CODH - 单周期 CPU 设计 实验报告

院系: 姓名: 学号:

2023 年 7 月 13 日

1 实验目的

- 理解单周期 CPU 的结构和工作原理
- 掌握单周期 CPU 的设计和调试方法
- 熟练掌握数据通路和控制器的设计和 Verilog 描述方法

2 实验环境

- macOS 13.0
- Rars1_5.jar (Riscv Assembler and Runtime Simulator)
- Vivado 2019.3
- Nexys 4 DDR 开发板

3 实验内容

3.1 设计 CPU 的数据通路（具备简单的 I/O 功能）和控制器

- 指令存储器和数据存储器均采用 IP 例化的分布式存储器，容量均为 1024x32 位，使用 LabH3 实验生成的 COE 文件初始化
- 寄存器堆增加一个用于调试的读端口，指令存储器和数据存储器均增加一个用于调试和初始化的读/写端口
- Led 指示灯：输出程序运行状态或结果，MMIO 地址 0x3f00
- 时钟计数器：除复位清零外，一直对 CPU 工作时钟递增计数，用于测试程序执行时间，MMIO 地址 0x3f20

3.2 将 CPU 和 SDU 整合后下载至 FPGA，进行逐条指令功能测试与进行排序程序测试

- Led 指示灯显示测试结果，或者排序程序耗费时钟周期数
- 查看电路资源使用情况和电路性能

之后将指令和数据存储器用 LabH3 中的文件初始化后，与串行调试单元模块（SDU_DM）整合，下载至 FPGA 中测试即可。

4 逻辑设计 / 核心代码

4.1 设计 CPU 的数据通路（具备简单的 I/O 功能）和控制器

- 寄存器堆，增加一个用于调试的读端口
- 指令存储器、数据存储器，增加一个用于调试和初始化的读/写端口
- ALU 运算器，可以通过修改 Lab1 中的 ALU 得到
- 立即数拓展器，依据不同的指令类型将立即数拓展为 32 位
- npc 选择器
- 寄存器输入端选择器
- 控制信号生成器
- 跳转信号控制器、ALU 信号控制器

