国科大计算机组成原理实验
prj3-内存及外设通路设计与处理器性能评估

李国峰 2018K8009922027

2021年6月6日

实验设计

改进访存接口

严格按照PPT给出的状态转移图进行状态机代码的书写。三段式状态机写法是：

第一段是`current_state`和`next_state`的更新机制，在`rst`释放后，每个时钟上升沿都将`current_state`更新为`next_state`的值。

```
always@(posedge clk) begin
    if(rst)
        current_state <= `INIT;
    else
        current_state <= next_state;
end
```

第二段利用always和switch语句的组合逻辑书写`next_state`的更新逻辑，需要注意的是赋值条件里不能出现`rst`。这部分代码的结构是显而易见的，PPT给出了示例。

```
always@(*) begin
    case(current_state)
        `INIT  : begin
            next_state = `IF;
         end

        `IF    : begin
            if(Inst_Req_Ready)
                next_state = `IW;
            else
                next_state = `IF;
        end

        `IW    : begin
            if(Inst_Valid)
                next_state = `ID;
```

```
            else
                next_state = `IW;
        end

        `ID    : begin
            if(nop)
                next_state = `IF;
            else
                next_state = `EX;
        end

        `EX : begin
            if(b_j) begin
                if(inst_jal || inst_jalr || inst_jr)
                    next_state = `WB;
                else
                    next_state = `IF;
            end

            else if(store)
                next_state = `ST;
            else if(load)
                next_state = `LD;
            else
                next_state = `WB;
        end

        `ST    : begin
            if(Mem_Req_Ready)
                next_state = `IF;
            else
                next_state = `ST;
        end

        `LD    : begin
            if(Mem_Req_Ready)
                next_state = `RDW;
            else
                next_state = `LD;
        end

        `RDW   : begin
            if(Read_data_Valid)
                next_state = `WB;
            else
                next_state = `RDW;
        end

        `WB    : begin
            next_state = `IF;
        end

        default : next_state = `INIT;
    endcase
end
```

第三段是用`current_state`作为控制信号的判断条件之一，即在上个实验的基础上，将`current_state`加入到那些需要在指定阶段产生变化的控制信号的生成逻辑中即可。

```
assign Inst_Req_Valid = current_state == `IF;

assign Inst_Ready = current_state == `INIT || current_state == `IW;

assign RF_wen = (inst_addiu | inst_addu  | inst_subu   | inst_and  |
                 inst_andi  | inst_nor   | inst_or     | inst_ori  |
                 inst_xor   | inst_xori  | inst_slt    | inst_slti |
                 inst_sltu  | inst_sltiu | inst_sll    | inst_sllv |
                 inst_sra   | inst_srav  | inst_srl    | inst_srlv |
                 inst_jal   | inst_jalr  | inst_lb     | inst_lh   |
                 inst_lw    | inst_lbu   | inst_lhu    | inst_lwl  |
                 inst_lwr   | inst_lui   | move_to_reg ) && current_state ==
`WB;

assign MemWrite = (inst_sw | inst_sb | inst_sh | inst_swl |
                   inst_swr) && current_state == `ST;

assign MemRead  = (inst_lw  | inst_lb  | inst_lbu | inst_lh |
                   inst_lhu | inst_lwl | inst_lwr) && current_state == `LD;

assign Read_data_Ready = current_state == `RDW || current_state == `INIT;
```

状态机的设计结束后，需要根据状态的转移调整`PC`的更新逻辑。在多周期处理器中，将下一条指令的`PC`的生成放在译码级（`ID`）。

```
always@(posedge clk) begin
    if(rst)
        next_pc <= 32'h00000000;
    else begin
        if(current_state == `ID) begin
            if(br_go)
                next_pc <= br_target;
            else
                next_pc <= PC + 4;
        end
    end
end
```

为了便于`PC`在`current_state`刚好进入`IF`时更新，所以PC更新判断条件为`next_state`的状态。

```
always@(posedge clk) begin
    if(rst)
        PC <= 32'h00000000;
    else begin
        if(next_state == `IF) begin
            if(nop)
                PC <= PC + 4;
            else
                PC <= next_pc;
        end
    end
end
```

由于指令和数据RAM输出的数据只持续一个周期，因此需要用触发器将RAM输出的数据储存起来。

```
always@(posedge clk) begin
    if(current_state == `IW && Inst_Valid)
        inst_reg <= Instruction;
end

always@(posedge clk) begin
   if(current_state == `RDW && Read_data_Valid)
        read_data_reg <= Read_data;
end
```

UART控制器驱动

按照流程图实现代码即可。

```
int
puts(const char *s)
{
    //TODO: Add your driver code here
    int i = 0;

    while(s[i] != '\0')
    {
        while(*(uart + UART_STATUS / 4) & UART_TX_FIFO_FULL)
            ;

        *(uart + UART_TX_FIFO / 4) = s[i];
        i++;
    }

    return i;
}
```

需要解释的是相应参数除以4的问题。因为`uart`是一个`int`类型的指针，指向一块长为4字节的空间，因此`uart+1`就代表了相对`uart`的偏移量为4字节，因此需要除以4。

性能计数器硬件实现和软件访问

硬件实现是直观的。

```
reg [31:0] Cycle_cnt;

always@(posedge clk) begin
    if(rst)
        Cycle_cnt <= 32'b0;
    else
        Cycle_cnt <= Cycle_cnt + 32'b1;
end

assign cpu_perf_cnt_0 = Cycle_cnt;
```

只实现了一个周期计数器，因此只需要添加 `_uptime` 的核心代码以及在 `main` 函数中添加 `printk` 语句即可。 `_uptime` 需要实现的功能为从CPU的相应端口（cpu_perf_cnt_0）读出计数的周期数。相应访问方法和第二部分一致。

```
unsigned long _uptime() {
  unsigned long timing;
  unsigned long *perf_cnt_addr = (void*)0x40020000;
  timing = *perf_cnt_addr;
  return timing;
}
```

存在的问题

在bhv_sim中的31_hello中可以成功打印出字符串，但是在hello_fpga_eval中无法打印出。但可以通过microbench_fpga_eval。不清楚问题所在，问题可能出在硬件代码中。