Lab 2 寄存器堆与存储器及其应用报告

PB21111681 朱炜荣

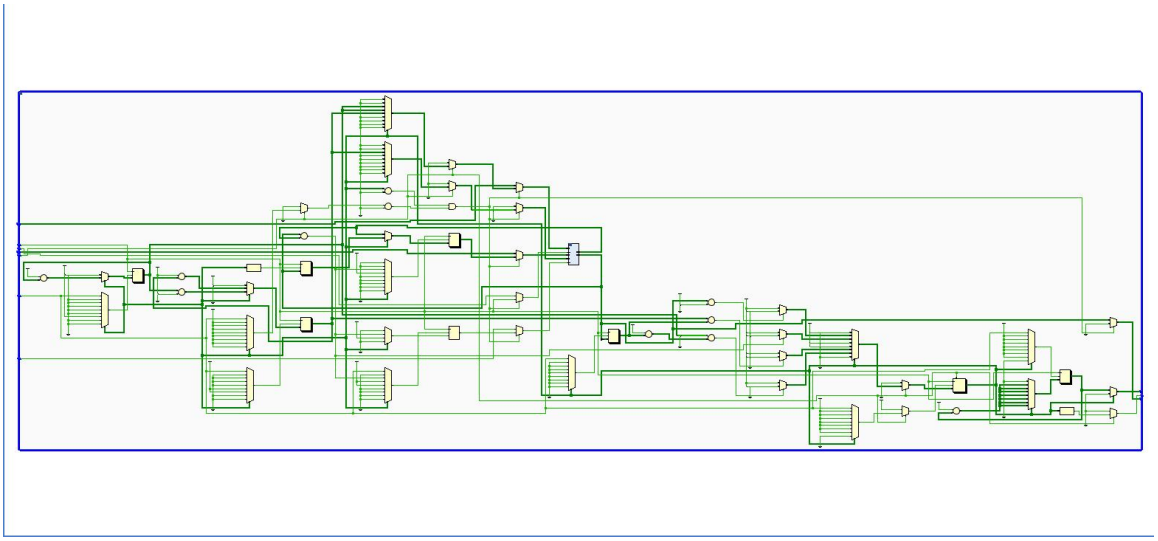
实验目的与内容

本次实验的主要目的为帮助回忆有关分布式存储器、IP 核使用和例化的有关知识，同时掌握寄存器堆和存储器的功能、时序及其应用。同时进一步熟练掌握数据通路和控制器的设计与编写实现。

