

# 数字钟设计

## 1. 实验要求

设计一个数字钟，具体设计要求如下：

1. 以时、分、秒形式显示；
2. 二十四小时循环
3. 具有复位功能
4. 有时间校准和调时功能
5. 具有整点报时功能（由点灯闪烁提醒）

## 2. **课题背景**

20世纪末，电子技术获得了飞速的发展。在其推动下，现代电子产品几乎渗透了社会的各个领域，有力地推动了社会生产力的发展和社会信息化程度的提高，同时也使现代电子产品性能进一步提高、产品更新换代的节奏也越来越快。数字钟已成为人们日常生活中必不可少的生活日用品。广泛用于个人家庭以及车站、码头、剧场、办公室等公共场所，给人们的生活、学习、工作、娱乐带来极大的方便。由于数字集成电路技术的发展和采用了先进的石英技术，使数字钟具有走时准确、性能稳定、集成电路有体积小、功耗小、功能多、携带方便等优点。  
因此本次设计就用数字集成电路和逻辑门电路来设计一个数字式电子钟，使其完成时间显示校准、调时功能、以及整点报时的功能。

## 3. **设计简介**

在确定选题为“数字钟设计”、以及明确了该选题的各项实验要求后，小组展开了细致的讨论，并查阅了大量的文献资料，最终确定了如下的设计思路：

1. 计时模块：
2. 数码管显示接口：
3. led显示接口：
4. 按键消抖接口：

## 4. **总体方案与分工**

* 模块划分

对于 FPGA，输入为

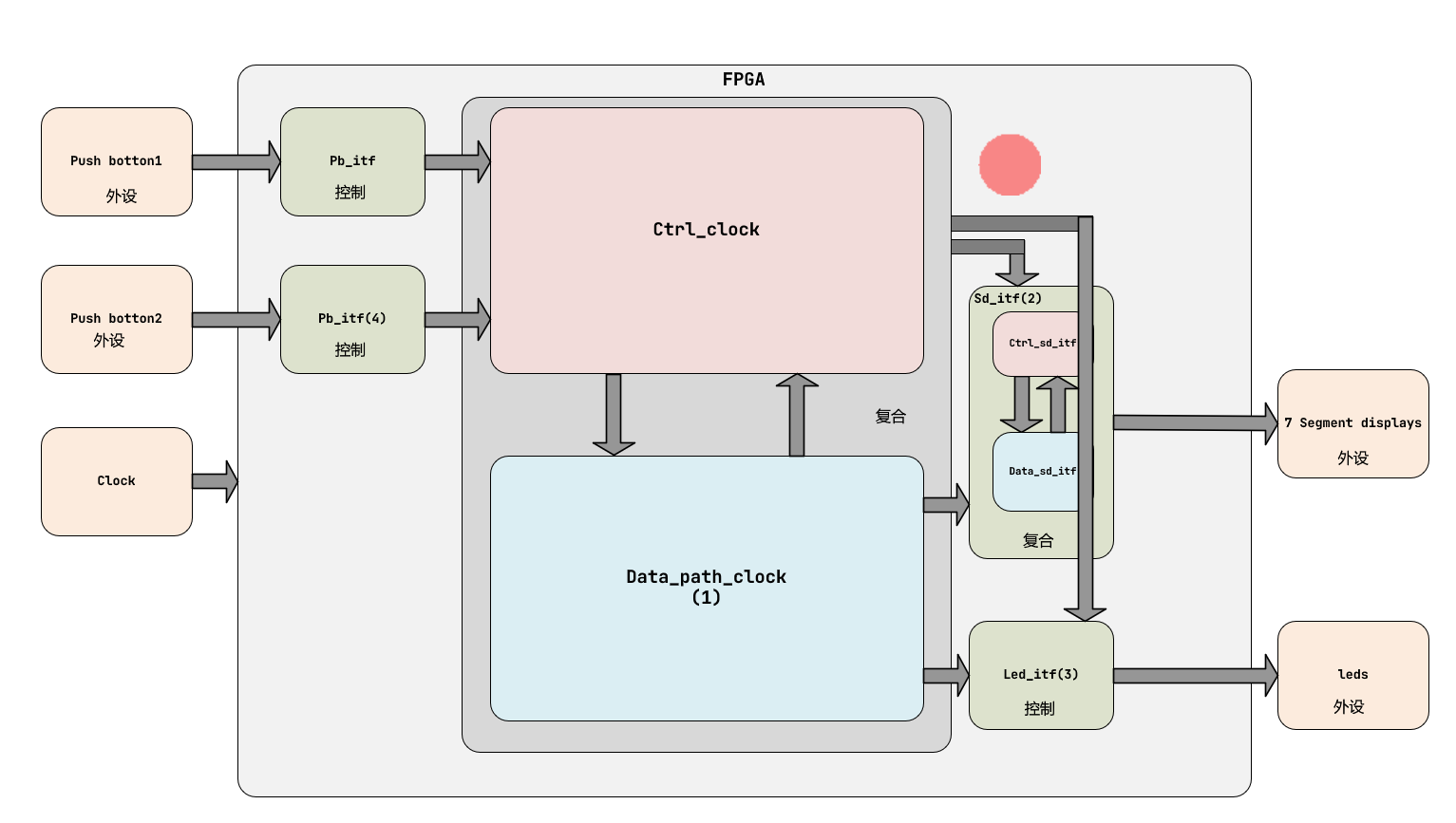
* 2 个按钮
* 系统时钟与复位

输出为

* 数码管
* led

对于输入和输出，都需要一个接口模块进行数据交互。

时钟主体负责与各个输入输出接口模块进行交互



* 分工

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 时钟主体 | 数码管接口 | led 接口 | 按键消抖 | 音乐播放 |
| **陈鸿冰** |  |  |  |  |  |
| **陈胜堃** |  |  |  |  |  |
| **戴宇韬** |  |  |  |  |  |

## 5. **单元模块**

以下分别设计各个模块

### 5.1 **时钟主体**

时钟主体的输入经过按键消抖后的控制信号，同时需要输出提供给数码管和 led，将输入输出信号按照控制信号和数据信号分类后，可以得到

* 数据通路模块
* 控制通路模块

如上图所示

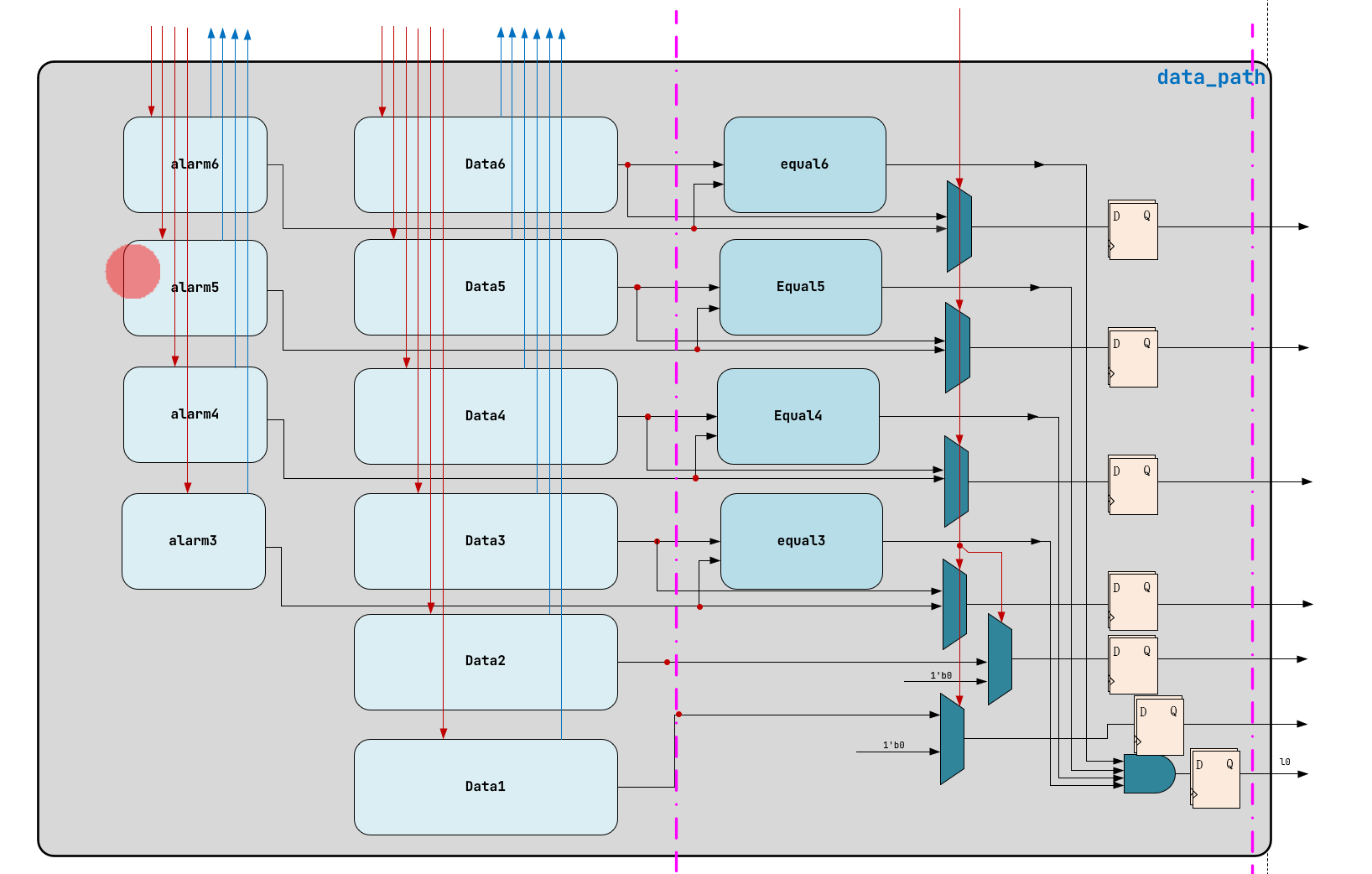
#### 5.1.1 **时钟主体数据通路**

数据通路负责将输入的信号经过处理，得的需要输出的数据。数据的处理可以按照流水线的方式进行分级，每一级的处理时间为 1 个时钟节拍，也就是输入数据到输出需要需要经过 1 个时钟周期

