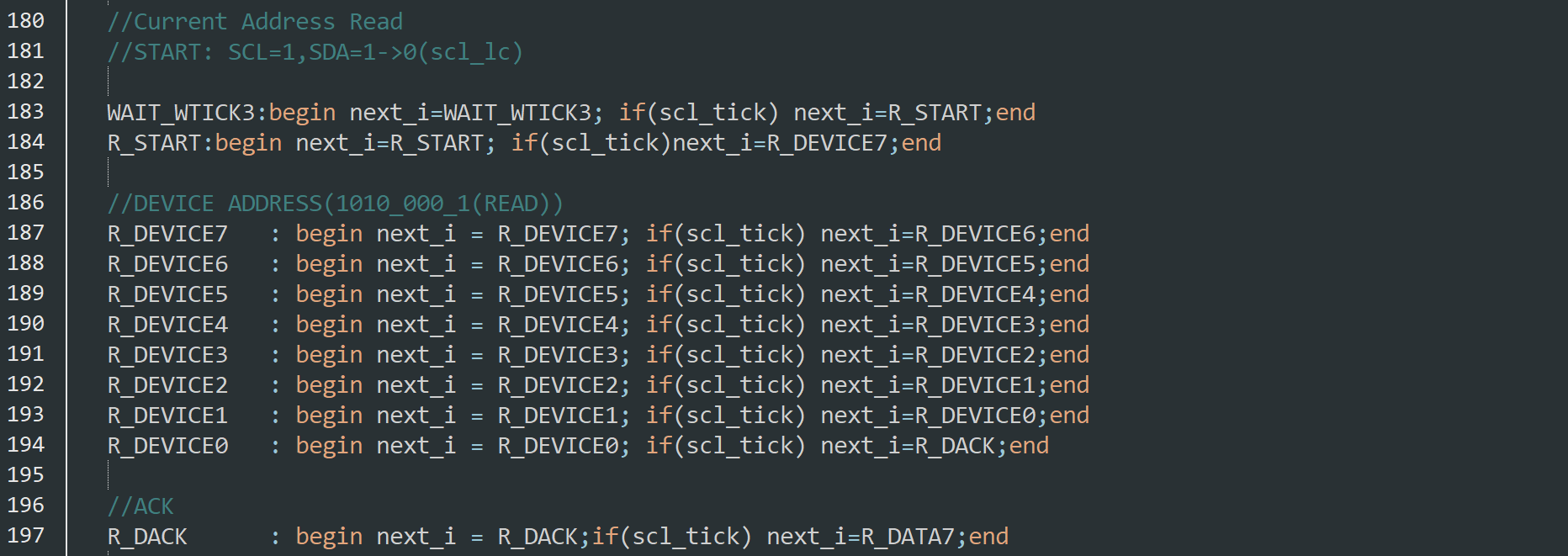


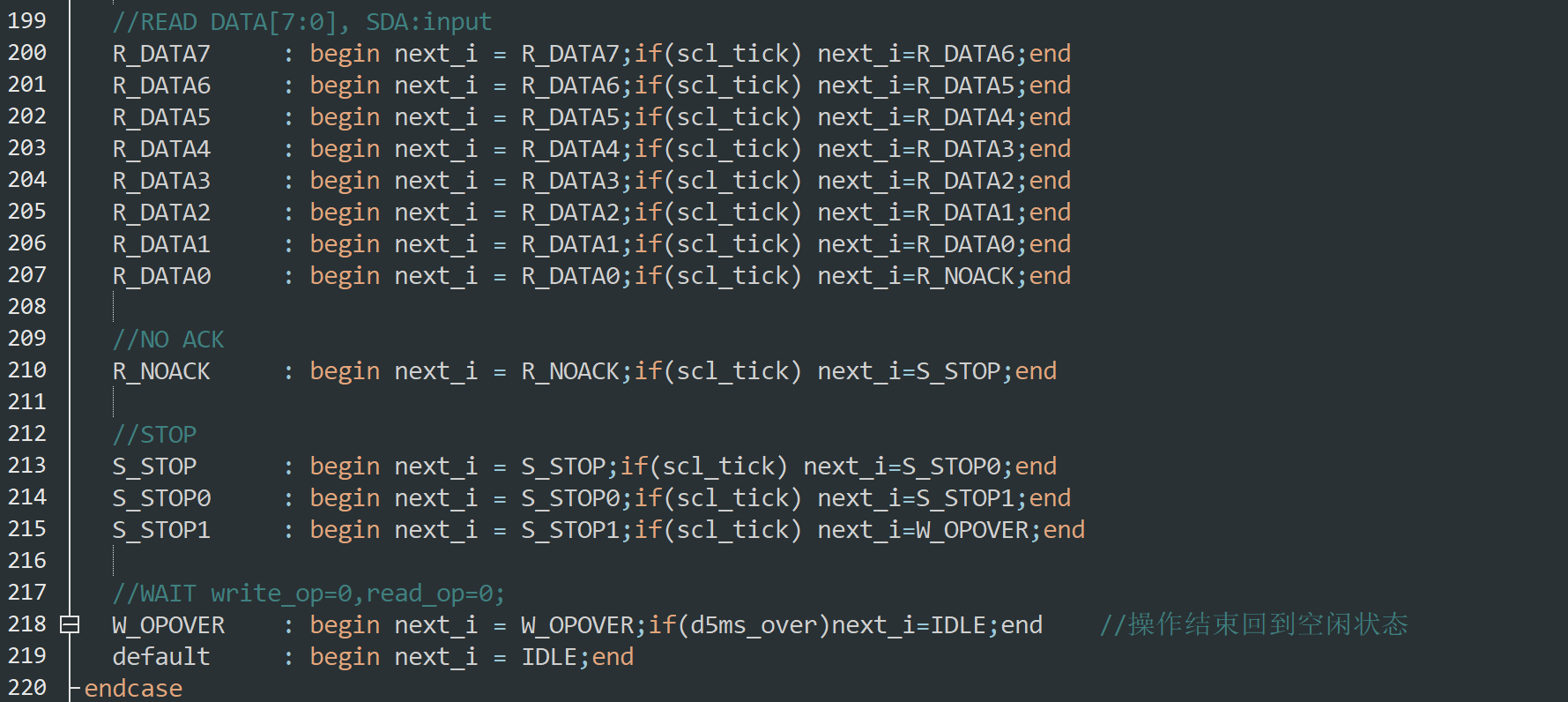


下一状态判断：



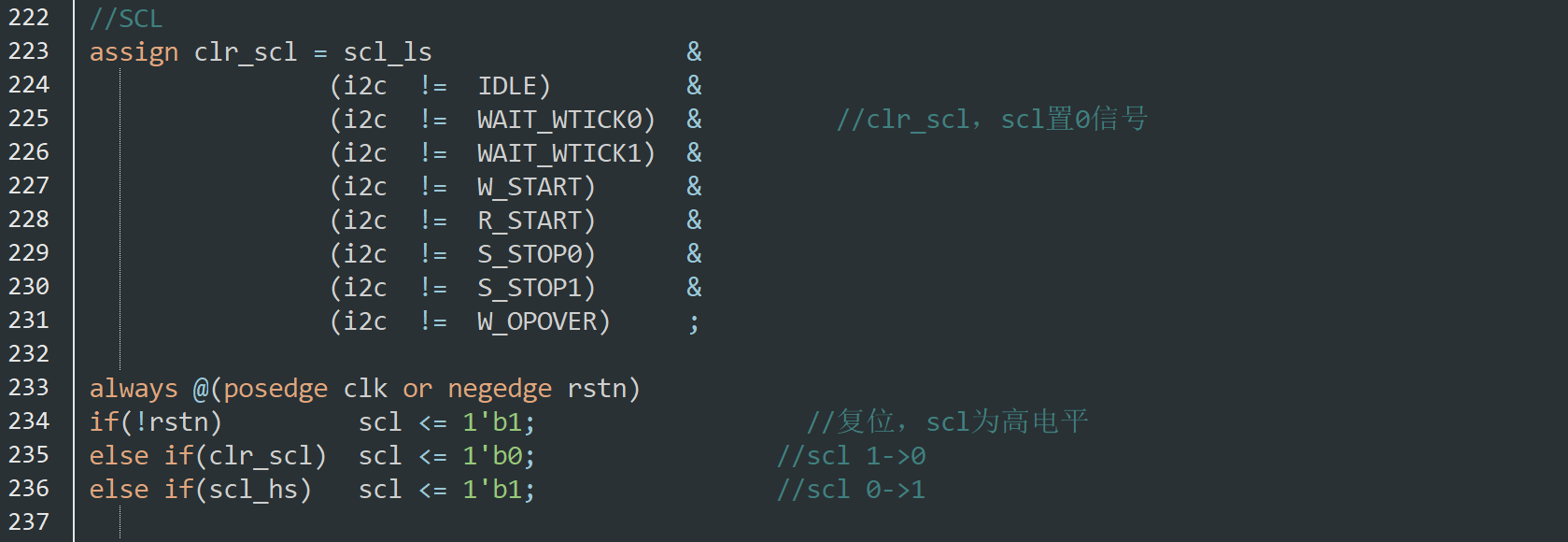




