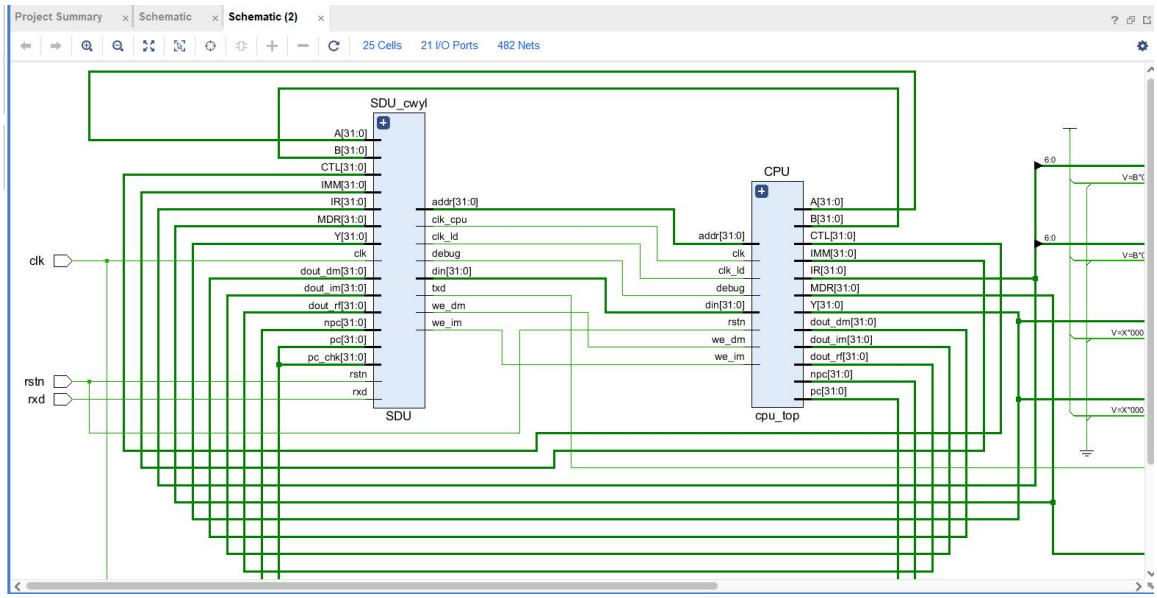
Lab 4 单周期 CPU 设计 report

PB21111681 朱炜荣

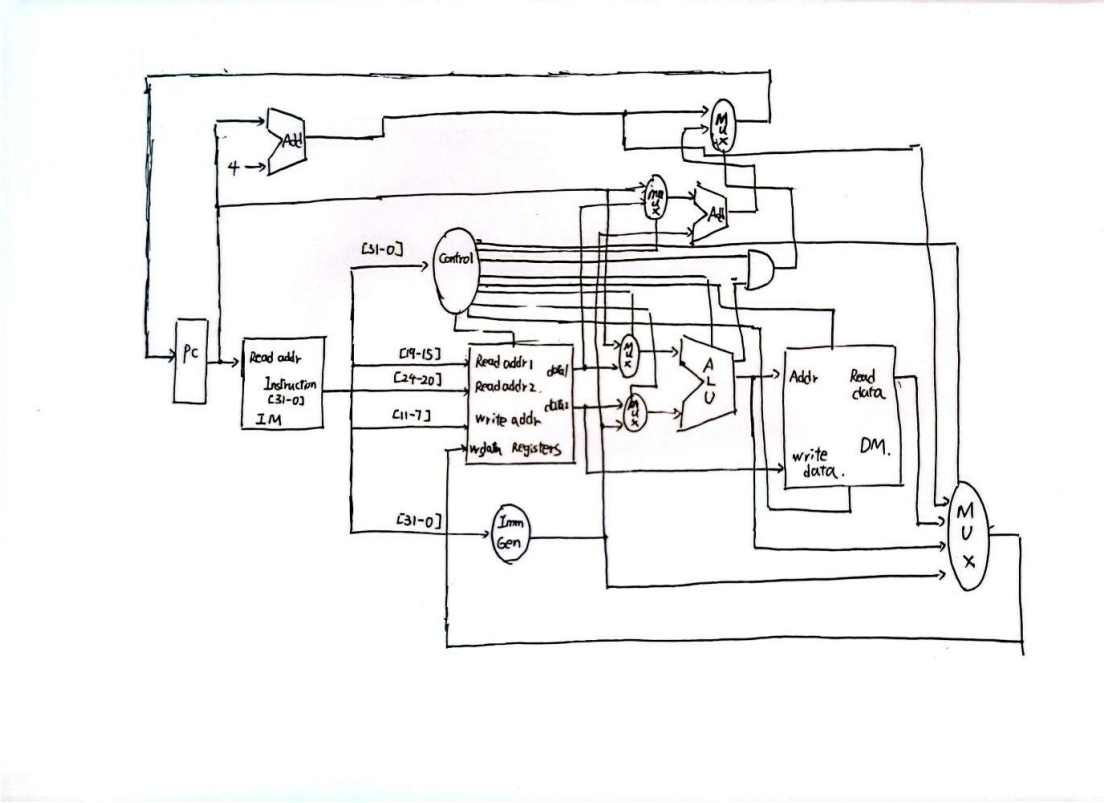
实验目的与内容

1. 熟悉并理解单周期 CPU 的工作流程
2. 进一步理解加深基础的 RISC-V 指令
3. 加深理解数据通路在 Verilog 代码的地位
4. 借助提供的串口学会检验指令和调试单周期 CPU 的能力

逻辑设计



数据通路如下所示：



我觉得单周期 CPU 中最重要的代码是控制信号的生成以及各个模块之间的数据通路，由于篇幅原因在这里只展示数据通路部分，每个小模块的用途已经用注释的方法标注在模块的前一行。

```
//PC_Mux1
Mux  pc_mux1(
        .choice(PCSrc),
        .in0(PC),
        .in1(rf_data1&32'hffff_fffe),
        .out(PC_c)
);
//PC 自增
ALU  pc_adder1(
        .a(PC),
        .b(32'h4),
        .f(3'b001),
        .y(ori_npc),
        .t(t0)
);
//跳转与分支
ALU  pc_adder2(
        .a(PC_c),
        .b(immediate),
```

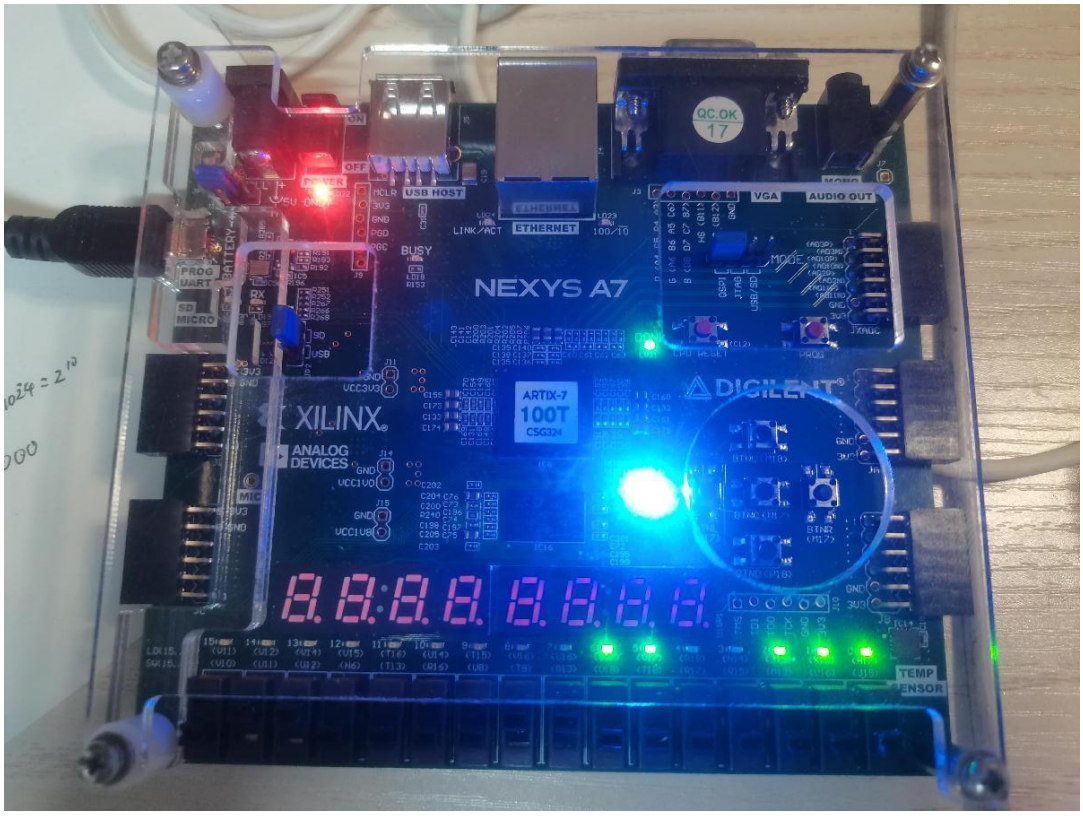
```
        .f(3'b001),
        .y(new_npc),
        .t(t1)
);
//PC_Mux2
Mux  pc_mux2(
        .choice(Branch  &  zero),
        .in0(ori_npc),
        .in1(new_npc),
        .out(NPC)
);
//ImmGen
ImmGen  immgen(
        .instruction(Instruction),
        .imm(immediate)
);
//指令存储器
text_memory  IM(
        .a(addr),
        .d(din),
        .dpra(PC[11:2]),
        .clk(clk_work),
        .we(we_im),
        .spo(dout_im),
        .dpo(Instruction)
);
//数据存储器
data_memory  DM(
        .a(dm_add1_work),
        .d(rf_data2),
        .dpra((MemRead==1)?((alu_out[13:0]  >=  14'h2200  ||  alu_out[13:0]  <14'h2000)?10'b0:((alu_out[13:0]  -  14'h2000)  >>  2)):10'b0),
        .clk(clk_work),
        .we(we_dm_work),
        .spo(dout_dm),
        .dpo(dm_out)
);
//寄存器堆
register_file  rf(
        .clk(clk_work),
        .ra1(rf_add1_work),
        .ra2(Instruction[24:20]),
        .rd1(rf_data1),
        .rd2(rf_data2),
        .wa(Instruction[11:7]),
```