数据通路需要做的事

1. 存储当前的时间并控制加减
2. 存储闹钟的时间并控制加减
3. 将两者进行对比
4. 输出对比结果给 led模块
5. 输出当前时间或闹钟时间给数码管

要实现这些事，可以构建如下的数据通路



如图，第一级进行存储当前时间和闹钟时间，并进行控制。包括加一，减一，置数，清零。

第二级进行选择，并采用寄存器输出。同时对比闹钟时间与当前时间，把对比结果用寄存器输出

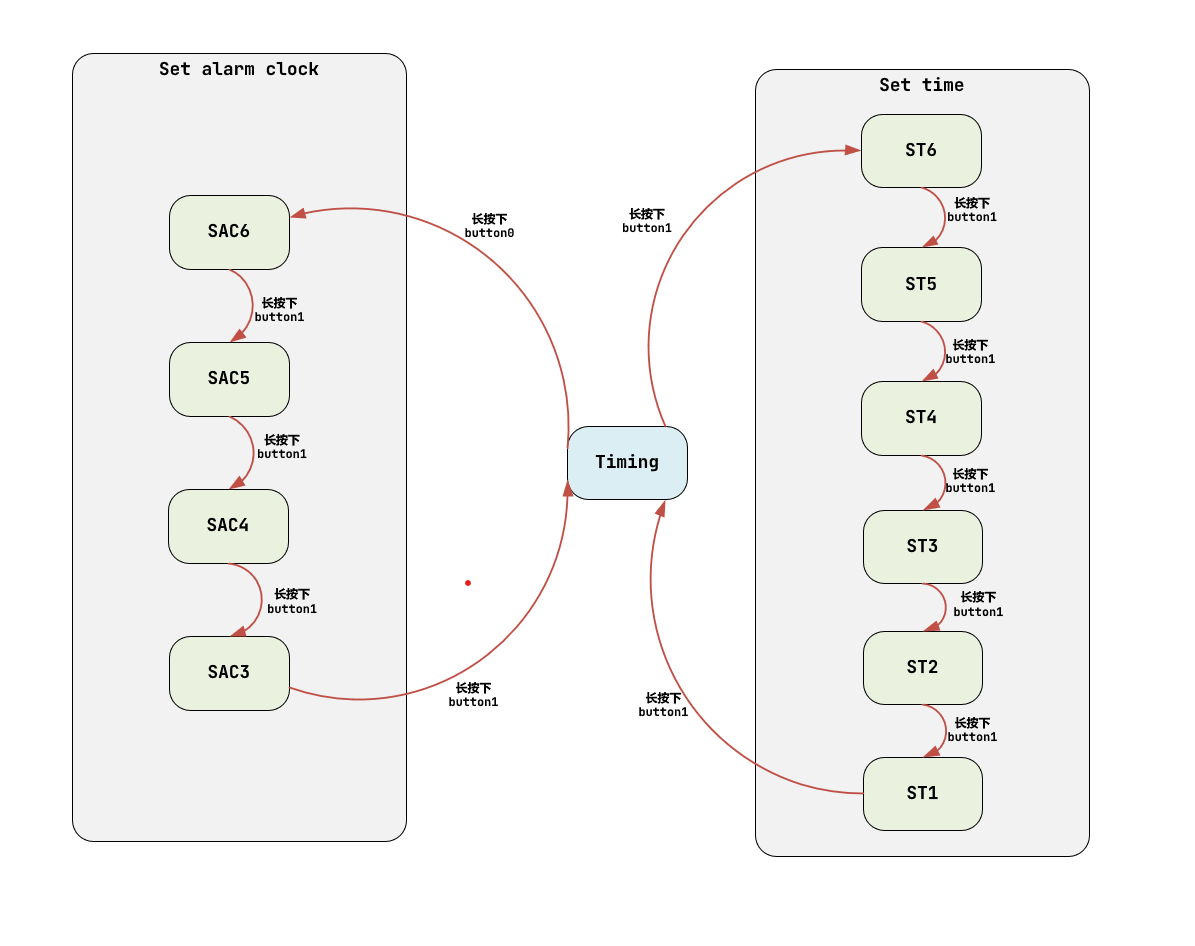
直接通过数据通路写出代码

alarm inst6\_alarm(  
 .alarm\_keep (alarm\_keep[3]),  
 .alarm\_add (alarm\_add[3]),  
 .alarm\_sub (alarm\_sub[3]),  
 .alarm\_reset (alarm\_reset[19:15]),  
 .alarm\_clear (alarm\_clear[3]),  
  
 .alarm\_numn (alarm\_num6),  
  
 .sysclk (sysclk),  
 .rst\_n (rst\_n)  
 );  
  
 equal inst6(  
 .numn (num6),  
 .alarm\_numn (alarm\_num6),  
  
 .eqn (eq6)  
 );  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 l0 <= 1'b0;  
 end  
 else if (eq6 & eq5 & eq4 & eq3 & eq2 & eq1) begin  
 l0 <= 1'b1;  
 end  
 else if (stop)begin  
 l0 <= 1'b0;  
 end  
 end  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 reg\_num6 <= 4'b0;  
 end  
 else if (tm\_sac) begin  
 reg\_num6 <= num6;  
 end  
 else begin  
 reg\_num6 <= alarm\_num6;  
 end  
 end