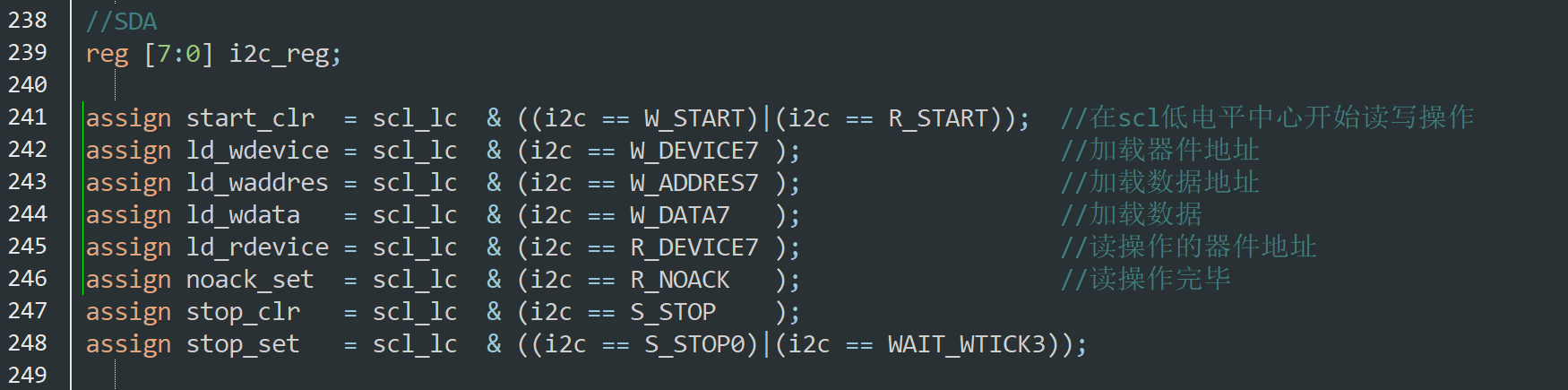


SCL同步的实现：

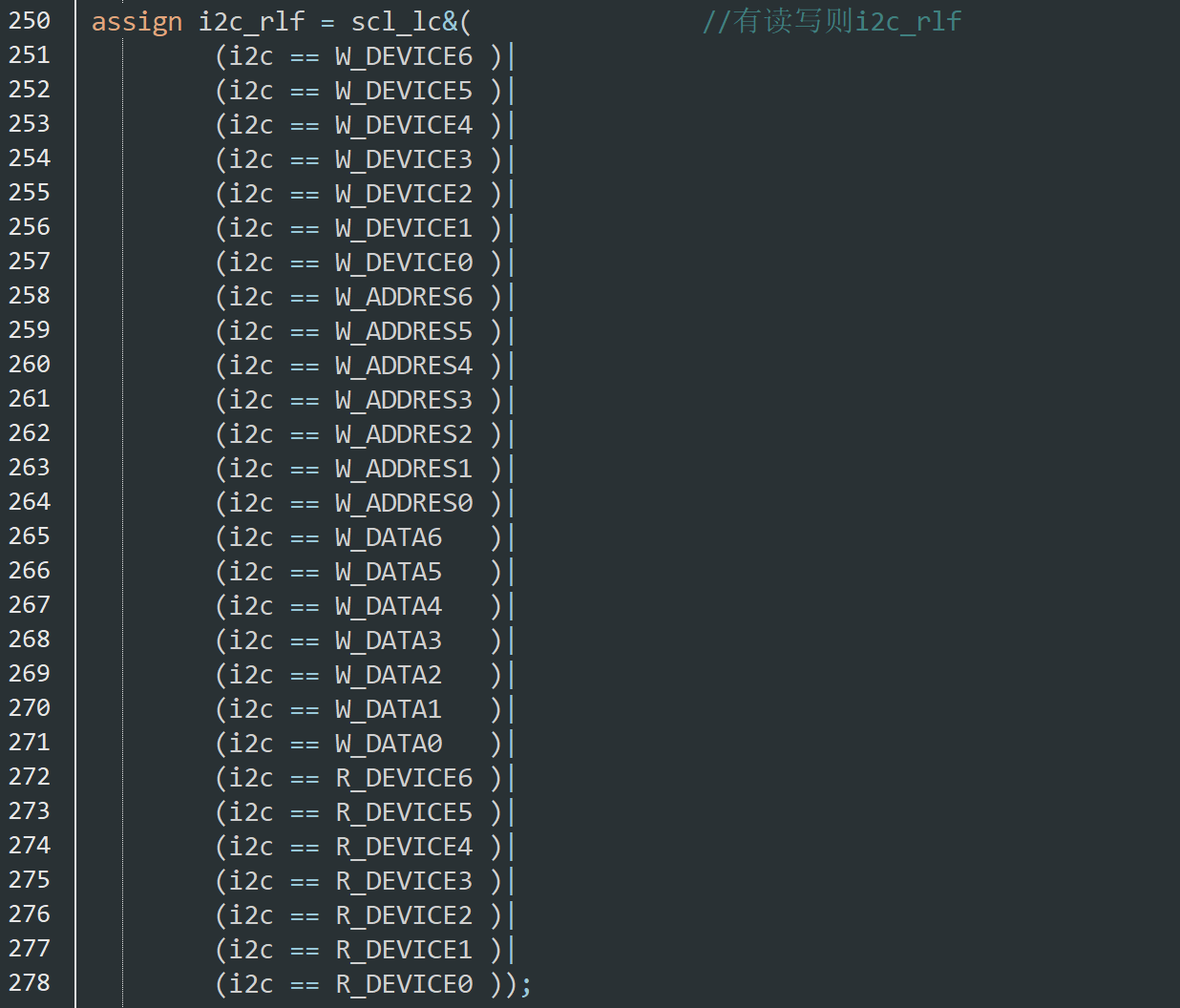
空闲（IDLE）、等待（WAIT）、操作结束（W\_OVER）、开始（START）等状态下SCL都是高电平。在scl\_ls（scl的低电平开始）处把clr\_scl置1，把scl清0，在15个clk周期的scl\_hs处,再把scl拉高，就实现了SCL周期。



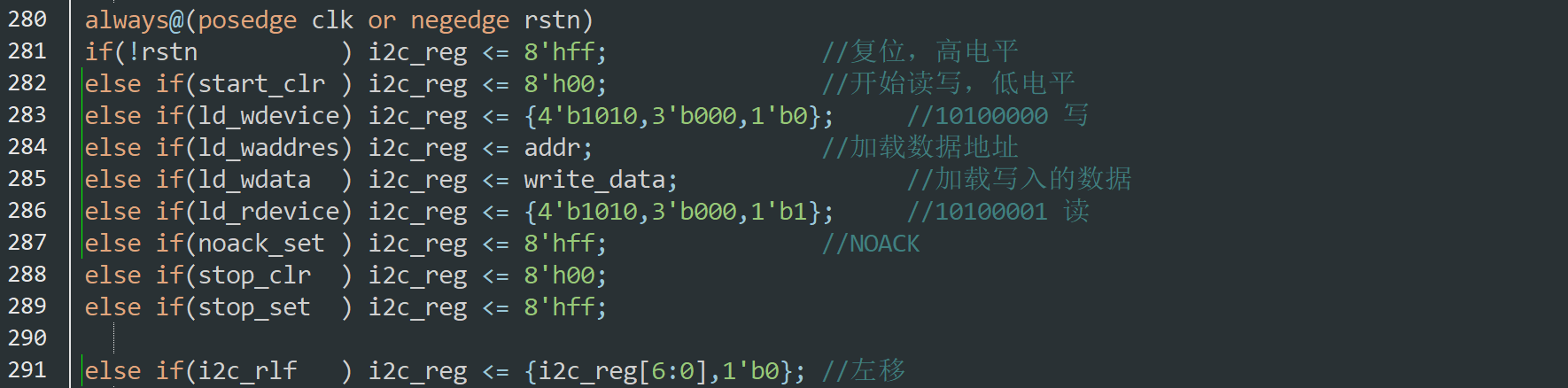
SDA的控制信号当处于对应的状态、并且scl位于低电平的中心位置时置1，根据控制信号在SDA上进行读写。用i2c\_reg寄存器存储中间数据。