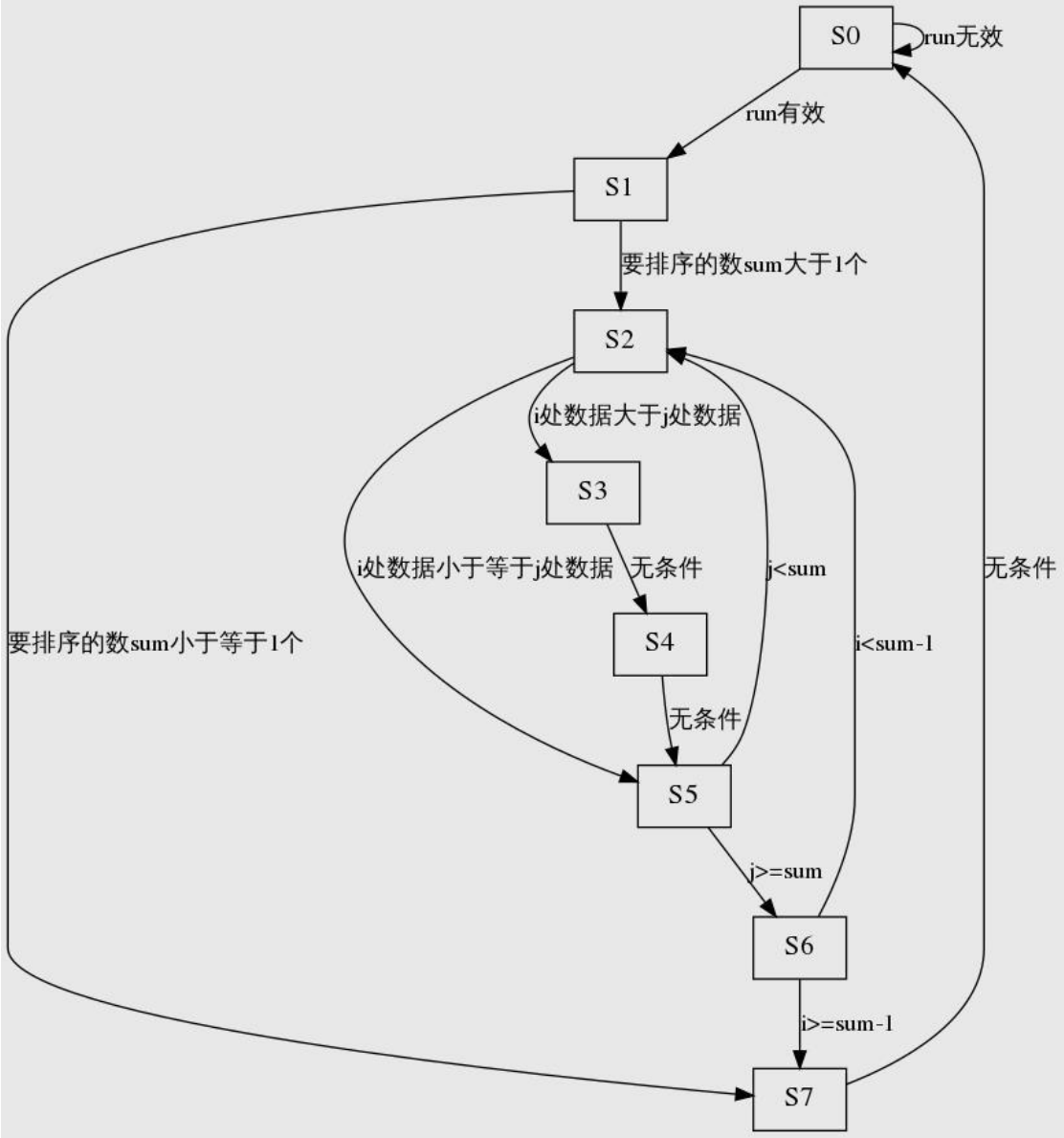
同时实验提供了一个串口调试的程序和 SDU 代码，希望大家通过这次实验可以简单地了解有关串口调试单元和调试指令的有关知识。

逻辑设计

数据通路以 RTL 电路代替：



程序的状态转换图如下：



其中较为核心的代码为，状态机的状态转移与冒泡排序之间的关联，代码如下：

```
always@(*) begin
    if(!rstn) begin
        addr1_s = 5'b0;
        addr2_s = 5'b0;
        next_state = S0;
        flag = 0;
        done = 0;
    end
    else begin
        next_state = current_state;
    case (current_state)
```

```
12.         S0:begin
13.             done = 1 ;
14.             addr1_s = 5'b0;
15.             addr2_s = 5'b0;
16.             if(run) begin
17.                 next_state = S1;
18.                 flag = 1;
19.             end
20.             else begin
21.                 flag = 0;
22.             end
23.         end
24.         S1: begin
25.             done = 0;
26.             addr1_s = 5'b0;
27.             addr2_s = 5'b0;
28.             flag = 1;
29.             if(r_value1 <= 1)
30.                 next_state = S7;
31.             else
32.                 next_state = S2;
33.         end
34.         S2: begin
35.             done = 0;
36.             addr1_s = i;
37.             addr2_s = j;
38.             flag = 1;
39.             if(r_value1 > r_value2)
40.                 next_state = S3;
41.             else
42.                 next_state = S5;
43.         end
44.         S3:begin
45.             done = 0;
46.             addr1_s = i;
47.             addr2_s = 5'b0;
48.             next_state = S4;
49.             flag = 1;
50.         end
51.         S4:begin
52.             done = 0;
53.             addr1_s = j;
54.             addr2_s = 5'b0;
55.             next_state = S5;
56.             flag = 1;
57.         end
58.         S5:begin
59.             done = 0;
```

```
            flag = 1;
            addr1_s = 5'b0;
            addr2_s = 5'b0;
            if(j >= sum)
                next_state = S6;
            else
                next_state = S2;
        end
        S6: begin
            done = 0;
            flag = 1;
            addr1_s = 5'b0;
            addr2_s = 5'b0;
            if(i >= sum - 1)
                next_state = S7;
            else
                next_state = S2;
        end
        S7:begin
            done = 1;
            next_state = S0;
            addr1_s = 5'b0;
            addr2_s = 5'b0;
            flag = 1;
        end
        default:begin
            done = 0;
            flag = 1;
            addr1_s = 5'b0;
            addr2_s = 5'b0;
        end
    endcase
    end
end


always@(posedge clk) begin
    case (current_state)
        S0: begin
            if(run) begin
                count <= 16'b0;
                we_s <= 0;
                i <= 5'h1;
                j <= 5'h2;
                sum <= r_value1;
            end

        end
```

