URISC 设计报告

集电硕 21 2021211066 方佳豪

一、 设计目标

设计一个 8bit 的 URISC 处理器，由数据通路，控制单元，以及 RAM 三部分构成。数据通路，控制单元，RAM 宽度均为 8 比特（一个字节 byte）。

二、 总体架构

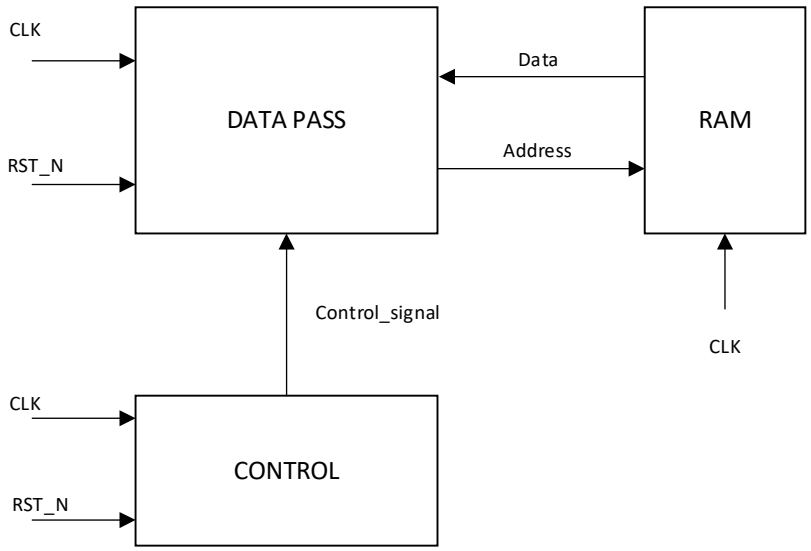

图 1 URISC 结构框图

2.1 数据通路

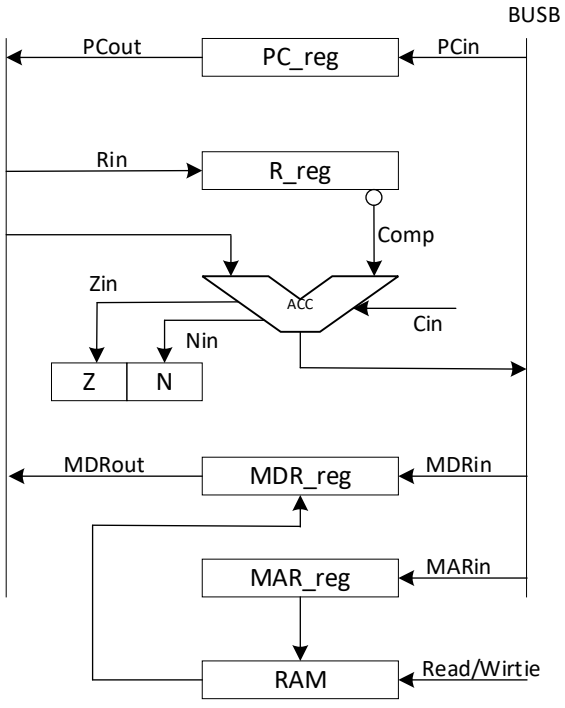

图 2 URISC 数据通路

2.2 控制单元逻辑

State 0:将 PC 值经 BUSA、ACC、BUSB 传给 MAR，存储器读取该地址的数据并寄存到 MDR，同时判断 PC 是否为 0。 这条微指令的功能是从 RAM 中读取指令的第一个字节，即地址 1。

State 1:将 MDR 的值经 BUSA、ACC、BUSB 传给 MAR 和地址线，存储器读取该地址的数据并寄存到 MDR。作用是读取指令中地址 1 的数据。

State 2:将 MDR 的值经 BUSA 传给寄存器 R。作用是将指令地址 1 的数据暂存起来以备使用。

State 3:将 PC 的值经 BUSA、ACC、BUSB 加 1 后传递回 PC 和 MAR，存储器读取该地址的数据并寄存到 MDR。功能是 PC+1，读取指令的第二个字节，即地址 2。