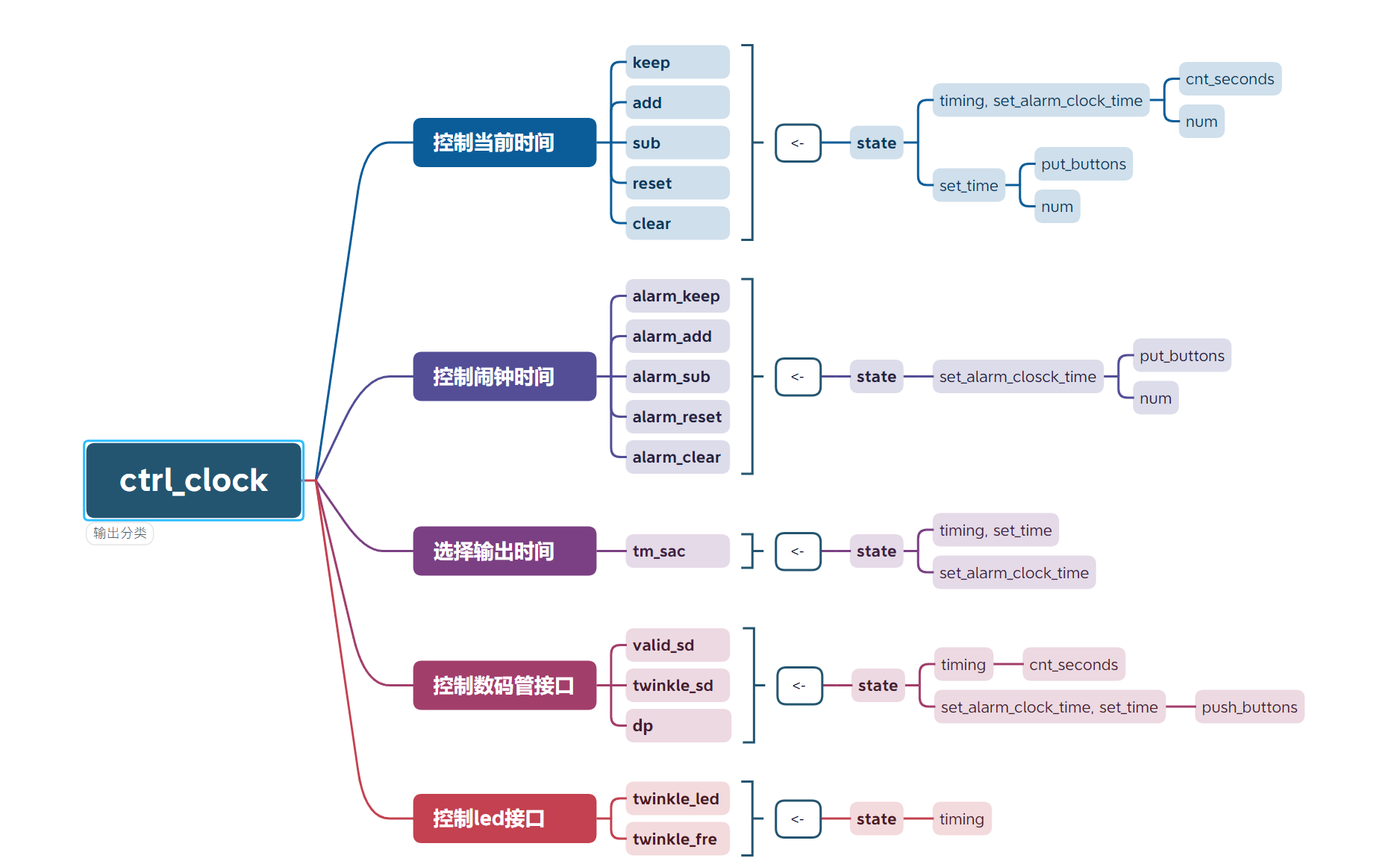
#### 5.1.2 **时钟主体控制模块**

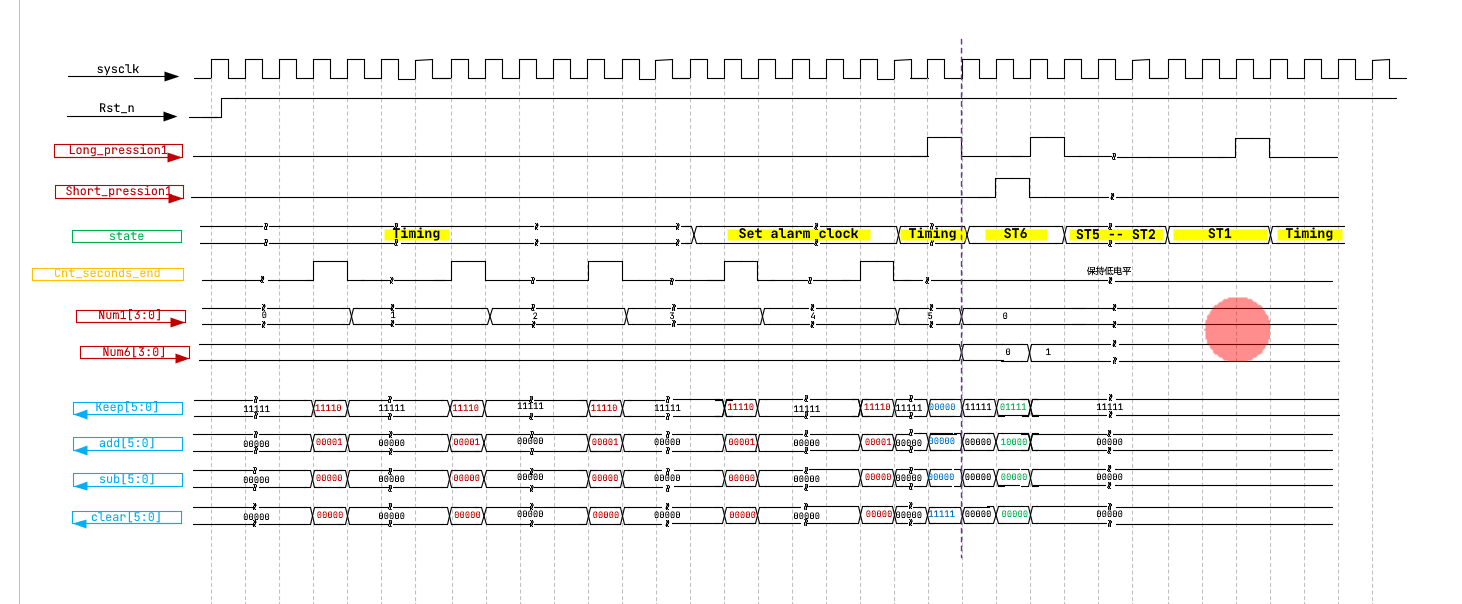
控制模块先考虑输出的控制信号主要实现怎么样的功能。再根据输出的行为确定控制模块的架构

* 计数器架构
* 状态机机构
* 复合结构



将信号分类，确定信号由架构的那一部分构成





根据波形，写出代码。注意先构造出架构，输出的控制信号主要由架构决定，辅助一些中间信号。

always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 state <= TIMING;  
 end  
 else begin  
 state <= next\_state;  
 end  
 end  
  
 always @ (\*) begin  
 case (state)   
 TIMING:  
 if (TIMING\_to\_ST6) begin  
 next\_state = ST6;  
 end  
 else if (TIMING\_to\_SAC6) begin  
 next\_state = SAC6;  
 end  
 else begin  
 next\_state = TIMING;  
 end  
 ST6:  
 if (ST6\_to\_ST5) begin  
 next\_state = ST5;  
 end  
 else begin  
 next\_state = ST6;  
 end  
 ST5:  
 if (ST5\_to\_ST4) begin  
 next\_state = ST4;  
 end  
 else begin  
 next\_state = ST5;  
 end  
 ST4:  
 if (ST4\_to\_ST3) begin  
 next\_state = ST3;  
 end  
 else begin  
 next\_state = ST4;  
 end  
 ST3:  
 if (ST3\_to\_ST2) begin  
 next\_state = ST2;  
 end  
 else begin  
 next\_state = ST3;  
 end  
 ST2:  
 if (ST2\_to\_ST1) begin  
 next\_state = ST1;  
 end  
 else begin  
 next\_state = ST2;  
 end  
 ST1:  
 if (ST1\_to\_TIMING) begin  
 next\_state = TIMING;  
 end  
 else begin  
 next\_state = ST1;  
 end  
 SAC6:  
 if (SAC6\_to\_SAC5) begin  
 next\_state = SAC5;  
 end  
 else begin  
 next\_state = SAC6;  
 end  
 SAC5:  
 if (SAC5\_to\_SAC4) begin  
 next\_state = SAC4;  
 end  
 else begin  
 next\_state = SAC5;  
 end  
 SAC4:  
 if (SAC4\_to\_SAC3) begin  
 next\_state = SAC3;  
 end  
 else begin  
 next\_state = SAC4;  
 end  
 SAC3:  
 if (SAC3\_to\_TIMING) begin  
 next\_state = TIMING;  
 end  
 else begin  
 next\_state = SAC3;  
 end  
 default:next\_state = TIMING;  
 endcase  
 end  
  
 assign TIMING\_to\_ST6 = long\_pression1 & (state == TIMING);  
 assign ST6\_to\_ST5 = long\_pression1 & (state == ST6);  
 assign ST5\_to\_ST4 = long\_pression1 & (state == ST5);  
 assign ST4\_to\_ST3 = long\_pression1 & (state == ST4);  
 assign ST3\_to\_ST2 = long\_pression1 & (state == ST3);  
 assign ST2\_to\_ST1 = long\_pression1 & (state == ST2);  
 assign ST1\_to\_TIMING = long\_pression1 & (state == ST1);  
  
 assign TIMING\_to\_SAC6 = long\_pression0 & (state == TIMING);  
 assign SAC6\_to\_SAC5 = long\_pression1 & (state == SAC6);  
 assign SAC5\_to\_SAC4 = long\_pression1 & (state == SAC5);  
 assign SAC4\_to\_SAC3 = long\_pression1 & (state == SAC4);  
 assign SAC3\_to\_TIMING = long\_pression1 & (state == SAC3);  
  
 reg [25:0] cnt\_seconds;  
 wire cnt\_seconds\_vld;  
 wire cnt\_seconds\_end;  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 cnt\_seconds <= 0;  
 end  
 else if (cnt\_seconds\_end) begin  
 cnt\_seconds <= 0;  
 end  
 else if (cnt\_seconds\_vld) begin  
 cnt\_seconds <= cnt\_seconds + 1'b1;  
 end  
 end  
  
 assign cnt\_seconds\_vld = 1'b1;  
 assign cnt\_seconds\_end = (cnt\_seconds == `SECONDS - 1); //测试的时候改成 50