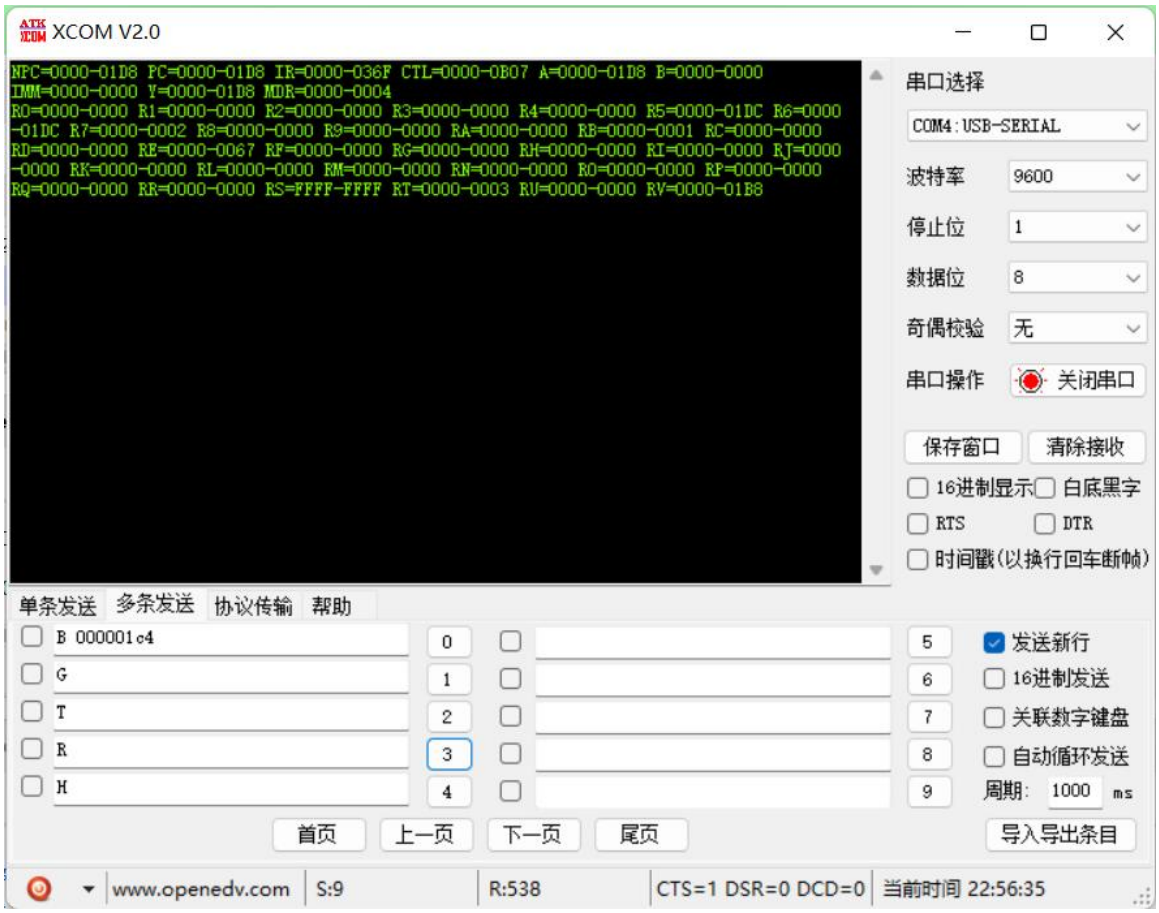
```
        .wd(data_wb),
        .we(RegWrite)
);
//控制信号
Control  ctrl(
        .instruction(Instruction),
        .signal({Branch,MemRead,ToReg[1:0],ALUOp[2:0],MemWrite,PCSrc,ALUSrc1,ALUSrc2,RegWrite})
);
//ALU_Mux1
Mux  alu_mux1(
        .choice(ALUSrc1),
        .in0(rf_data1),
        .in1(PC),
        .out(alu_data1)
);
//ALU_Mux2
Mux  alu_mux2(
        .choice(ALUSrc2),
        .in0(rf_data2),
        .in1(immediate),
        .out(alu_data2)
);
//ALU
ALU  alu(
        .a(alu_data1),
        .b(alu_data2),
        .f(ALUOp),
        .y(alu_out),
        .t(index_zero)
);
//Data_Mux
//  Mux  data_mux(
//          .choice(MemtoReg),
//          .in0(alu_out),
//          .in1(dm_out),
//          .out(data_wb)
//  );
always@(*)
begin
        case(ToReg)
        2'b00:    data_wb  =  alu_out;
        2'b01:    data_wb  =  (alu_out  ==  32'h0000_3f20  &&  Instruction[6:0]  ==  7'b0000011)?da
    ta_mmio:dm_out;
        2'b10:    data_wb  =  immediate;
        2'b11:    data_wb  =  ori_npc;
```

```
108.            endcase
109.  end
```