State 4:同 State 1，读取地址 2 的数据。

State 5:将 MDR 的值与 R 的值相减并寄存到 MDR，同时判断结果是否为负。随后将 MDR 的值写入存储器，写入的地址为 MAR，即地址 2。功能是将指令中地址 2 的数据减去地址 1 的数据，结果存回地址 2。

State 6:同微指令 3，PC+1 读取指令的第三个字节，即跳转地址并寄存到 MDR。

State 7:将 PC+1 并写回 PC，准备读取接下来一条指令。

State 8:将 MDR 的值写入 PC，准备读取跳转到的指令。

2.3 控制信号状态表

State	MDRout	MARin	Nin	Rin	PCin	ZEND	Cin	WRITE	NNEND
0	0	1	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0	0
2	1	0	0	1	0	0	0	0	0
3	0	1	0	0	1	0	1	0	0
4	1	1	0	0	0	0	0	0	0
5	1	0	1	0	0	0	1	1	0
6	0	1	0	0	1	0	1	0	0
7	0	0	0	0	1	0	1	0	1
8	1	0	0	0	1	0	0	0	0

2.4 RAM 设计

实际一条完整的指令执行，PC_next=PC_current+3。因而在 RAM 设计时，位宽为一个字节即可，这样连续的三个地址组成一条完整的指令。

三、　具体实现

两相时钟,均为上升沿触发。Clk_PH1 控制时序跳转以及控制信号的跳转, Clk_PH2 控制 MAR 寄存器的写入;异步复位，增加状态位 IDLE，复位后，IDLE 自动跳转到 state 0 进行计算；若不增加 IDLE 状态，需要在 Clk_PH1 的上升沿后，下降沿之前复位，这样功能才正常进行。

3.1 时序图

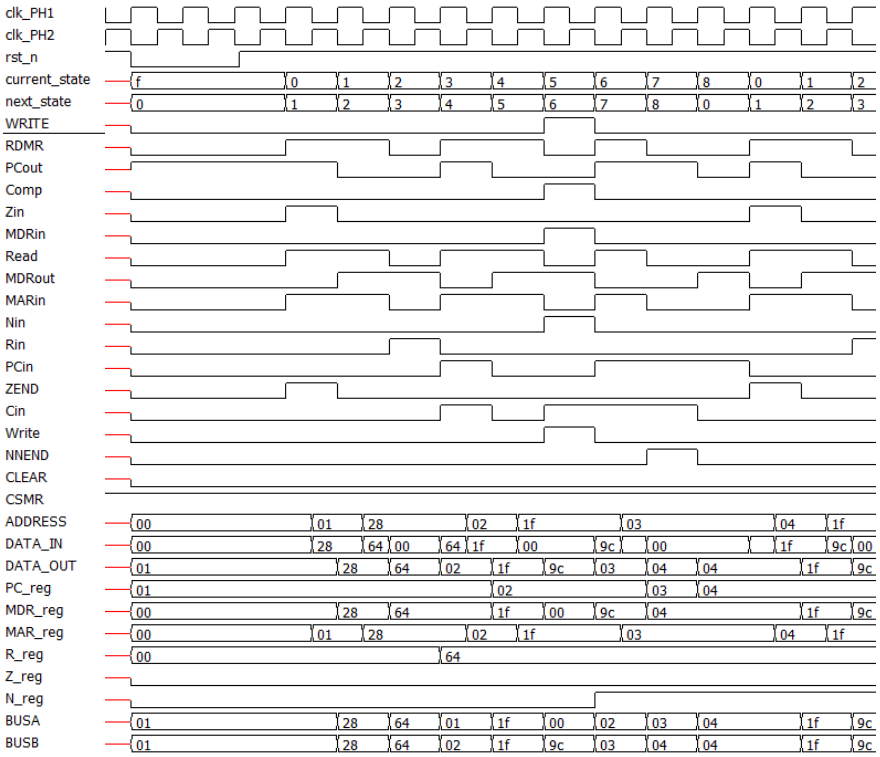

图 3 URISC 时序图

3.2 总线

由多路选择器实现，不增加高阻 Z 状态；

```
assign A_bus=MDRout?MDR:(PCout?PC:8'b00000000);
assign B_bus=Acc_Res;
```