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。

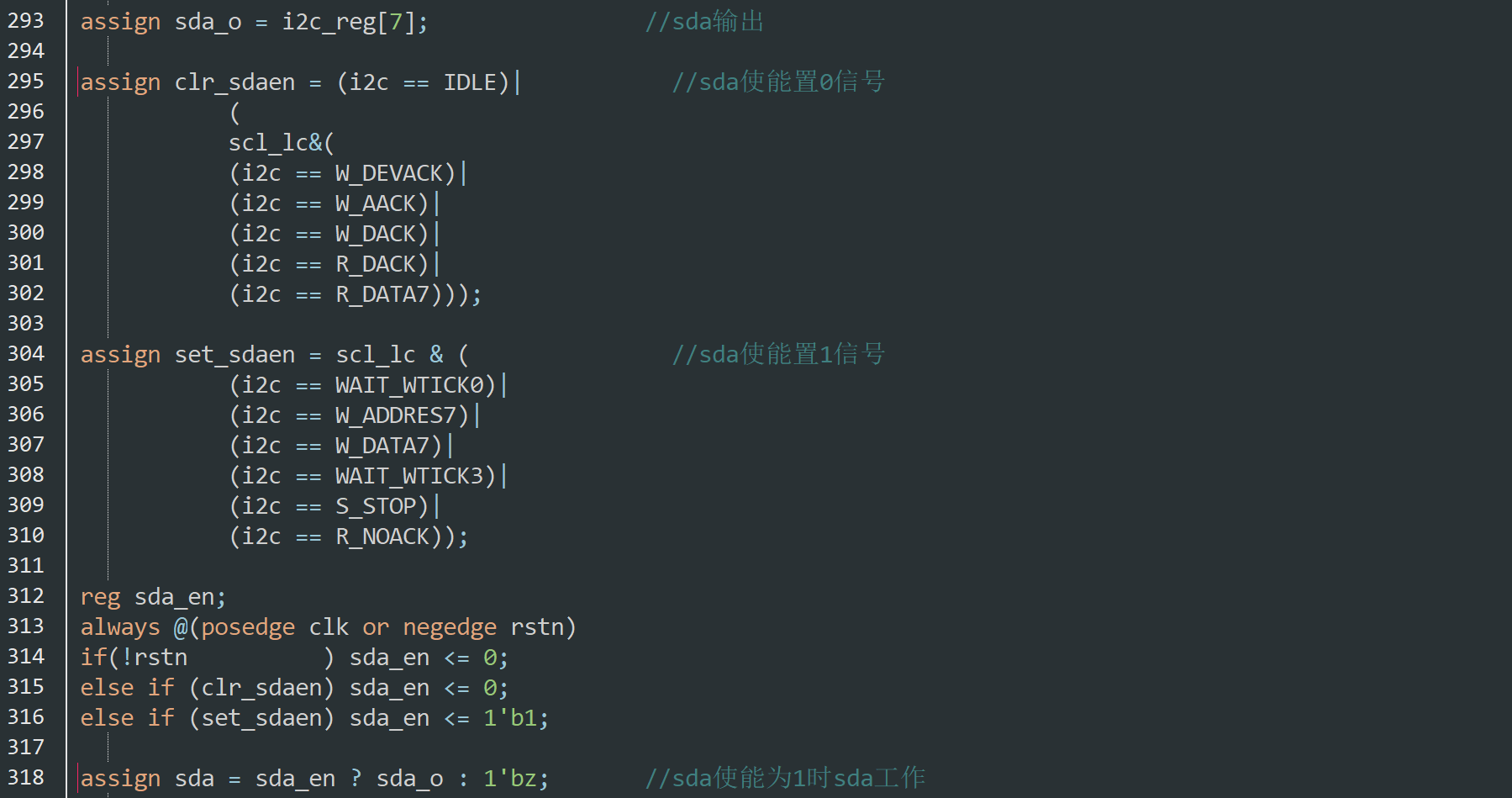


根据上述控制信号，将输入的特定数据保存到 i2c\_reg：

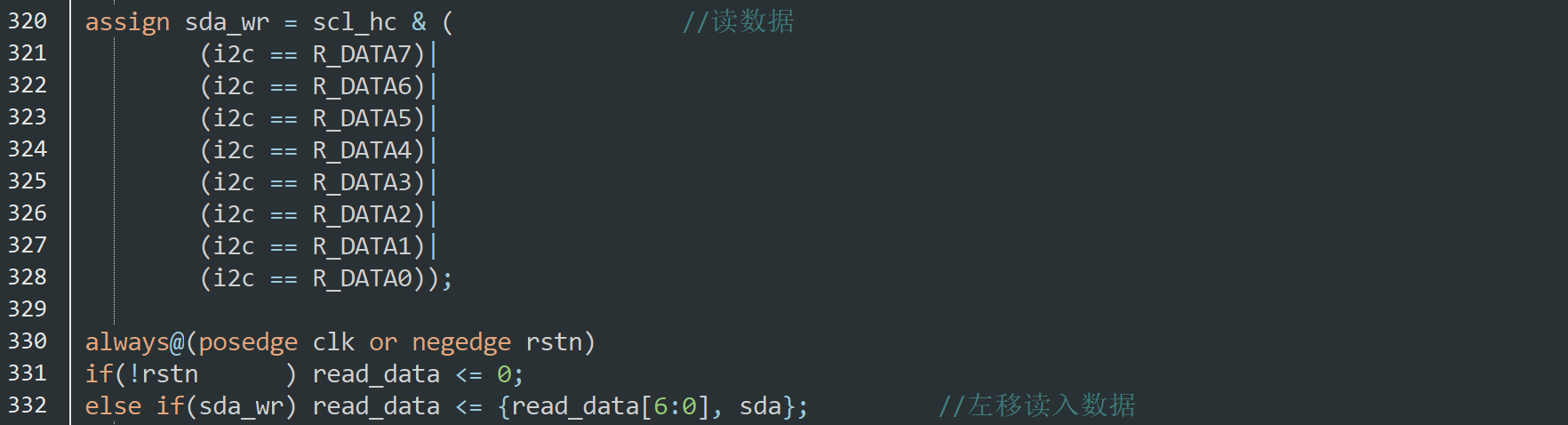


sda输出使用sda使能信号sda\_en控制，写器件地址，数据地址，写数据时使能信号为1，接收ACK响应时使能为0。

sda输出i2c\_reg的最高位，通过i2c\_reg的移位，一位一位完成读或写。



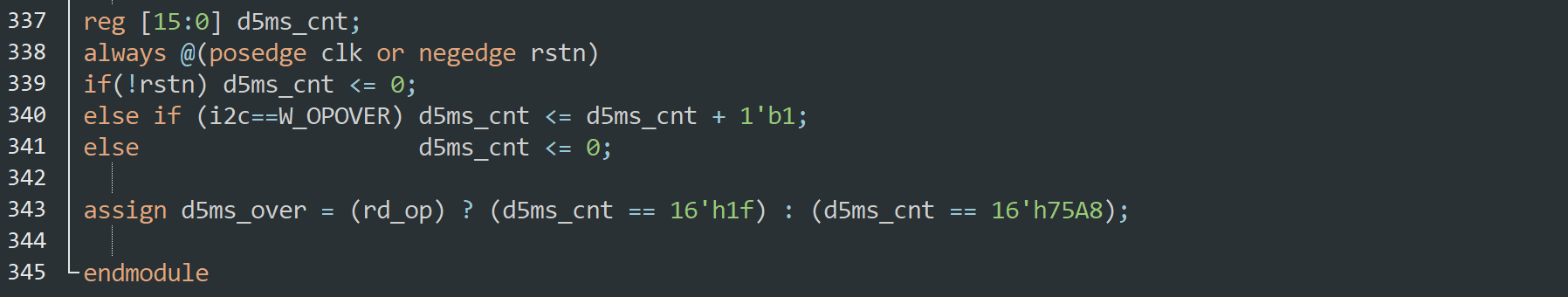
通过sda\_wr判断是否读数据，将数据读到read\_data



时钟周期取**6MHz，设置16位计数器d5ms\_cnt：**

**对于EEPROM开发板，写操作时间最大为5ms，于是设置写操作的阈值：**

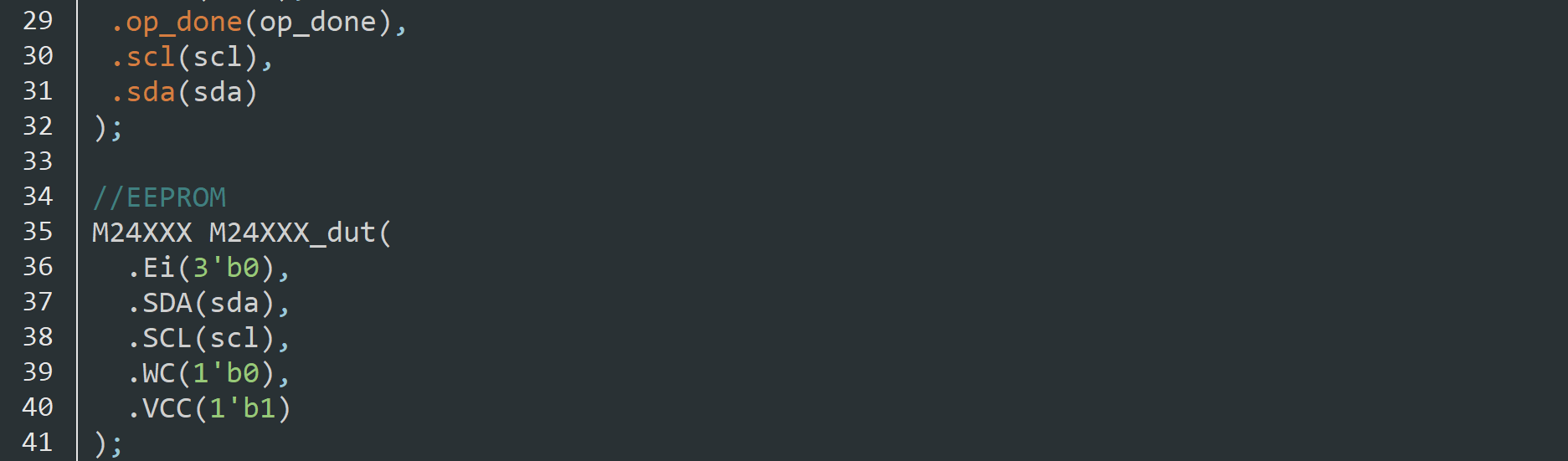
**d5ms\_cnt == 16'h75A8，有0x75A8 \* 166.66ns == 5.02ms，故设置每当d5ms\_cnt == 16'h75A8时d5ms\_over为1。同理读操作设置为31个时钟周期。**



