

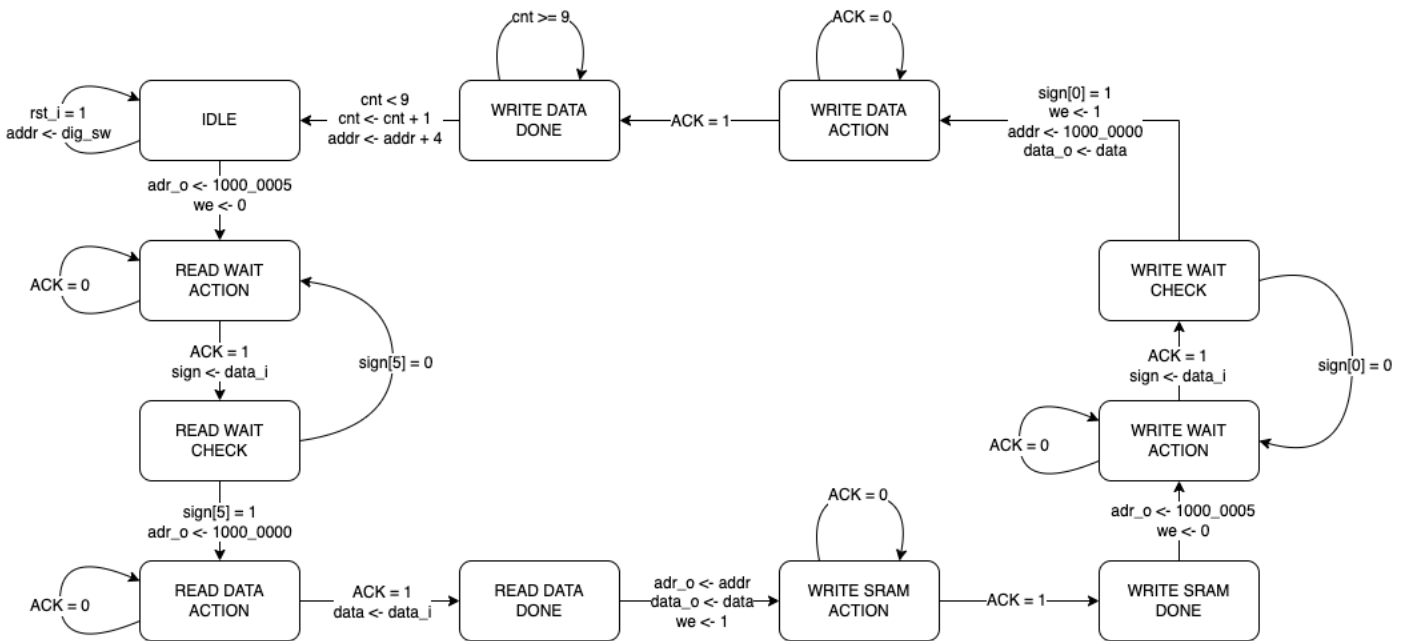
# 计算机组成原理 · 实验5报告

计01 容逸朗 2020010869

## 实验过程

### 状态机设计

具体设计如下所示：（其中  $\leftarrow$  是赋值操作， $=$  或  $<$  是跳转判断条件）



### 代码实现

本次任务只需要按照上面的状态机写代码即可，此处不再详述。

### 接口

需要加入拨码开关 `input wire [31:0] dip_sw`，然后增加本地寄存器用于记录数据：

```
1 reg [31:0] addr; // SRAM 地址
2 reg [31:0] sign; // 控制信号
3 reg [31:0] data; // 串口读入的数据
4 reg [3:0] cnt = '0; // 计数器
```

### 状态转移

由于状态数较多，这里仅包含部分代码：

```
1 always_comb begin
2     case (state)
3         ST_IDLE: begin
```

```

4      state_n = ST_READ_WAIT_ACTION;
5  end
6
7  ST_READ_WAIT_ACTION: begin
8      if (wb_ack_i == 1) begin
9          state_n = ST_READ_WAIT_CHECK;
10     end else begin
11         state_n = ST_READ_WAIT_ACTION;
12     end
13 end
14
15 ST_READ_WAIT_CHECK: begin
16     if (sign[0]) begin
17         state_n = ST_READ_DATA_ACTION;
18     end else begin
19         state_n = ST_READ_WAIT_ACTION;
20     end
21 end
22
23 // 还有更多...
24
25 endcase
26 end

```

## 数据处理

我采用了三段式的状态机实现方式，因此需要用下面的方式调整参数：

```

1  always_ff @(posedge clk_i) begin
2      if (rst_i) begin
3          addr <= dip_sw;
4          cnt <= '0;
5          sign <= '0;
6          data <= '0;
7      end else begin
8          case (state)
9              ST_IDLE: begin
10                 wb_adr_o <= REG_STATUS;
11             end
12
13             ST_READ_WAIT_ACTION: begin
14                 wb_adr_o <= REG_STATUS;
15                 sign <= wb_dat_i;
16             end
17
18             ST_READ_WAIT_CHECK: begin
19                 if (sign[0]) begin
20                     wb_adr_o <= REG_DATA;
21                 end else begin
22                     wb_adr_o <= REG_STATUS;

```

```
23         end
24     end
25
26     // 还有更多 ...
27
28     endcase
29 end
30 end
```

## 实验总结

本次实验中，我共提交了八次评测才通过本次实验：(

- 第一次尝试：写好代码后，粗略看了一遍代码觉得没啥问题，逐提交，果不其然没有通过。
- 第二次尝试：仔细察看代码，发现状态转移逻辑有一处漏洞，改之，再提交，喜提零分。
- 有了前两次的经验后，我决定写一个仿真程序。运行仿真程序后发现有部分状态转移不到，于是再次观察代码，发现有一处的状态转移的名字写错了，改之再仿真，看到状态机正确运行，于是再次提交，再得零分。
- 使用云平台测试，看到 BaseRam 的内容完全正确，但串口只有五个数输出，心想是平台出问题了，故提交评测，再夺零分。
- 猜想可能是时序的问题，于是改为收到 `wb_ack_i` 后就把 `wb_cyc_o` 置零而非等待下一周期的写法（避免连续读写串口使有用信息被复盖），观察仿真时序正常便再次提交之，又没过。
- 实在想不到其他原因，于是尝试把时序和状态机周期对齐，还是没通过。
- 毫无思路下推倒重来，先手写状态机和转移图，再检查，发现仿真和预想的结果一样，因而上传到云平台测试，看到内存正确无误，但提交后仍是串口超时。
- 猜想问题可能出自串口，于是仿真时顺便把 `uart_controller` 的信号加入，看到 `txd_busy = 1` 的情况下 `wb_data_i` 的第五位仍为 1，即此时能写入信息，逐觉不妙，于是观察 `uart_controller` 的实现，发现在 `WRITE_WAIT_ACTION` 状态时读取 `controller` 状态时应把 `wb_we_o` 置为零（读），改之，仿真无误，上板测试也能接收到串口的所有内容。大喜，提交之，终获一百，实在是可喜可贺。

总结：写代码前应当看清楚用法，而非看见状态机名字就觉得 `WRITE` 状态对应的读写使能为写的状态。以及评测平台的可靠性良好，应该多加信任而非无视。