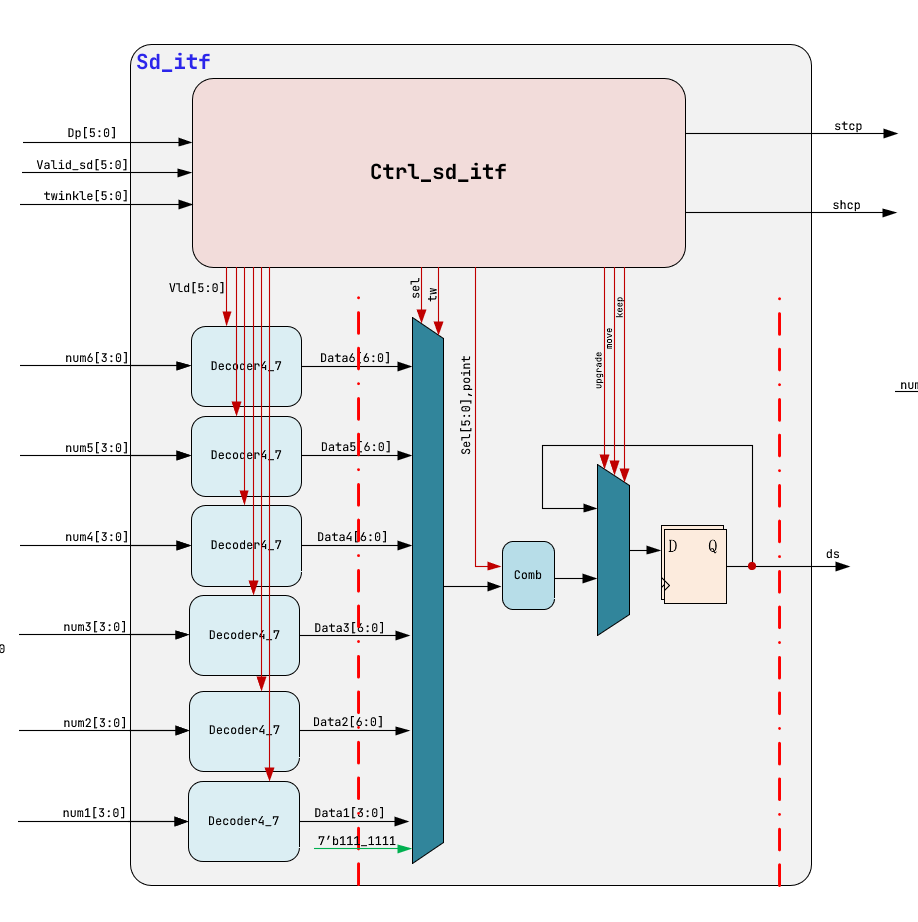
### 5.2 **数码管显示接口**

数码管显示接口要遵守外设接受数据的准则

#### 5.2.1 **数码管显示接口数据通路**

方法同上

1. 将数据的 4 位数据翻译为 8 段数码管识别的数据
2. 将 8 位数据与片选信号拼接，并移位，转换为串行输出

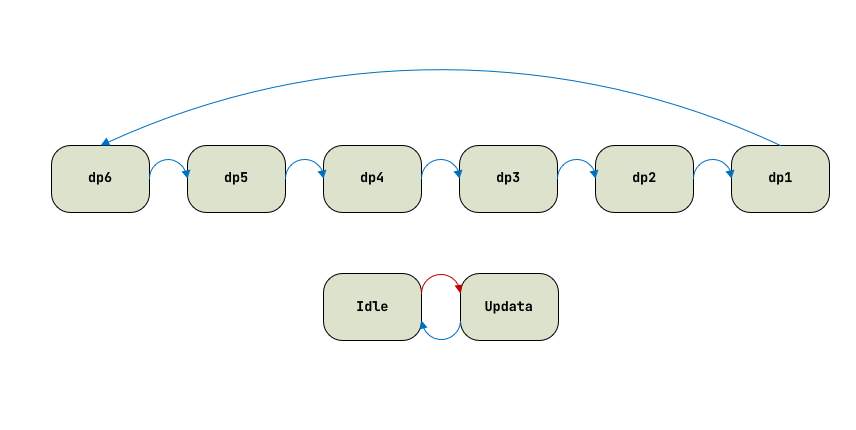


由数据通路直接写出代码

decoder4\_7 inst1\_decoder4\_7(  
 .numn (num1),  
  
 .vld (vld[0]),  
  
 .datan (data1),  
  
 .sysclk (sysclk),  
 .rst\_n (rst\_n)  
 );  
  
//------------------------------------------------------------  
  
 always @ (\*) begin  
 if (tw) begin  
 data = 7'b111\_1111;  
 end  
 else begin  
 case (sel)  
 6'b100\_000 : data = data6;  
 6'b010\_000 : data = data5;  
 6'b001\_000 : data = data4;  
 6'b000\_100 : data = data3;  
 6'b000\_010 : data = data2;  
 6'b000\_001 : data = data1;  
 default : data = 7'b111\_1111;  
 endcase  
 end  
 end  
  
 assign d\_p = ~dp;  
 assign data\_comb = {sel\_sd, d\_p, data};  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 data\_shift <= 14'b000000\_1\_111111;  
 end  
 else if (upgrade) begin  
 data\_shift <= data\_comb;  
 end  
 else if (move) begin  
 data\_shift <= {data\_shift[12:0],data\_shift[13]};  
 end  
 else if (keep) begin  
 data\_shift <= data\_shift;  
 end  
 end  
  
 assign ds = data\_shift[13];

#### 5.2.2 **数码管显示接口控制通路**

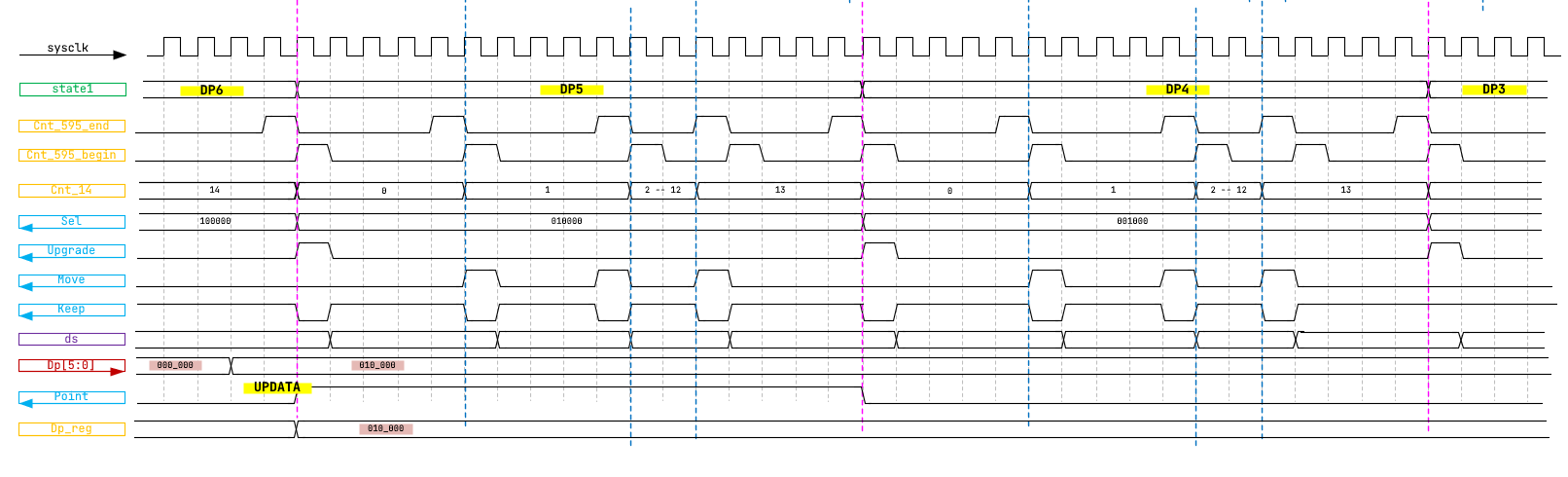
方法同上，确定架构



将控制信号进行分类，并确定由架构的那一部分构成



绘制主要波形



根据波形，先写搭建的架构，也就是计数器。输出的数据基本以计数为根据，辅助以一些中间信号

always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 state1 <= DP6;  
 end  
 else begin  
 state1 <= next\_state1;  
 end  
 end  
  
 always @ (\*) begin  
 case (state1)  
 DP6:  
 if (DP6\_to\_DP5) begin  
 next\_state1 = DP5;  
 end  
 else begin  
 next\_state1 = state1;  
 end  
 DP5:  
 if (DP5\_to\_DP4) begin  
 next\_state1 = DP4;  
 end  
 else begin  
 next\_state1 = state1;  
 end  
 DP4:  
 if (DP4\_to\_DP3) begin  
 next\_state1 = DP3;  
 end  
 else begin  
 next\_state1 = state1;  
 end  
 DP3:  
 if (DP3\_to\_DP2) begin  
 next\_state1 = DP2;  
 end  
 else begin  
 next\_state1 = state1;  
 end  
 DP2:  
 if (DP2\_to\_DP1) begin  
 next\_state1 = DP1;  
 end  
 else begin  
 next\_state1 = state1;  
 end  
 DP1:  
 if (DP1\_to\_DP6) begin  
 next\_state1 = DP6;  
 end  
 else begin  
 next\_state1 = state1;  
 end  
 default : next\_state1 = DP6;  
 endcase  
 end  
  
 assign DP6\_to\_DP5 = cnt\_595\_end & (cnt\_14\_end)& (state1 == DP6);  
 assign DP5\_to\_DP4 = cnt\_595\_end & (cnt\_14\_end)& (state1 == DP5);  
 assign DP4\_to\_DP3 = cnt\_595\_end & (cnt\_14\_end)& (state1 == DP4);  
 assign DP3\_to\_DP2 = cnt\_595\_end & (cnt\_14\_end)& (state1 == DP3);  
 assign DP2\_to\_DP1 = cnt\_595\_end & (cnt\_14\_end)& (state1 == DP2);  
 assign DP1\_to\_DP6 = cnt\_595\_end & (cnt\_14\_end)& (state1 == DP1);