3.3 控制信号产生

由 ROM 实现。

```
module ROM_state(
    input [3:0] addr,
    output [8:0] dataout
    );
    wire [8:0] rom[15:0];
    assign rom[0] = 9'b0100_0100_0;
    assign rom[1] = 9'b1100_0000_0;
    assign rom[2] = 9'b1001_0000_0;
    assign rom[3] = 9'b0100_1010_0;
```

```
        assign rom[4] = 9'b1100_0000_0;
        assign rom[5] = 9'b1010_0011_0;
        assign rom[6] = 9'b0100_1010_0;
        assign rom[7] = 9'b0000_1010_1;
       assign rom[8] = 9'b1000_1000_0;
……
       assign rom[15] = 9'b0000_0000_0;// IDLE state for reset
        assign dataout =rom[addr];
endmodule
```

3.4 RAM

数据异步读出，同步写入。

```
module RAM (
      input clk,
      input rst_n,
      input CS,
      input WRITE,
      input READ,
      input [7:0] ADDRESS,
      input [7:0] WDATA,
      output [7:0] RDATA
);
reg[7:0]    uram[0:127];
assign RDATA=(CS&READ) ?uram[ADDRESS]:8'b0;
wire [7:0] test_x;//check the value of X register
assign test_x=uram[35];
wire [7:0] test_z;//check the value of Z register
assign test_z=uram[36];
always @(posedge clk or negedge rst_n) begin
      if(!rst_n)begin
            /**the initialization    of the    instruction and data    about computing (X+Y)/2**/
            /*input X=16 Y=100    output Z=58*/
                  uram[0]<=8'h00;//STOP
                  uram[1]<=8'h28;//READY Y_addr=40
                  uram[2]<=8'h1F;//READY TEMP1=31
                  uram[3]<=8'h04;//READY NEXT1=4
                  uram[4]<=8'h1F;//NEXT1 TEMP1=31
                  uram[5]<=8'h23;//NEXT1 X_addr=35
                  uram[6]<=8'h07;//NEXT1 NEXT2=7
                  uram[7]<=8'h24;//NEXT2 Z_addr=36
                  uram[8]<=8'h24;//NEXT2 Z_addr=36
                  uram[9]<=8'h0A;//NEXT2 TEST=10
                  uram[10]<=8'h23;//TEST X_addr=35
                  uram[11]<=8'h20;//TEST TEMP2=32
```

```
                uram[12]<=8'h16;//TEST POSITIVE=22
                ……
    end
        else if(WRITE & CS)
        uram[ADDRESS]<=WDATA;
    end
endmodule
```

三、 仿真综合

4.1 功能仿真

实现计算 Z=(X+Y)/2，其中 X=16,Y=100，若功能正确，Z 对应的地址存放的数据为 58;仿真数据直接存放于 RAM，通过复位信号赋初值。通过读取 ram[36]（需要引出端口 test_z）的数据，如图所示，当程序 PC 跳转到 0，程序结束后，test_z 的值为 58，表明运算结果正确。