这些部件的设计是依照课本以及课件上给出的单周期 CPU 数据通路进行设计的，同时为了兼容 SDU 和实现 MMIO 进行了些许更改。

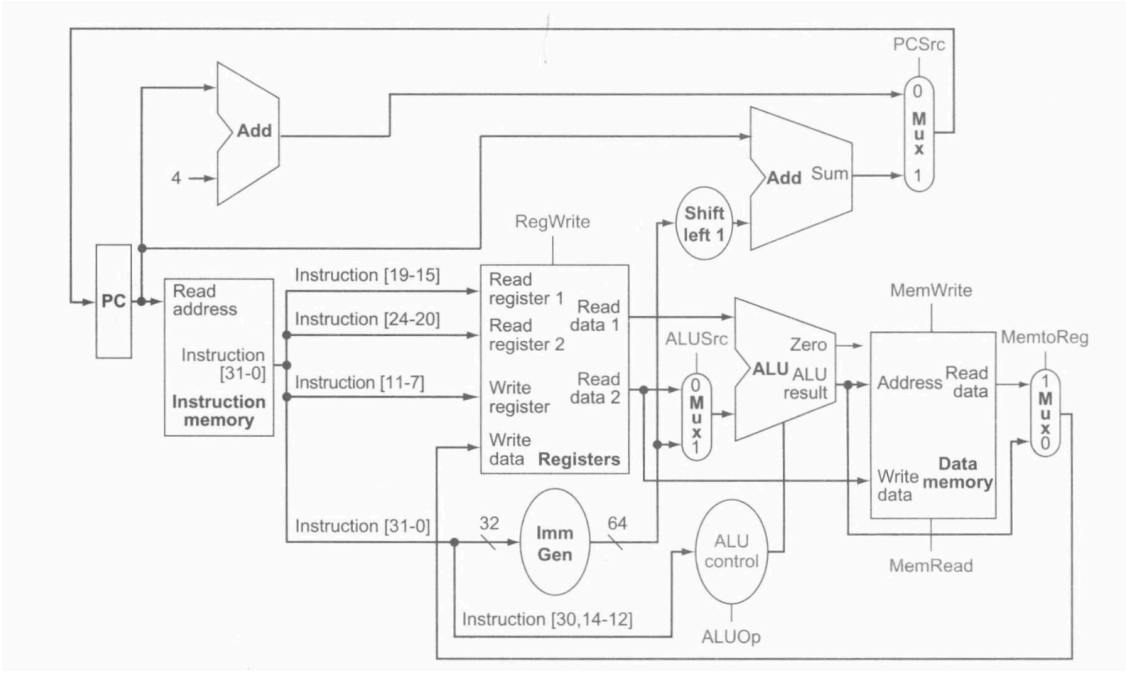

图 1: CPU-fig

需要注意的是，在 npc 选择器处对一些计算以及加法器进行了整合；在控制信号生成器中将 ALU 信号与跳转信号控制器单独做成了模块方便编写代码，因此控制信号和实际的数据通路会也有所不同。

具体来说，这些模块的接口如下，模块内的具体实现请参见原始代码：

```
module  reg_file(
    input  clk,    //时钟
    input [4:0]  ra0, ra1, ra2,  //读地址
    output [31:0] rd0, rd1, rd2,  //读数据
    input [4:0]  wa,    //写地址
    input [31:0] wd,   //写数据
    input we,    //写使能
);
data_mem_inst(.a(), .d(), .clk(), .we(), .spo()); //这是ip核，对CPU和SDU都要实现读写，因此选择单端口，使用控制信号决定信号来自何处
inst_mem_inst(.a(), .d(), .dpra(), .clk(), .we(), .spo(), .dpo()); //这是ip核，对CPU只需读，对SDU需要实现读写，因此选择双端口，分别接入CPU和SDU的信号
module alu (
    input [31:0] a, b,      //两操作数
    input [2:0] f,            //功能选择
    output reg [31:0] y,     //运算结果
    output reg [2:0] t            //比较标志
);
module imm_gen (
    input [31:0] raw,
    input [6:0] opcode,
    input [2:0] funct3,
    output reg [31:0] ext_imm
);
module next_pc_sel(
    input [2:0] pc_sel,
    input [31:0] alu_out,
    input [31:0] pc,
    input [31:0] ext_imm,
    output reg [31:0] npc
);
module reg_file_sel(
    input [2:0] reg_sel,
    input [31:0] alu_out,
    input [31:0] pc,
    input [31:0] dm_rd,
    input [31:0] ext_imm,
    output reg [31:0] rf_wd
);
module control(
    input [6:0] opcode,
    input [2:0] cc,
    input [2:0] funct3,
    input [6:0] funct7,
    output branch,
    output reg [2:0] reg_sel,
    output reg [2:0] pc_sel,
    output reg mem_write,
    output reg alu_src,
    output [2:0] alu_op,
    output reg reg_write
);
module branch_control(
    input [6:0] opcode,
    input [2:0] cc,
    input [2:0] funct3,
    output reg branch
);
```

```
57    module alu_control(
58        input [6:0] opcode,
59        input [2:0] funct3,
60        input [6:0] funct7,
61        output reg [2:0] alu_op
62    );
63
```

cpu.v

有关 MMIO 的实现，主要在检测到数据存储器的地址输入信号为 MMIO 地址时，设置 mmio_flag，则此时数据存储器的写信号切换为零，同时在外部处理对 mmio 寄存器的读写。具体代码如下：

```
1     always@(*) begin
2         if ((dm_addr == 16'h7f00) || (dm_addr == 16'h7f20)) begin
3             mmio_flag = 1;
4         end
5         else begin
6             mmio_flag = 0;
7         end
8     end
9     ...
10    assign dm_we = ~flag ? (mmio_flag ? 0 : mem_write) : _we_dm;
11    ...
12    if(mem_write) begin
13        if (dm_addr == 16'h7f00) begin
14            led <= dm_d;
15        end
16        if (dm_addr == 16'h7f20) begin
17            clock <= dm_d;
18        end
19    end
```

