计算机组成原理 · 实验4报告

计01　容逸朗　2020010869

实验过程

状态机

状态机的设计在实验文档中已经给出，按照思路容易写出下面的代码：

```
// reset and state change
always_ff @(posedge clk_i or posedge rst_i) begin
  if (rst_i) begin
    state <= ST_IDLE;
  end else begin
    state <= state_n;
  end
end

// states
always_comb begin
  state_n = state;
  case (state)
    ST_IDLE: begin
      if (wb_stb_i && wb_cyc_i) begin
        if (wb_we_i) begin
          state_n = ST_WRITE;
        end else begin
          state_n = ST_READ;
        end
      end
    end

    ST_READ: begin
      state_n = ST_READ_2;
    end

    ST_READ_2: begin
      state_n = ST_DONE;
    end

    // 其他省略...

    default: begin
      state_n = ST_IDLE;
    end
```

```
  endcase
end
```

信号处理

由于状态较为简单，因此选用组合逻辑实现：

```
// signals
always_comb begin
  sram_ce_n = (state == ST_IDLE) || (state == ST_DONE);
  sram_oe_n = !((state == ST_READ) || (state == ST_READ_2));
  sram_we_n = !(state == ST_WRITE_2);
  wb_ack_o = (state == ST_DONE);
  sram_addr = wb_adr_i[21: 2];
  sram_be_n = (state == ST_WRITE || state == ST_WRITE_2 || state == ST_WRITE_3) ? ~wb_sel_i : '0;
end

// data processing
assign sram_data = sram_we_n ? 32'bz : wb_dat_i;
assign wb_dat_o = sram_data;
```

唯一需要注意的是 SRAM 的单位是 4 字节而非总线的 1 字节。

思考题

> 1. 静态存储器的读和写各有什么特点？

- SRAM 的读和写过程都是通过读写信号控制，且读写共用一条数据线。
- SRAM 的数据是以行为单位存储的，每一行有 32 列（视位数而定，此处为 32 位），因此需要额外的时间来找到地址对应的行才能进行下一步的读 / 写操作。

> 2. 什么是 RAM 芯片输出的高阻态？它的作用是什么？

- 高阻态是电路的一种输出状态，既不是高电平也不是低电平，类似于引脚悬空的情况。
- RAM 芯片输出高阻态是因为 RAM 芯片为了节省引脚数量，导致了同一个信号在不同的时间传输不同方向的数据的现象。此时为了防止两端设备同时输出造成数据不稳定的情况，因此设备在不输出信号时需要设置高阻态。

> 3. 本实验完成的是将 BaseRAM 和 ExtRAM 作为独立的存储器单独进行访问的功能。如果希望将 Base_RAM 和 Ext_RAM 作为一个统一的 64 位数据的存储器进行访问，该如何进行？

- 由于 Base_RAM 和 Ext_RAM 都是 32 位的，如果要作为一个统一的 64 位数据的存储器访问，可以进行如下操作：
  - 地址缩小为 0x0 - 0x3FFFFF，且每个地址都由 Base_RAM 和 Ext_RAM 共用；
  - 数据的高 32 位存入 Base_RAM 对应地址行，低 32 位存入 Ext_RAM 的同一行；
  - 更改 1M2S 复用器接口，在此处对数据进行分割 / 合并操作。每次操作时，除了数据线以外需要给 Base_RAM 和 Ext_RAM 相同的信号，这样才能正确进行读 / 写操作，当两者的操作均完成后才返回 `ack_o = 1`。

实验总结

本次实验中，我只提交了一次评测就通过本次实验了:-)