### 5.3 **led 显示接口**

由单一控制模块构成

输入的控制信号有

* 闪烁频率 ：twinkle\_led
* 选择闪烁 led ：twinkle\_led

需要根据输入的闪烁频率，确定计数的终值。再根据终值的为分界线，将 led 点亮信号的占空比设置为 0.5

主要代码如下

always @ (\*) begin  
 case (twinkle\_led)  
 10'b00000\_00001: //0.2s  
 x = 10\_000\_000;  
 10'b00000\_00010: //0.5s  
 x = 25\_000\_000;  
 10'b00000\_00100: //1.0s  
 x = 50\_000\_000;  
 10'b00000\_01000: //1.5s  
 x = 75\_000\_000;  
 10'b00000\_10000: //2.0s  
 x = 100\_000\_000;  
 10'b00001\_00000: //2.5s  
 x = 125\_000\_000;  
 10'b00010\_00000: //3.0s  
 x = 150\_000\_000;  
 10'b00100\_00000: //4.0s  
 x = 200\_000\_000;  
 10'b01000\_00000: //5.0s  
 x = 250\_000\_000;  
 10'b10000\_00000: //6.0s  
 x = 300\_000\_000;  
 default: //1.0s  
 x = 50\_000\_000;  
 endcase  
 end  
  
 reg [28:0] cnt;  
 wire cnt\_vld;  
 wire cnt\_end;  
  
 //  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 cnt <= 0;  
 end  
 else if (cnt\_end) begin  
 cnt <= 0;  
 end  
 else if (cnt\_vld) begin  
 cnt <= cnt + 1'b1;  
 end  
 end  
  
 assign cnt\_vld = 1'b1;  
 assign cnt\_end = (cnt == x-1);  
  
 wire twinkle;  
  
 assign twinkle = (cnt >= 0 & cnt < x/2) ? 1'b1 : 1'b0;  
  
 genvar i;  
 generate  
 for (i=9; i>=0; i=i-1) begin : label  
 tw inst\_tw(  
 .lin (lin[i]),  
 .valid\_led (valid\_led[i]),  
 .twinkle\_led (twinkle\_led[i]),  
 .twinkle (twinkle),  
  
 .lout (lout[i]),  
  
 .sysclk (sysclk),  
 .rst\_n (rst\_n)  
 );  
 end  
 endgenerate  
  
 assign led9 = lout[9];  
 assign led8 = lout[8];  
 assign led7 = lout[7];  
 assign led6 = lout[6];  
 assign led5 = lout[5];  
 assign led4 = lout[4];  
 assign led3 = lout[3];  
 assign led2 = lout[2];  
 assign led1 = lout[1];  
 assign led0 = lout[0];

### 5.4 **按键消抖接口**

由两个控制模块构成，第一个模块负责将输入流的信号平滑化。第二个模块将平滑后信号的持续时间分为长按与短按

#### 5.4.1 **按键平滑模块**

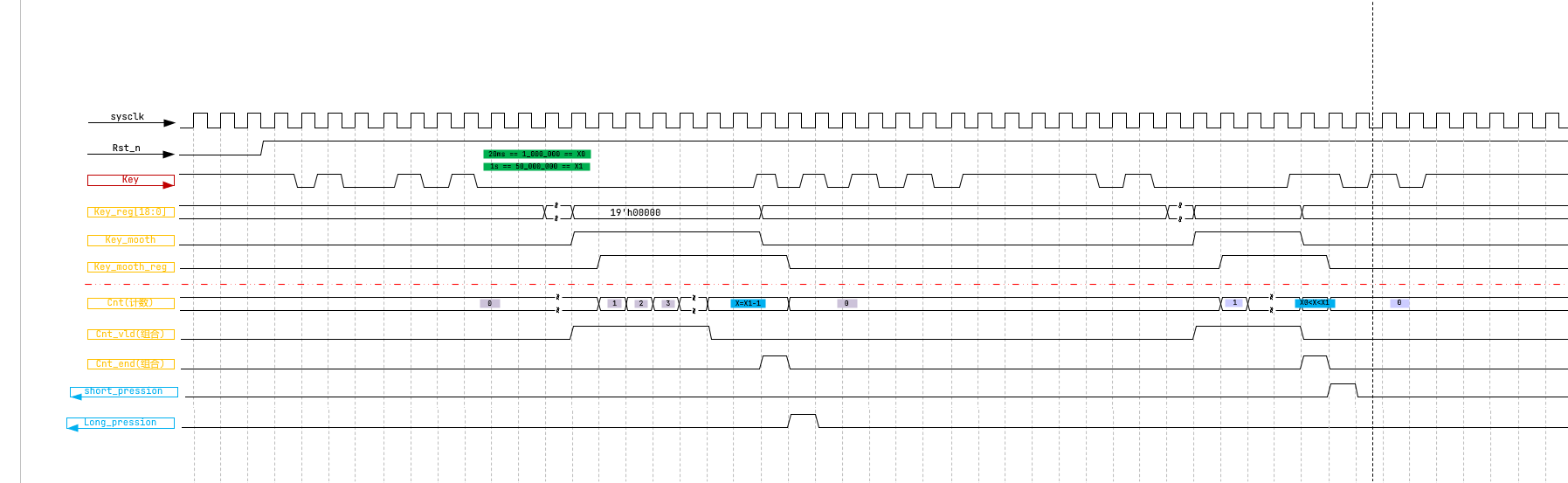
采用移位寄存器的方式判断输入信号时候稳定，设寄存器的输入个数为 19 个，只要连续输入 19 个 0，就认为输入的信号已经平稳

代码如下

always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 key\_reg <= 19'h7ffff;  
 end  
 else begin  
 key\_reg <= {key\_reg[17:0], key};  
 end  
 end  
  
 assign key\_mooth = &(~(key\_reg));