cpu.v

对于 NPC 选择器以及寄存器输入端选择器，总共有如下五种选择，只需要依照对应的参数切换即可。

```
1     parameter ALU_OUT = 3'b000;
2     parameter MEM_OUT = 3'b001;
3     parameter NEXT_PC = 3'b010; //PC + 4
4     parameter IMM_PC = 3'b011;  //PC + IMM
5     parameter IMM_ONLY = 3'b100; //IMM
6
```

control.v

另外由于需要实现 SDU 对数据存储器与指令存储器的读写，这个 cpu 中会根据 SDU 输入的 debug 信号来切换读写信号的来源，如下：

```
1     wire flag;
2     assign flag = debug;
3     assign dm_addr = ~flag ? alu_out[11:2] : _w_addr;
4     assign dm_d = ~flag ? rf_rd2 : _din;
5     assign dm_we = ~flag ? (mmio_flag ? 0 : mem_write) : _we_dm;
6     assign dm_clk = ~flag ? clk : _clk_ld;
7     data_mem data_mem_inst(.a(dm_addr), .d(dm_d), .clk(dm_clk), .we(dm_we), .spo(dm_rd));
8
```

reg.v

最后，SDU 调试时还要输出各类控制信号以及数据通路信息，需要将这些信号一一接给输出端，较为繁琐且没有难点，在这里不展示核心代码。

4.2 将 CPU 和 SDU 整合后下载至 FPGA，进行逐条指令功能测试与进行排序程序测试