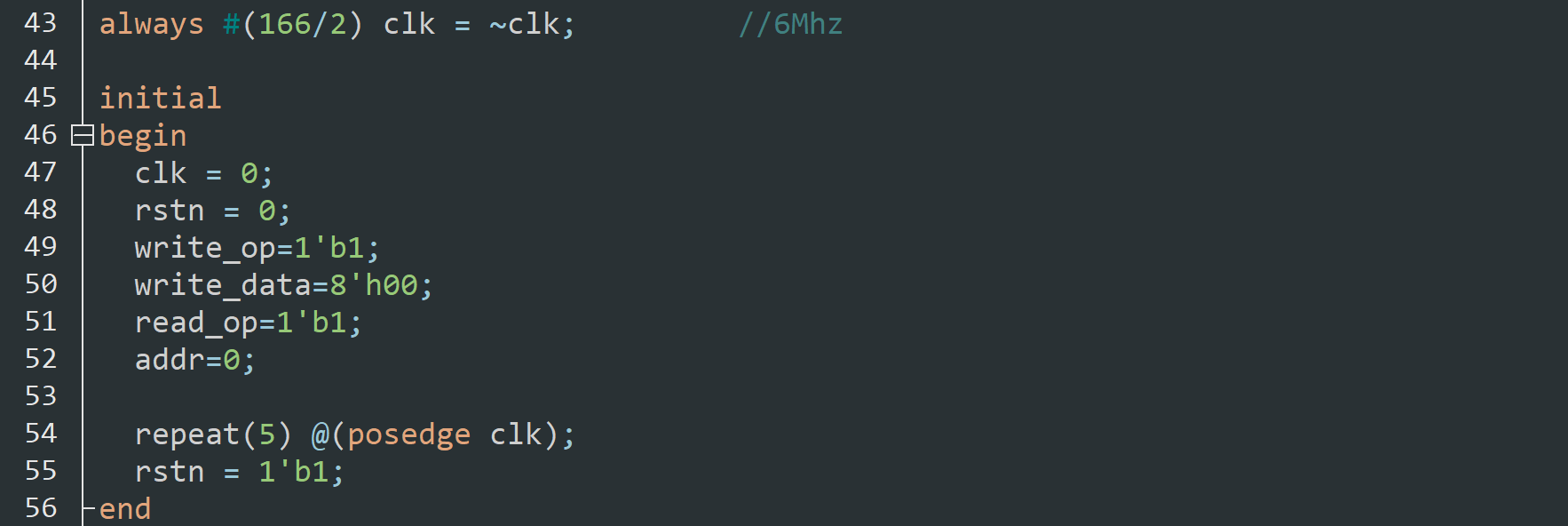
### TestBench代码说明：

模块声明与实例化：

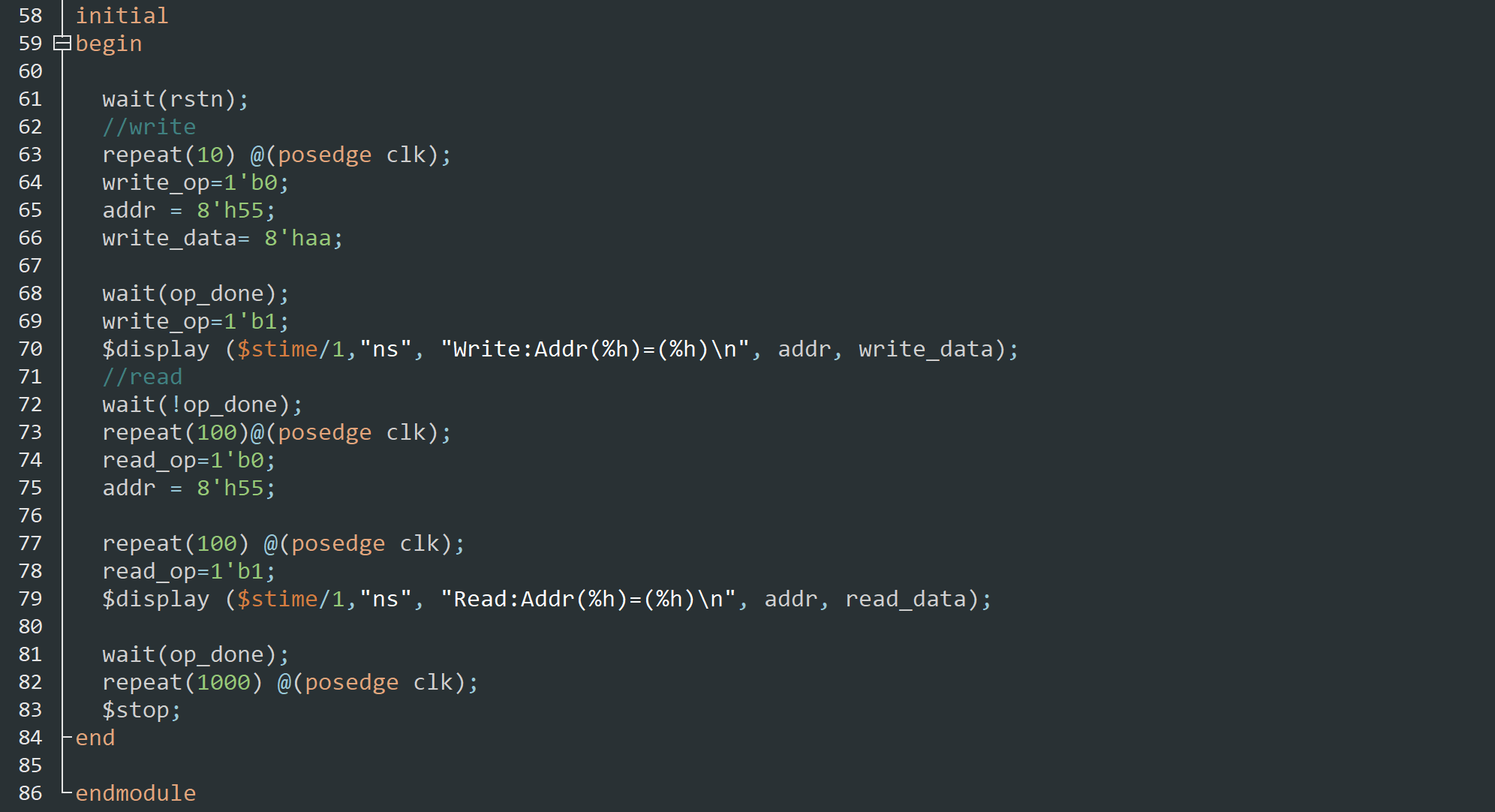




根据时钟频率 6Mhz 设置周期 166ns，并对信号初始化：