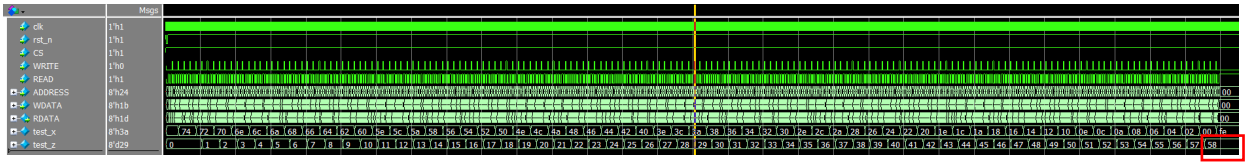

图 4 Modelsim 仿真 RAM 读取

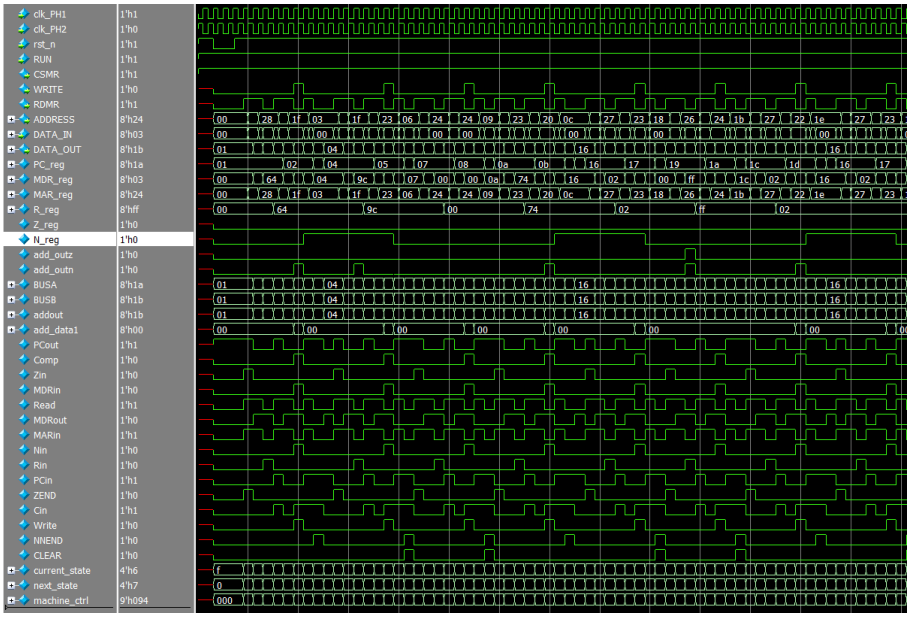

图 5 Modelsim 仿真 URSIC 信号一览图

4.2 DC 综合

单元库：tt_1v8_25c ,将 clk_PH2 定义为 clk_PH1 取反，进行综合，将 clk_PH1 定义为 clk_PH2 取反，进行综合。取延迟的最大值。
时序约束文件见 run.tcl。设定时钟周期为 3.5ns,约束由以下代码给出：

```
set CLK_PERIOD 3.5 //设置时钟周期
create_clock [get_ports $CLK] -period $CLK_PERIOD
set_dont_touch_network [all_clocks]
set_clock_uncertainty [expr $CLK_PERIOD * 0.001] [all_clocks] //设置时钟抖动
set_clock_transition 0.01 [all_clocks] //设置时钟偏差
set all_in_except_clk [remove_from_collection [all_inputs] [get_ports $CLK]]
set_input_delay [expr $CLK_PERIOD * 0.0] -clock $CLK $all_in_except_clk //输入延迟设为 0
```

综合性能如下表，时钟周期为 3.5ns

性能	结果
面积/um^2	5495.212843
延迟/ns	1.53
功耗/mW	1.4436

具体综合脚本以及结果见对应文件。

4.3 RTL 级仿真

根据 DC 综合出来的网表 URISC_map.v 和 URISC.sdf 文件，进行网表级仿真。所需 umc18.v 文件。

在测试文件中加入以下代码，进行时序反标：

```
$sdf_annotate("URISC.sdf",ursic,,"sdf.log");
```

并且设置半时钟周期为 1.8ns：

```
forever begin
#1.8 PH1=~PH1;
End
```

terminal 运行以下指令：

```
ncverilog
    +access+wrc
    +gui
    tb_TOP_URISC.v
```

得到如下时序图，可以看到数据翻转过程中比前仿多了毛刺。

图 6 NcVerilog 仿真 URSIC 信号一览图

功能验证由以下代码给出：

```
always @(posedge clk ) begin
      if((CS&READ&&ADDRESS==0))begin
     $fwrite(file,"(X+Y)/2=%d\n",uram[36]);
end
```

当 URISC 开始读寄存器 0 时，表明程序结束，这个时候将计算结果所在的 36 号寄存器的值输出即可。本人实现的 RAM 并没有进行综合，只对 URSIC 本身进行综合。

四、 思考总结

本次设计与课程讲述 URISC 在细节上存在差异：

首先，本人多定义了一个 IDLE 状态，使得异步复位在任何时钟状态下均能复位且程序正常运行；否则需要进行同步复位，一般在综合中，同步复位需要额外的逻辑单元。

二是时序存在细节差异，本人两相时钟均在上升沿触发，总的来讲，以 clk_PH1 为基准，上升沿完成所有控制信号的赋值，因而对于 MAR 来讲，只能在下降沿读出 PC 经由 ACC 传过来的数据。即所有逻辑滞后半拍。

三是 RTL 级仿真以前用的比较少，这次大作业进行了流程上的训练。