#### 5.4.2 **长短按判断模块**

以计数器作为架构，绘制出主要的波形图

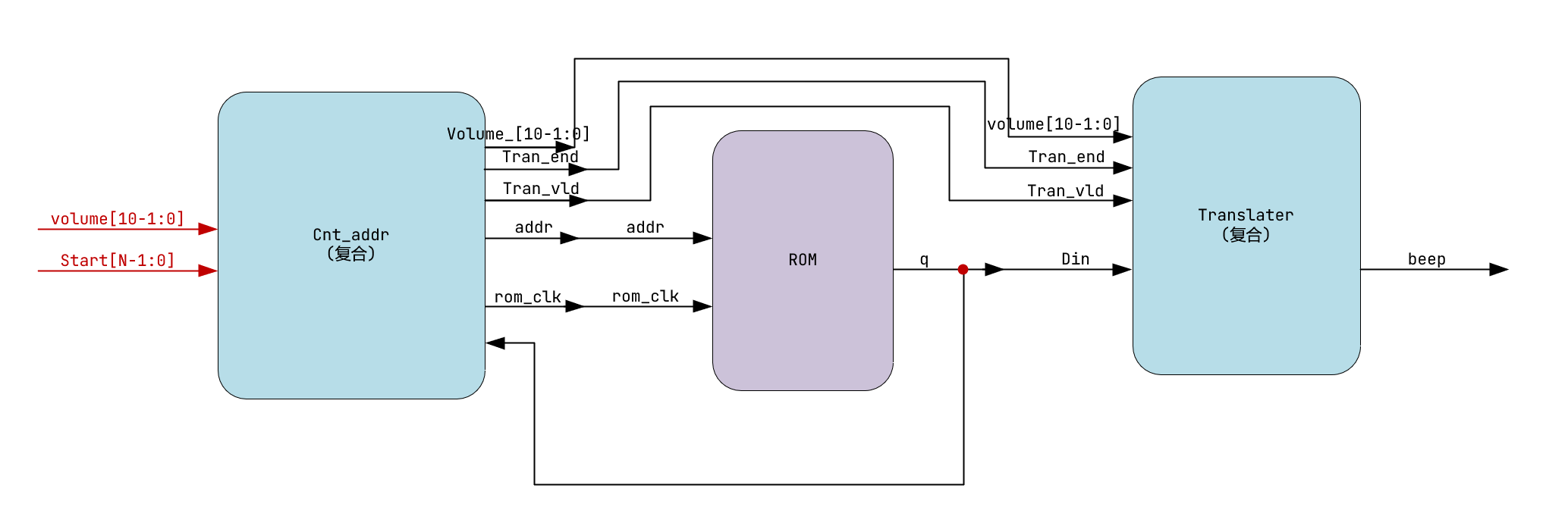


主要代码如下

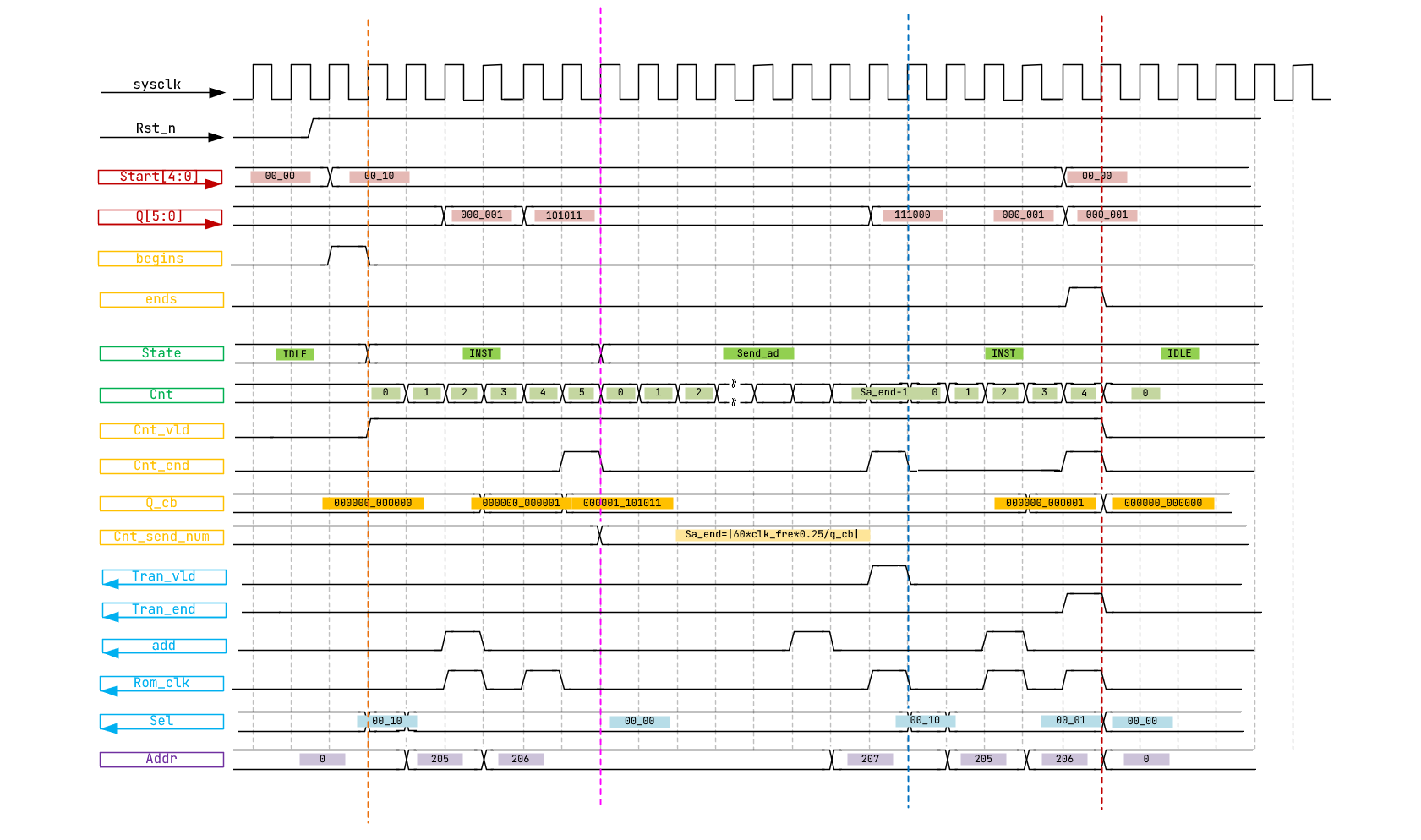
always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 key\_mooth\_r1 <= 1'b0;  
 end  
 else begin  
 key\_mooth\_r1 <= key\_mooth;  
 end  
 end  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 cnt <= 26'b0;  
 end  
 else if (cnt\_end) begin  
 cnt <= 26'b0;  
 end  
 else if (cnt\_vld) begin  
 cnt <= cnt + 1'b1;  
 end  
 end  
  
 assign cnt\_vld = key\_mooth & (cnt < `LONG\_PRESSION - 1);  
 assign cnt\_end = (~key\_mooth & key\_mooth\_r1);  
  
//-------------------------------------------------------  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 short\_pression <= 1'b0;  
 end  
 else if (short\_pression) begin  
 short\_pression <= 1'b0;  
 end  
 else if ((cnt>=`SHORT\_PRESSION-1) & (cnt<`LONG\_PRESSION-1) & cnt\_end) begin  
 short\_pression <= 1'b1;  
 end  
 end  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 long\_pression <= 1'b0;  
 end  
 else if (long\_pression) begin  
 long\_pression <= 1'b0;  
 end  
 else if ((cnt==`LONG\_PRESSION-1) & cnt\_end) begin  
 long\_pression <= 1'b1;  
 end  
 end  
endmodule

### 5.5 **音乐播放代码**

#### 5.1 总体连接



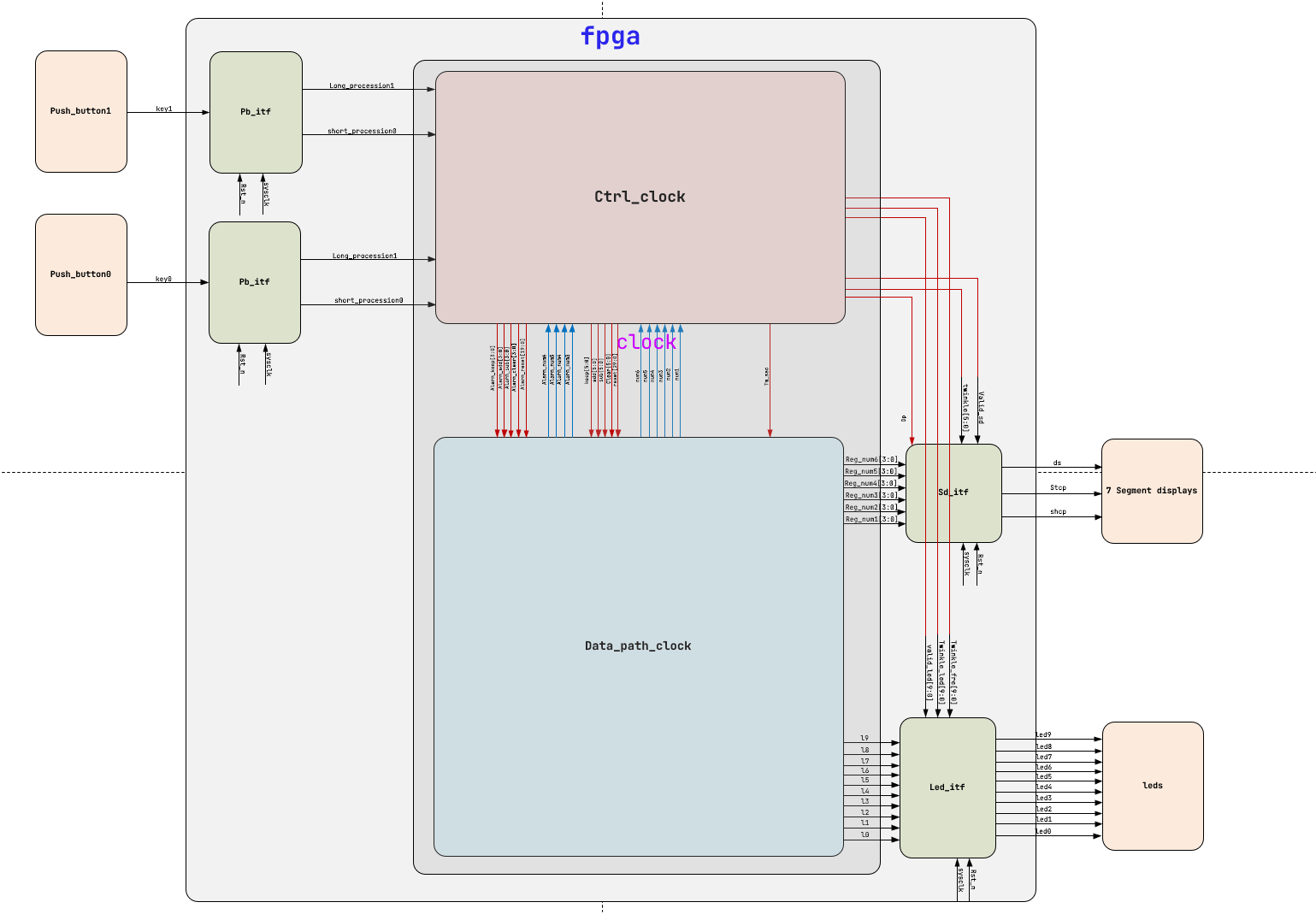
#### 5.2 控制波形



#### 5.3 实现代码

always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 start\_r1 <= `MSC\_N'b0;  
 end  
 else begin  
 start\_r1 <= start;  
 end  
 end  
  
 assign begins = (start\_r1 != start) & (start != 0);  
 assign ends = (start\_r1 != start) & (start == 0);  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 state <= IDLE;  
 end  
 else begin  
 state <= next\_state;  
 end  
 end  
  
 always @ (\*) begin  
 case (state)  
 IDLE:  
 if (idle\_to\_inst) begin  
 next\_state = INST;  
 end  
 else begin  
 next\_state = state;  
 end  
 INST:  
 if (inst\_to\_send\_ad) begin  
 next\_state = SEND\_AD;  
 end  
 else if (inst\_to\_idle) begin  
 next\_state = IDLE;  
 end  
 else begin  
 next\_state = state;  
 end  
 SEND\_AD:  
 if (send\_ad\_to\_inst) begin  
 next\_state = INST;  
 end  
 else if (send\_ad\_to\_idle) begin  
 next\_state = IDLE;  
 end  
 else begin  
 next\_state = state;  
 end  
 default:  
 next\_state = IDLE;  
 endcase  
 end  
  
 assign idle\_to\_inst = (state == IDLE) & (begins);  
 assign inst\_to\_send\_ad = (state == INST) & (cnt\_end);  
 assign inst\_to\_idle = (state == INST) & (ends);  
 assign send\_ad\_to\_inst = (state == SEND\_AD) & (begins | q == 6'b111000);  
 assign send\_ad\_to\_idle = (state == SEND\_AD) & (ends);  
   