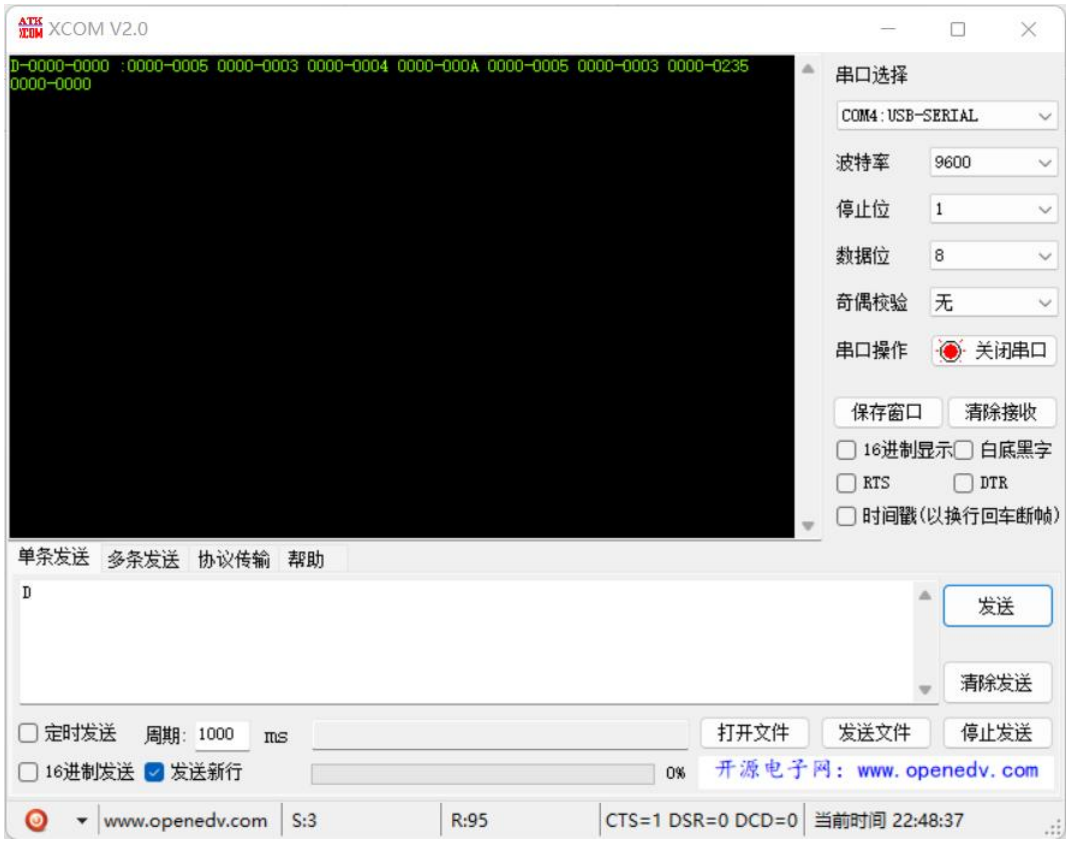
其中//Data_Mux 的修改是因为实验文档中的数据通路无法处理如 lui、auipc、jal 等 PC 与寄存器数据之间的交互，所以进行了进一步修改。