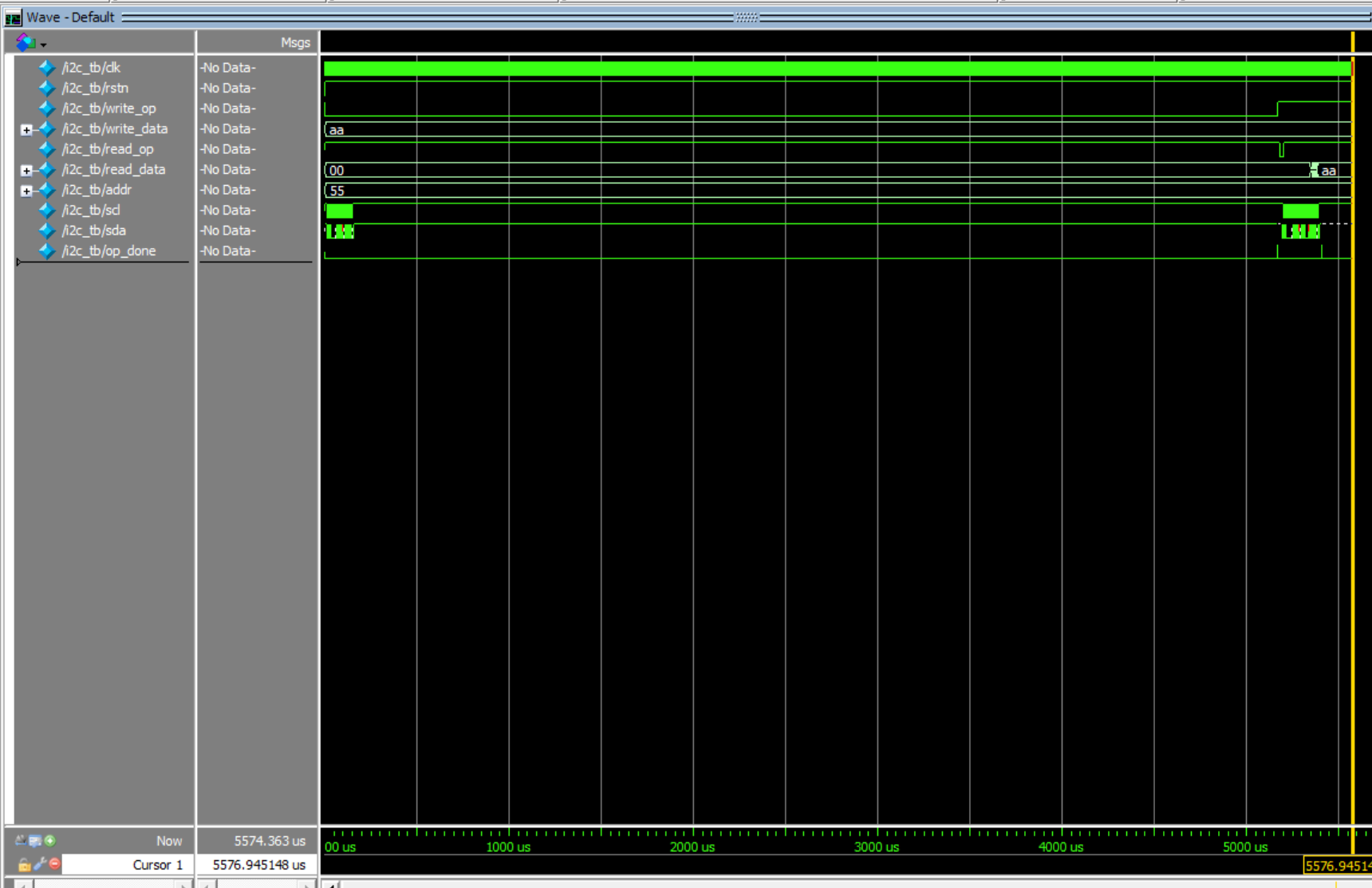


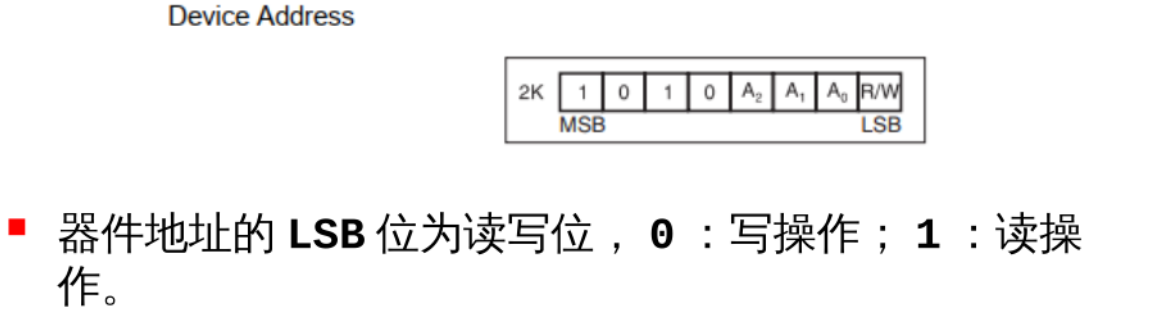
输入写命令信号，地址为8’h55，写入的数据为8’haa。等待操作完成后，将write\_op设为1（高电平无效），输入读命令信号，读出地址8’h55中的数据，读出的数据应该为刚刚写入的8’haa。