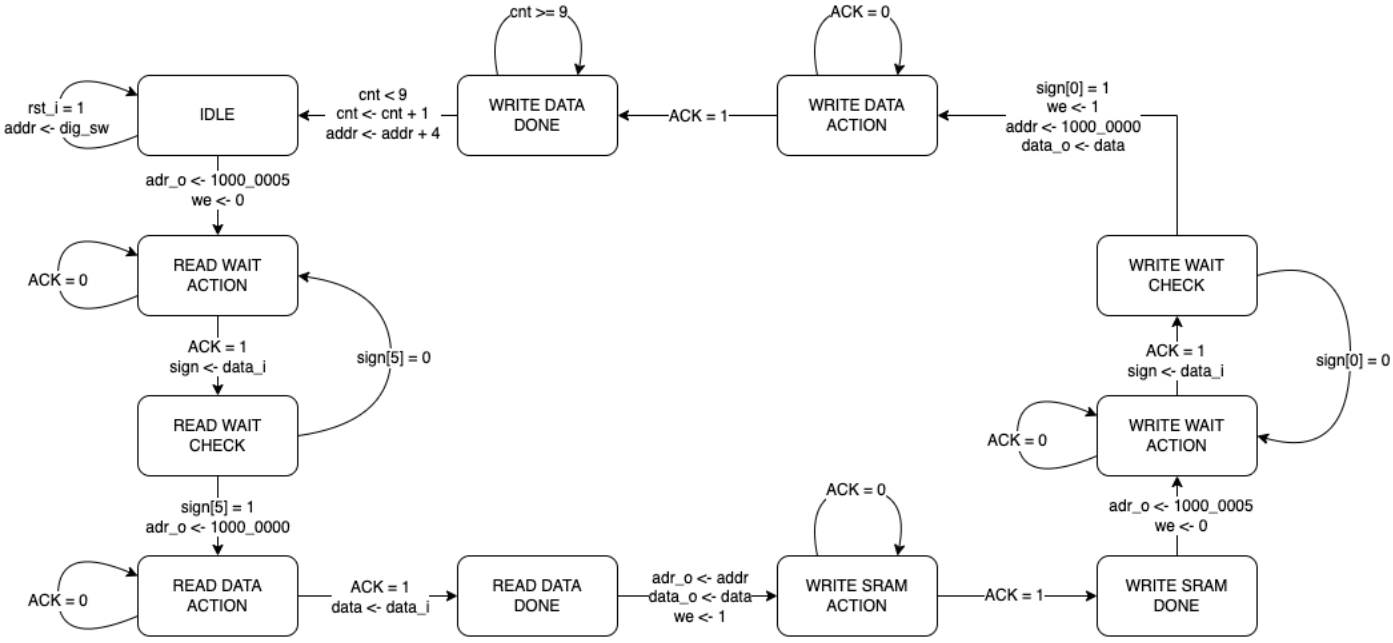
计算机组成原理 · 实验5报告

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 ← 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：

```
reg [31:0] addr; // SRAM 地址
reg [31:0] sign; // 控制信号
reg [31:0] data; // 串口读入的数据
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：

```
always_comb begin
   case (state)
      ST_IDLE: begin
```

```
        state_n = ST_READ_WAIT_ACTION;
      end

      ST_READ_WAIT_ACTION: begin
        if (wb_ack_i == 1) begin
          state_n = ST_READ_WAIT_CHECK;
        end else begin
          state_n = ST_READ_WAIT_ACTION;
        end
      end

      ST_READ_WAIT_CHECK: begin
        if (sign[0]) begin
          state_n = ST_READ_DATA_ACTION;
        end else begin
          state_n = ST_READ_WAIT_ACTION;
        end
      end

     // 还有更多...

   endcase
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：

```
always_ff @(posedge clk_i) begin
  if (rst_i) begin
    addr <= dip_sw;
    cnt <= '0;
    sign <= '0;
    data <= '0;
  end else begin
    case (state)
      ST_IDLE: begin
        wb_adr_o <= REG_STATUS;
      end

      ST_READ_WAIT_ACTION: begin
        wb_adr_o <= REG_STATUS;
        sign <= wb_dat_i;
      end

      ST_READ_WAIT_CHECK: begin
        if (sign[0]) begin
          wb_adr_o <= REG_DATA;
        end else begin
          wb_adr_o <= REG_STATUS;
```

```
        end
      end

      // 还有更多...

    endcase
  end
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。