这里需要小幅修改 LabH3 中得到的测试程序。例如添加对 MMIO 的读写，用 led 来显示测试成功/失败。

烧录后测试程序以及排序程序的正确性已经由助教检查完成，单周期 CPU 设计任务完成。

4.2.1 结果分析

RTL 电路：

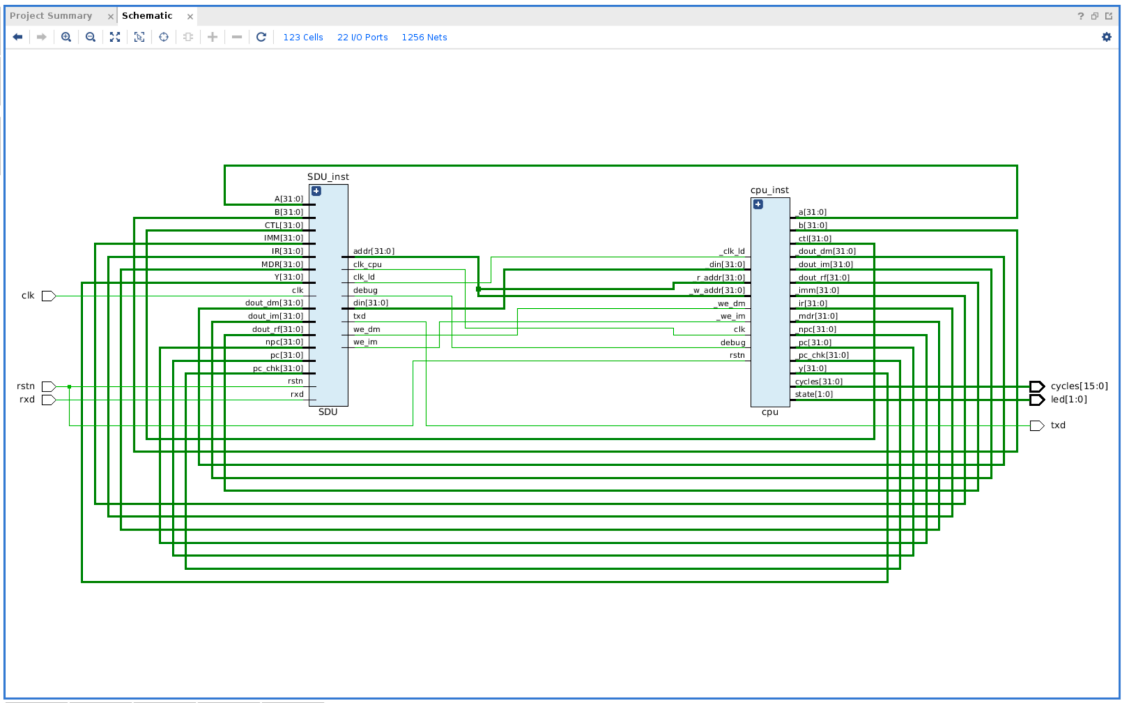

图 2: RTL

电路资源使用情况：

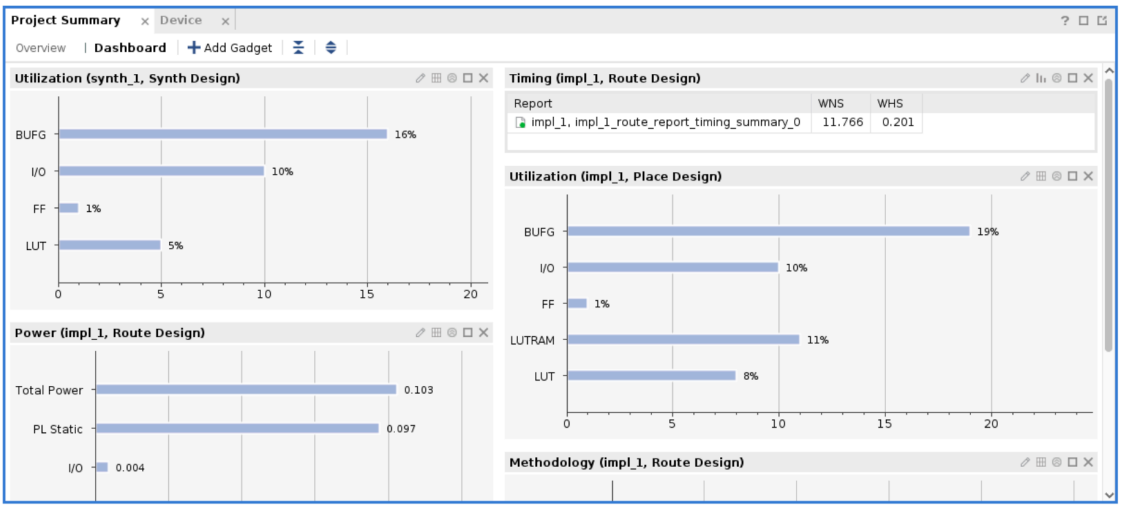

图 3: UTL1

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
top	5481	1825	1654	18	22	5
cpu_inst (cpu)	4154	1241	1587	14	0	0
control_isnt (c	452	7	27	0	0	0
data_mem_inst	1160	64	576	0	0	0
inst_mem_inst	1160	64	576	0	0	0
npc_sel_inst (r	48	32	24	0	0	0
rf_inst (reg_file	1196	992	384	14	0	0
rf_sel_inst (reg	5	32	0	0	0	0
SDU_inst (SDU)	1327	584	67	4	0	0

图 4: UTL2

电路性能:

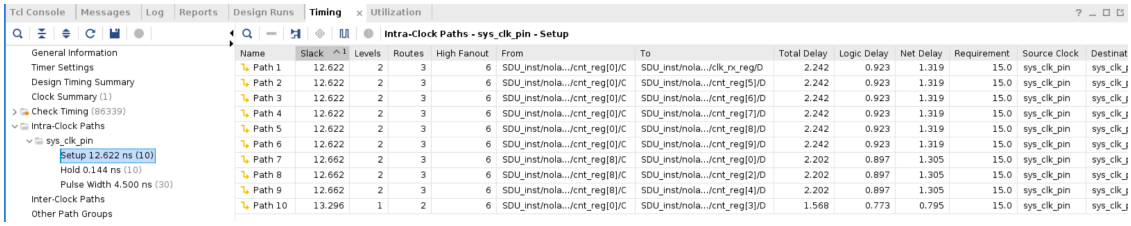

图 5: TIMING

5 实验总结

1. 通过本次实验，我学习到了如何依据需求设计 CPU 的数据通路，如何划分模块以及设计控制信号以完成需求。

2. 同时也学习了有关 MMIO 的知识，通过 MMIO 可以轻松地控制外设辅助显示必要信息。

3. 本次实验思维难度不大，主要难点在于模块过多过于复杂，需要提前想好如何接各个模块之间的信号以及整体设计，否则编写代码会十分困难。

6 意见/建议

无，这个实验设计很完美。