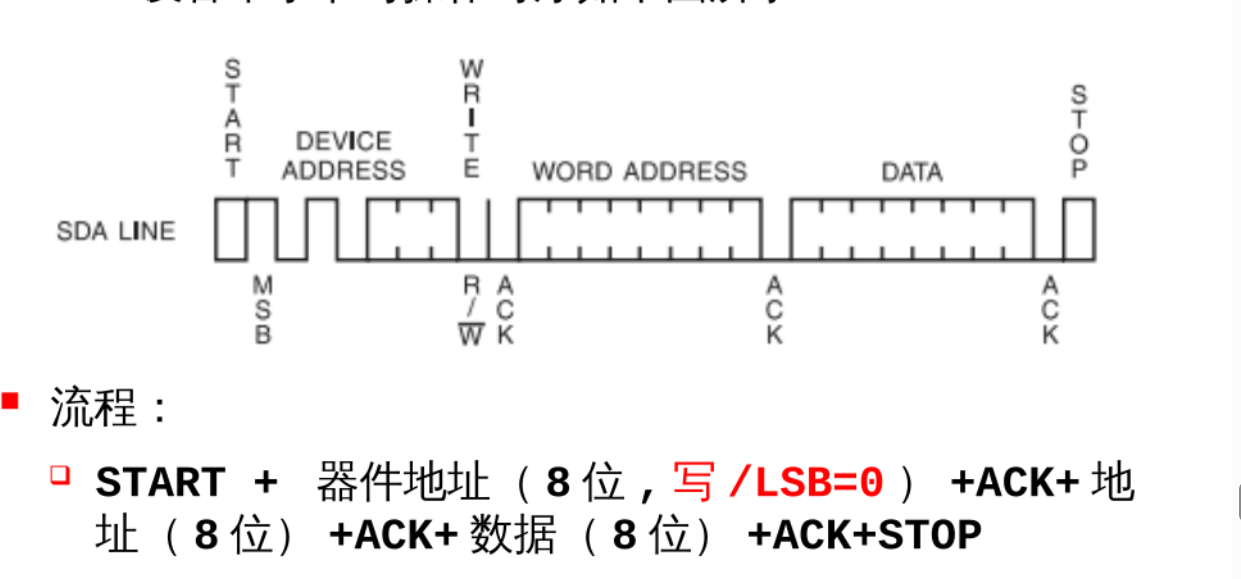


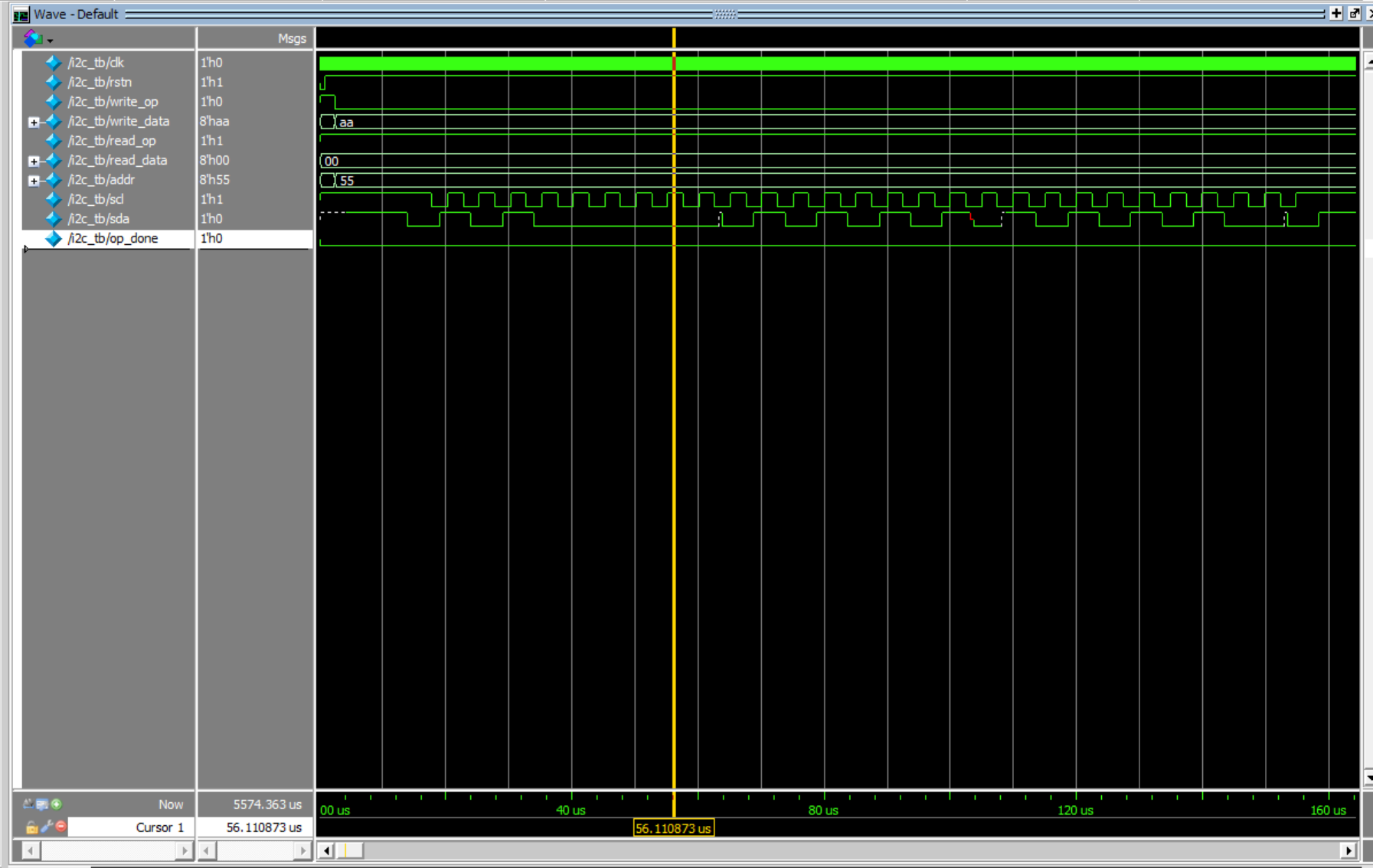
### 仿真结果：

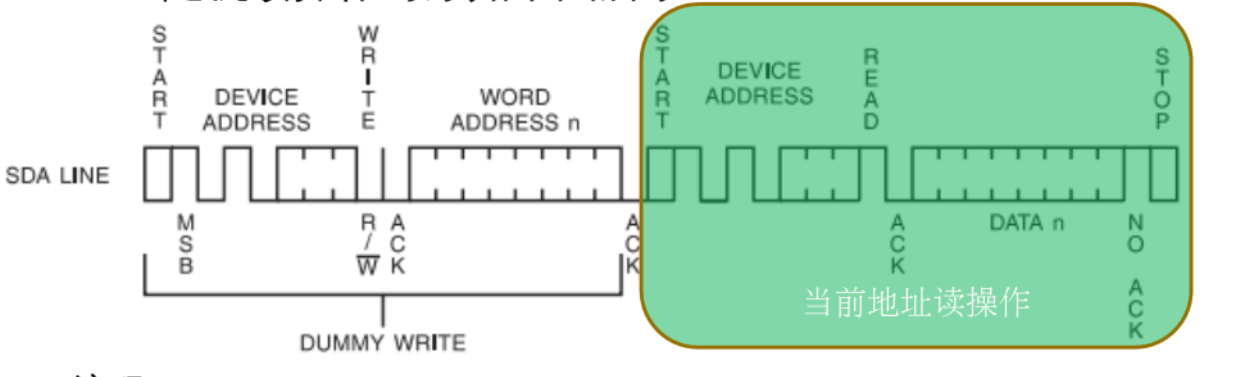
### 

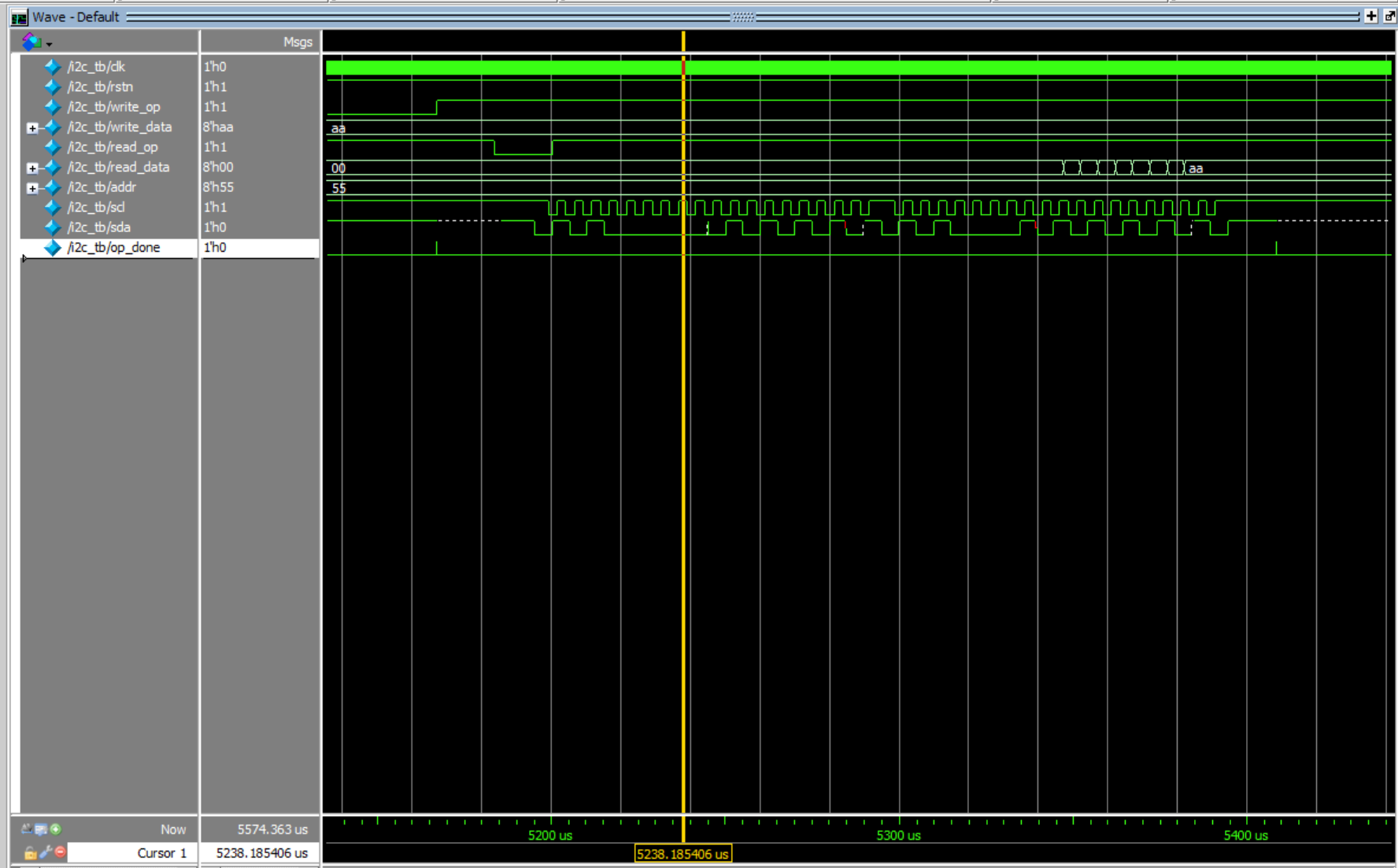












# 三．实验总结

1. 通过本次实验，熟悉了verilog语法，可以分辨并使用verilog语言描述时序逻辑和组合逻辑。
2. 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。
3. 了解了test\_bench的编写以及熟悉了使用ModelSim进行波形仿真。
4. 理解了I2C接口协议以及I2C协议下SCL，SDA数据是如何传输的。
5. 能够基本理解给出的参考代码，可以根据对应波形仿真结果解释I2C协议的数据传输。
6. 对于具体的实现流程还不太熟练，自己直接写会有些吃力，还需要进一步理解。