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 cnt <= 25'b0;  
 end  
 else if (cnt\_end) begin  
 cnt <= 25'b0;  
 end  
 else if (cnt\_vld) begin  
 cnt <= cnt + 1'b1;  
 end  
 end  
  
 assign cnt\_vld = state != IDLE;  
 assign cnt\_end = cnt\_vld & (state == INST & cnt == 5) | ((state == SEND\_AD) & (cnt >= (60\*50\_000\_000/(4\*q\_cb)) - 1)) | ends;  
  
 always @ (posedge sysclk or negedge rst\_n) begin  
 if (~rst\_n) begin  
 q\_cb <= 12'b0;  
 end  
 else if (state == INST & cnt == 2) begin  
 q\_cb <= {6'b000000,q};  
 end  
 else if (state == INST & cnt == 4) begin  
 q\_cb <= {q\_cb[5:0],q};  
 end  
 else if (ends) begin  
 q\_cb <= 12'b0;  
 end  
 end  
  
//------------------------------------------------------------------  
  
 assign tran\_vld = (state == SEND\_AD & cnt\_end) & ~ends;  
 assign tran\_end = ends;  
  
//-------------------------------------------------------------------  
  
 assign rom\_clock = (state == INST & (cnt == 2 | cnt == 4))   
 | (state == SEND\_AD & cnt\_end);  
  
//-------------------------------------------------------------------  
  
 assign add = (state == INST & cnt == 2)  
 | (state == SEND\_AD & cnt >= 60\*50\_000\_000/(4\*q\_cb) - 3 & cnt < 60\*50\_000\_000/(4\*q\_cb) - 2);  
  
 always @ (\*) begin  
 if (ends) begin  
 sel = `MSC\_N'b1;  
 end  
 else if (state == INST & cnt == 0) begin  
 sel = start;  
 end  
 else begin  
 sel = `MSC\_N'b0;  
 end  
 end  
  
 assign volume\_out = volume\_in;