测试结果与分析

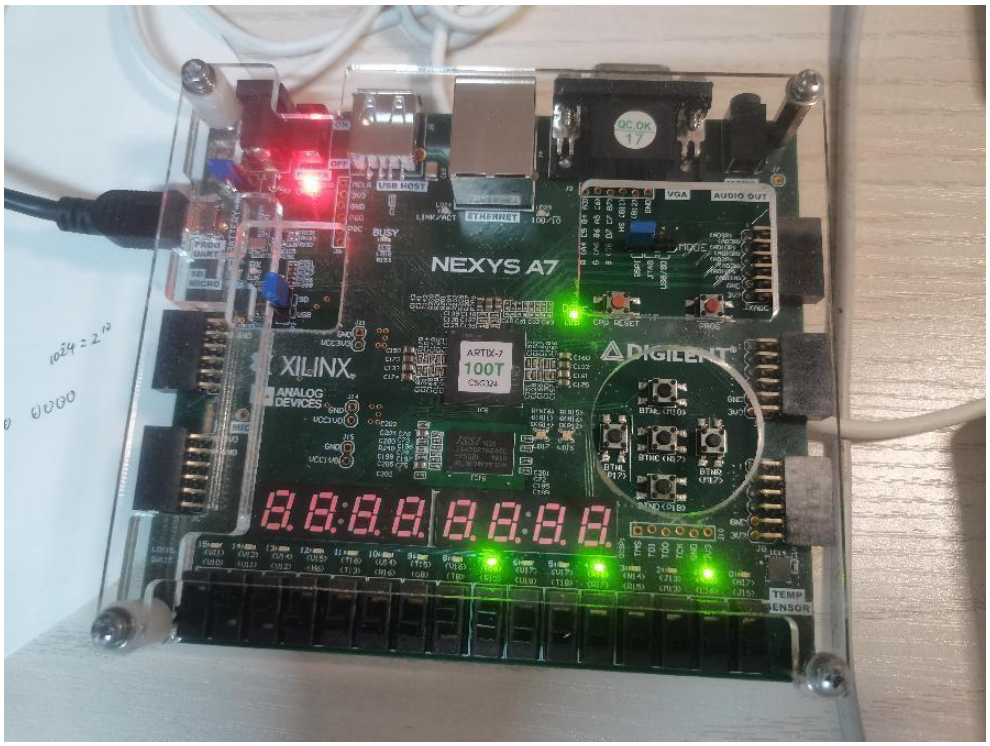


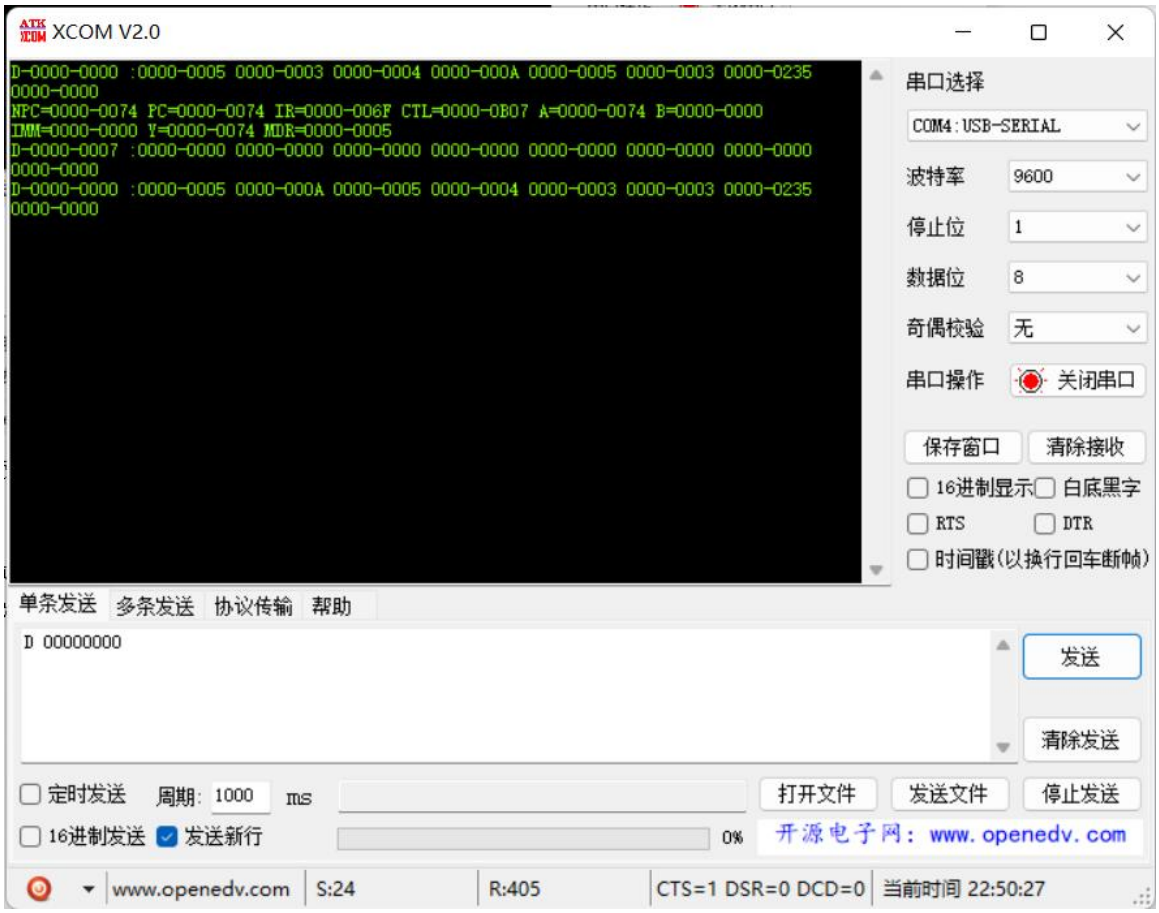


首先是测试指令集的 asm 汇编码，将 Lab3 中编写测试的汇编码生成对应的 coe 文件并分别用来例化我们的分布式存储器的 IP 核，然后利用提供的 PC 串口来进行调试，其中对于 RB 我们存储了的是是否成功结束的信号，RE 存储了 CPU 一共有效运行了多少个周期，R6 存储了结束标志 PC+4 的数值，进行检验之后均相同。