### 5.6 **总体连接**

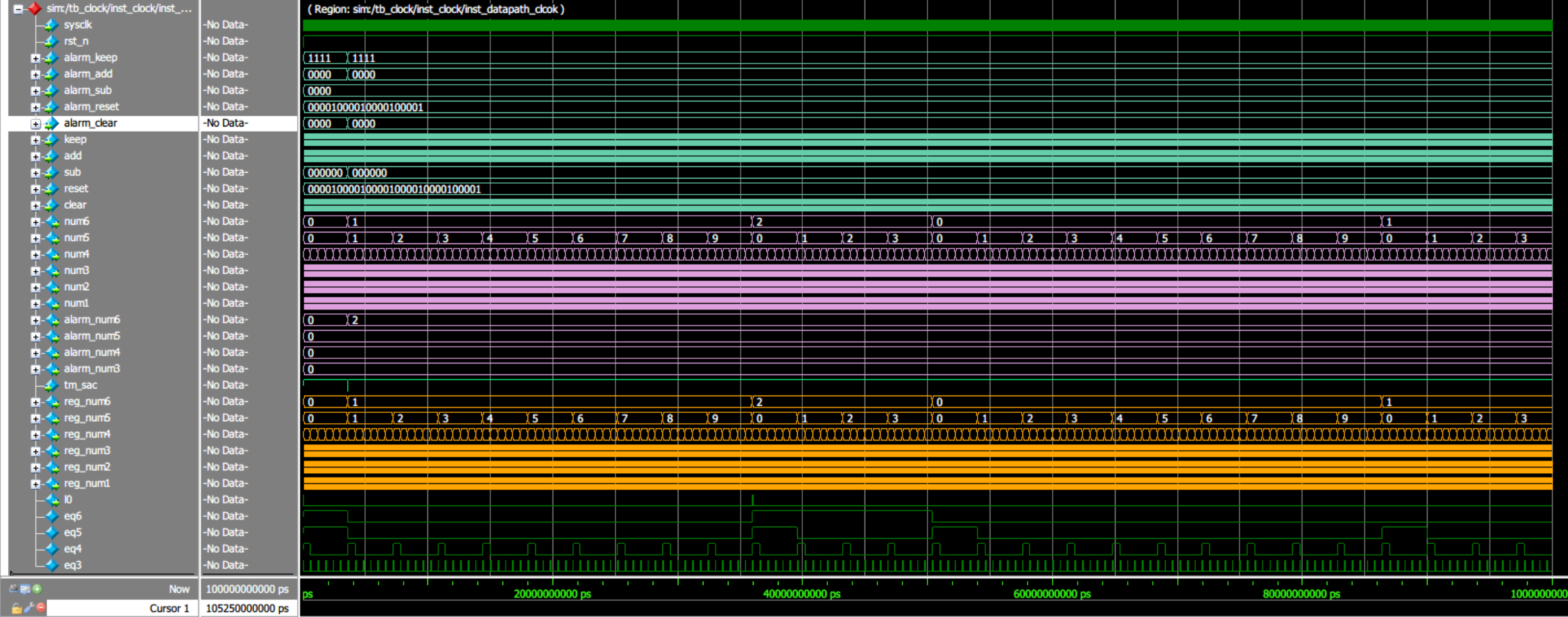


## 6. **仿真结果**

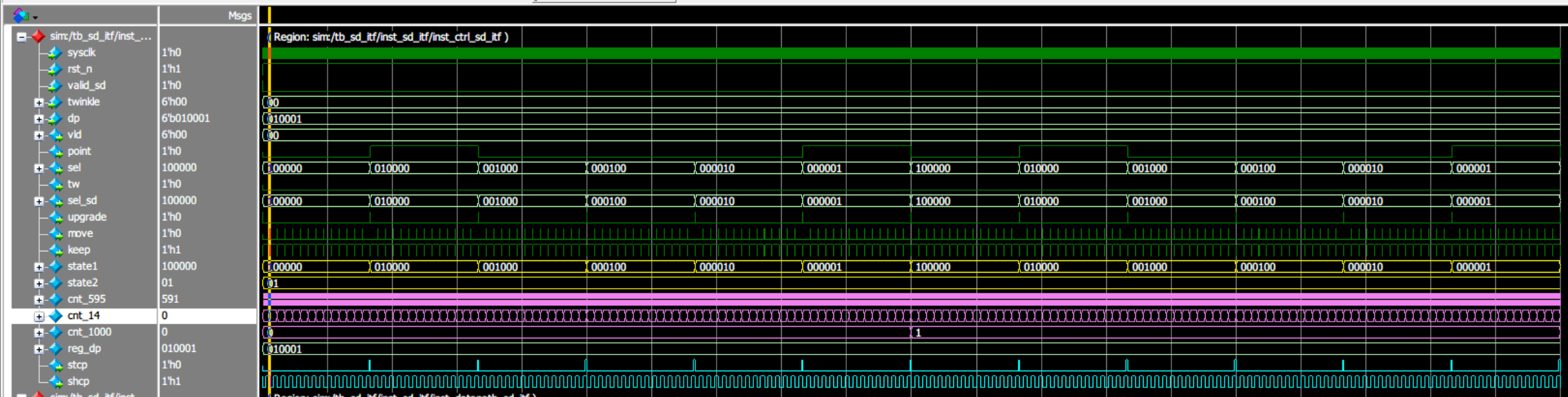
通过 quartus2 与 modelsim 联合仿真

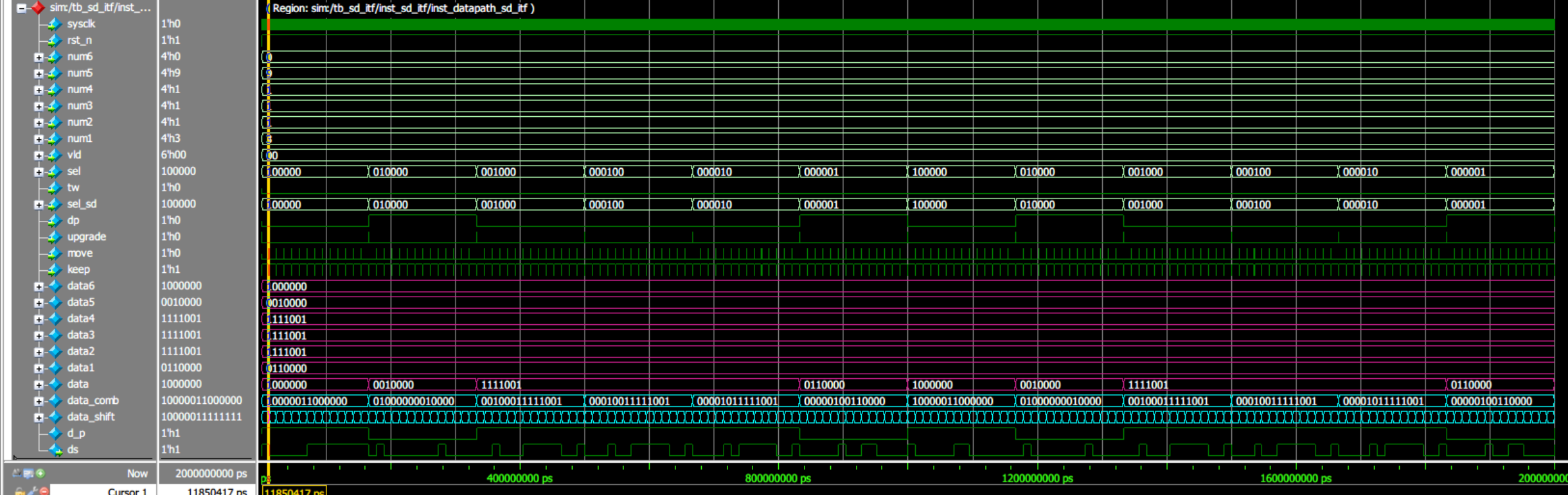
### 6.1 **时钟主体仿真**



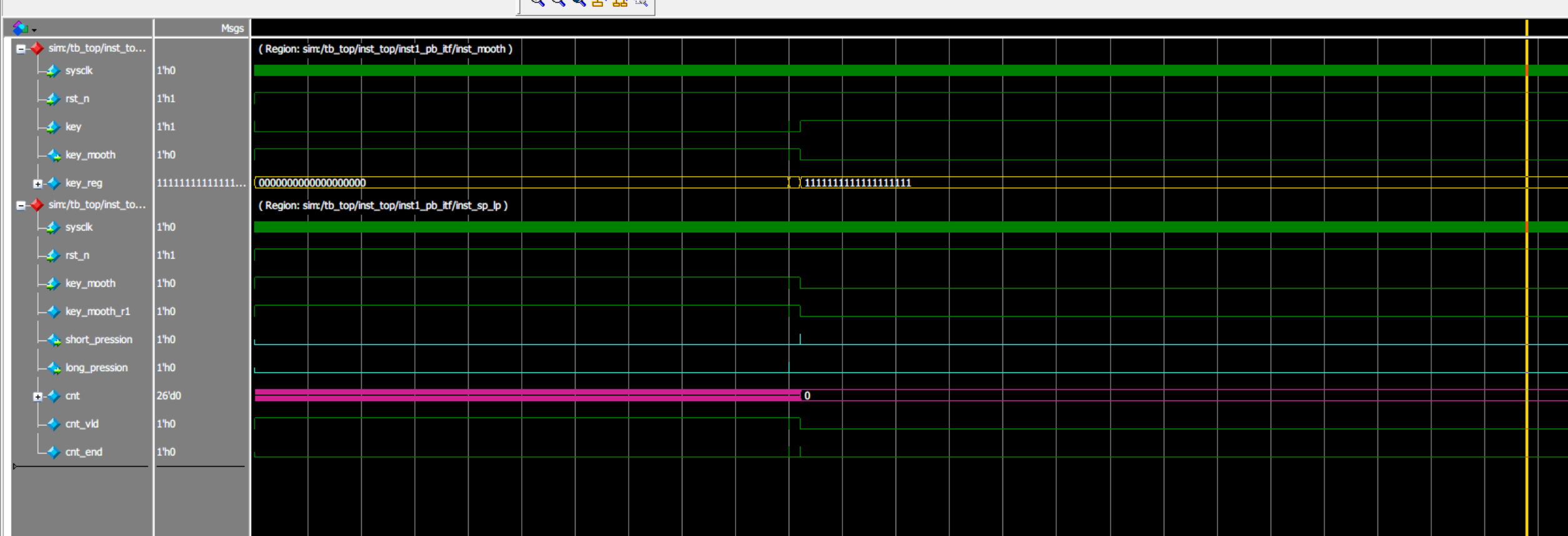


### 6.2 **数码管显示接口仿真**

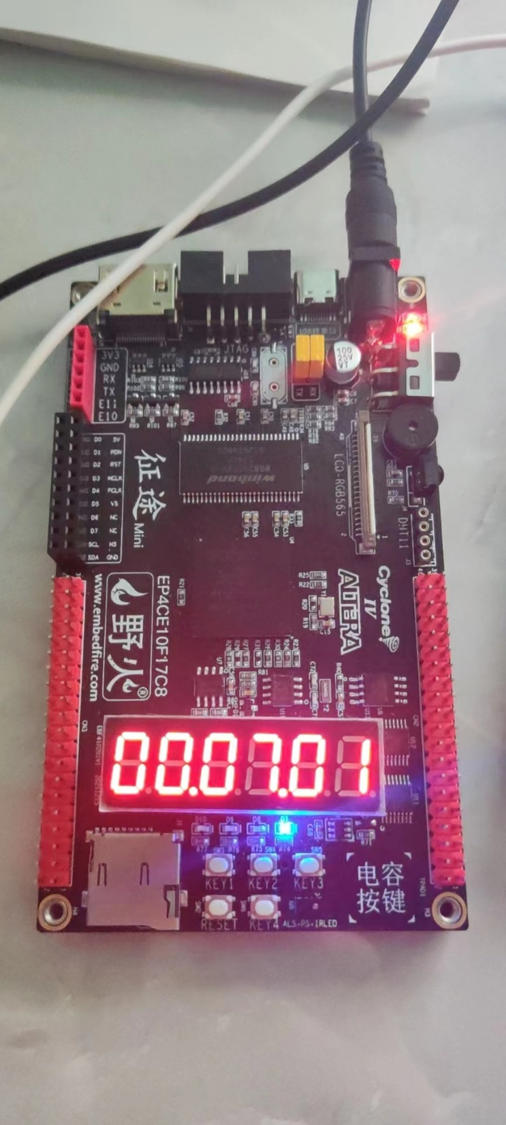


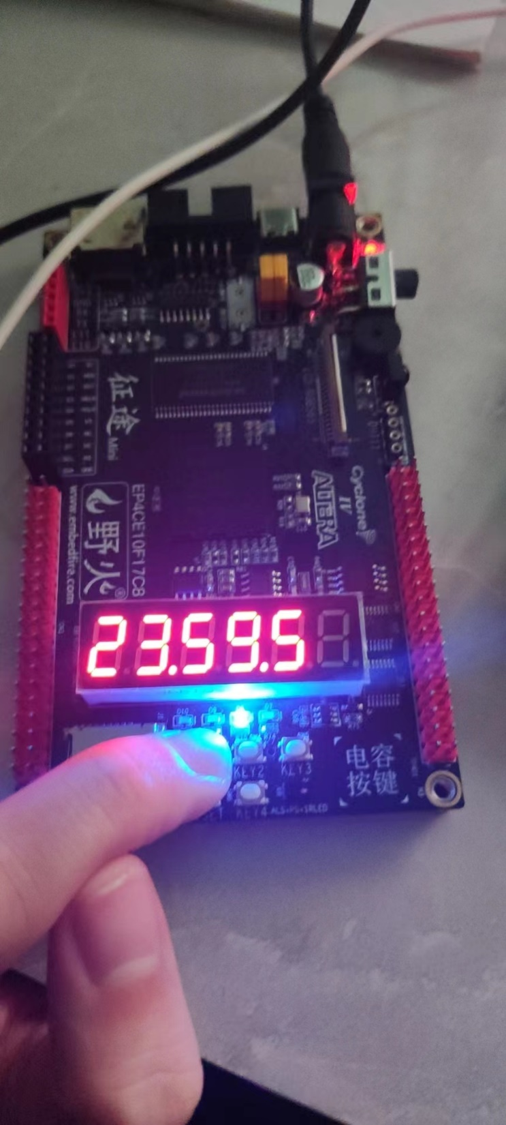


### 6.3 **按键消抖仿真**



## 7. **上板测试**





## 8. **实验心得**

1. 根据实现的功能进行划分模块，注意模块间低耦合，模块内高内聚。可以有效地提升模块的复用性与可拓展性
2. 分析每一个模块的类型，可以吧模块分成数据通路 和 控制通路，两者都包含的就是 复合通路。分析的时候先确定模块的输入输出有哪些，以控制和数据为标准将输入输出分类
3. 先数据后控制，先确定输入的数据需要经过怎么样的处理才可以得出输出数据，数据的输入到输出需要哪些辅助的控制信号，数据的输入到输出需要几个时钟周期，可以采用流水线的方式构造数据通路
4. 根据数据通路中确定的控制信号，以及数据通路的输出信号，对它们的行为变化进行分析。从而确定大致架构。如状态机，计数器。输出信号以架构为根，辅助一些中间型号进行控制
5. 数据通路的框图决定数据通路的代码。控制通路的波形决定控制通路的代码
6. 通过modelsim仿真。查看关键信号时候正确，尤其是架构。