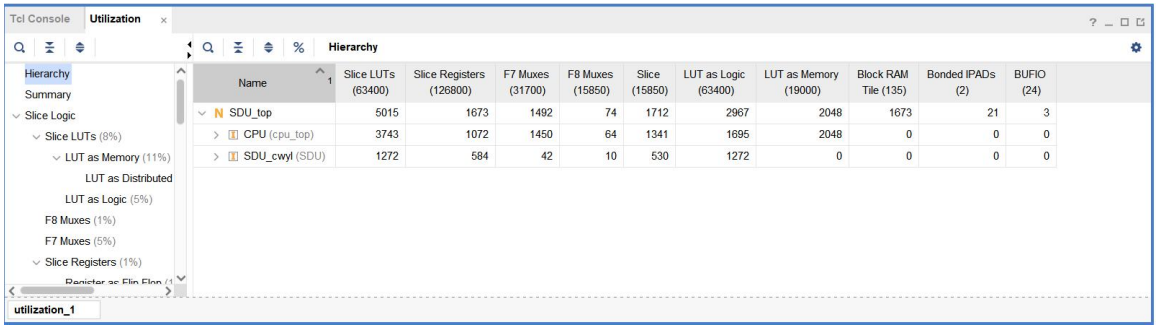
然后是排序的程序，在排序之前我们先通过串口来查看数据存储器的排列情况，第一个数为我们要排多少个数，然后紧接着的是我们即将被排序的数组。



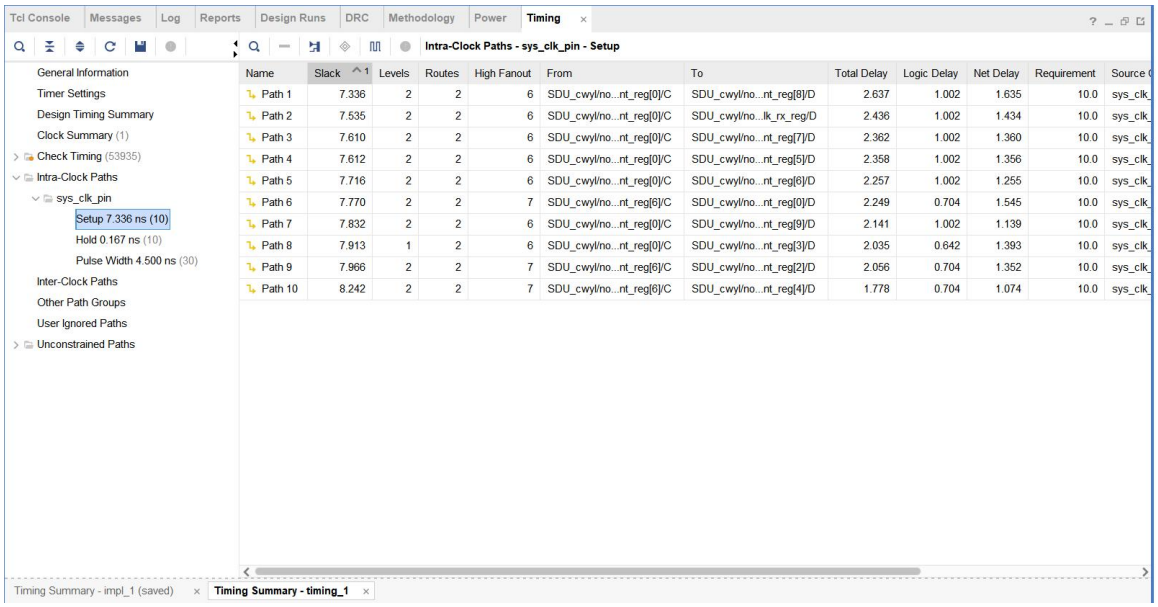


在排序完成之后，我们利用串口查看排序的结果。我们发现在进行 5 个数的排序之后，数组变为了 10 5 4 3 3 235，其中 235 已经超出了 5 个数的限制，所以并没有被排到最前面。

电路资源的使用情况：



综合电路性能：



总结

在本次实验中，我加深理解了数据通路在 Verilog 中的作用，同时也进一步学习巩固了 Verilog 的语法和规范书写方法。学习了仿真程序在没有串口调试的情况下如何来检查程序中的可能错误。同时接触到了 MMIO 与 load 和 store 的实现逻辑和具体定义。在检查的过程中，我从助教身上学习到了如何写出测试指令集完备性的周密思考。最后加深理解了单周期 CPU 的工作流程，对于单周期 CPU 的控制信号和有关各个指令 CPU 各个部件的输出有了更为深刻的理解。

吐槽：老师的实验文档总给我一种没看懂的感觉，而且中间的有些错误看起来不是很严重，但是对于云里雾里的我来说就是致命的误导。老师能否将单周期 CPU 文档中有关 mmio 部分的解释再细致一些。