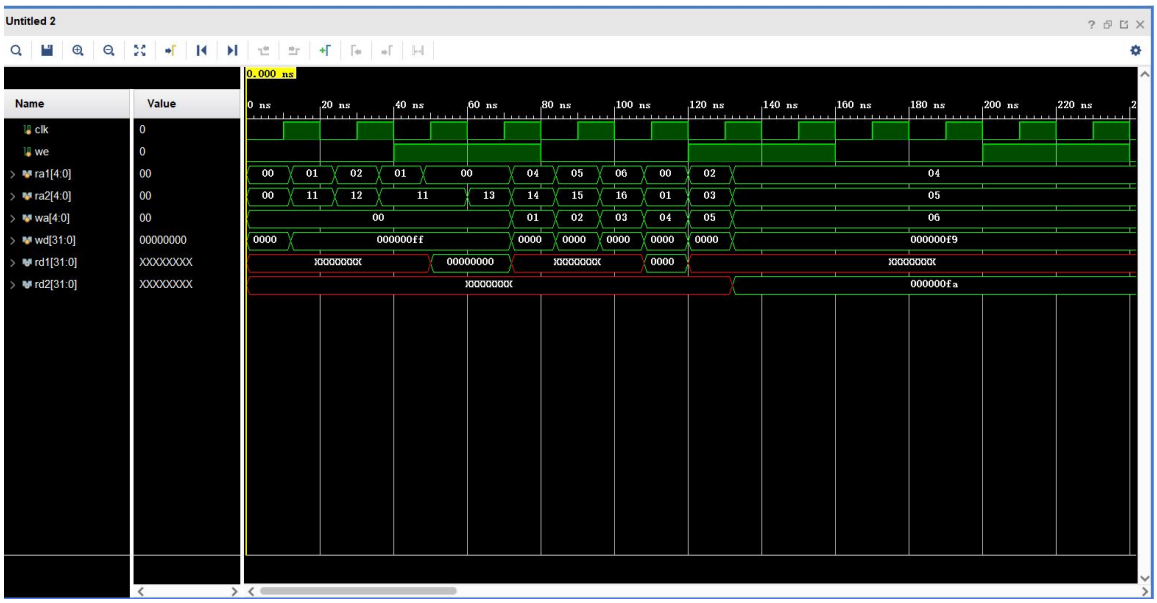
```
        S1: begin
            count <= count + 1;
        end
        S2: begin
            count <= count + 1;
            num <= r_value1;
//            num2 <= r_value2;
            if(r_value1 > r_value2) begin
                we_s <= 1;
                w_value_s <= r_value2;
            end
        end
        S3: begin
            count <= count + 1;
            we_s <= 1;
            w_value_s <= num;
        end
        S4: begin
            count <= count + 1;
            we_s <= 0;
        end
        S5: begin
            count <= count + 1;
            j <= j + 1;
        end
        S6: begin
            count <= count + 1;
            i <= i + 1;
            j <= i + 2;
        end
        S7: begin
            count <= count + 1;
        end
        default: begin
            count <= count;
        end
    endcase
end
```

其中 S0 状态是状态机的初始状态，这个时候程序会一致处在这个阶段并将读写等数据转交给 SDU 模块，而当 run 等于 1 时，程序将存储器读写权转交 SRT 模块。在 S1 状态，SRT 分别进行了 i,j 的初始化，并将要排序的数组长度赋值给 sum。然后依据排序长度来判断状态跳转，当长度小于等于 1 时，直接进入 S7 结束状态，若大于 1 则进入 S2 状态。S2 状态中程序将 i,j 的大小分别赋给 addr1 和 addr2，并在将读取的结果进行比较，通过比较大小的情况判断状态跳转情况。如果 i 处数值小于等于 j 处数值则直接跳到 S5 状态，相反地则会跳到 S3 状态。S3、S4 两个状态主要在进行数值修改的工作，将 i、j 两个地址的数值大小交换，无条

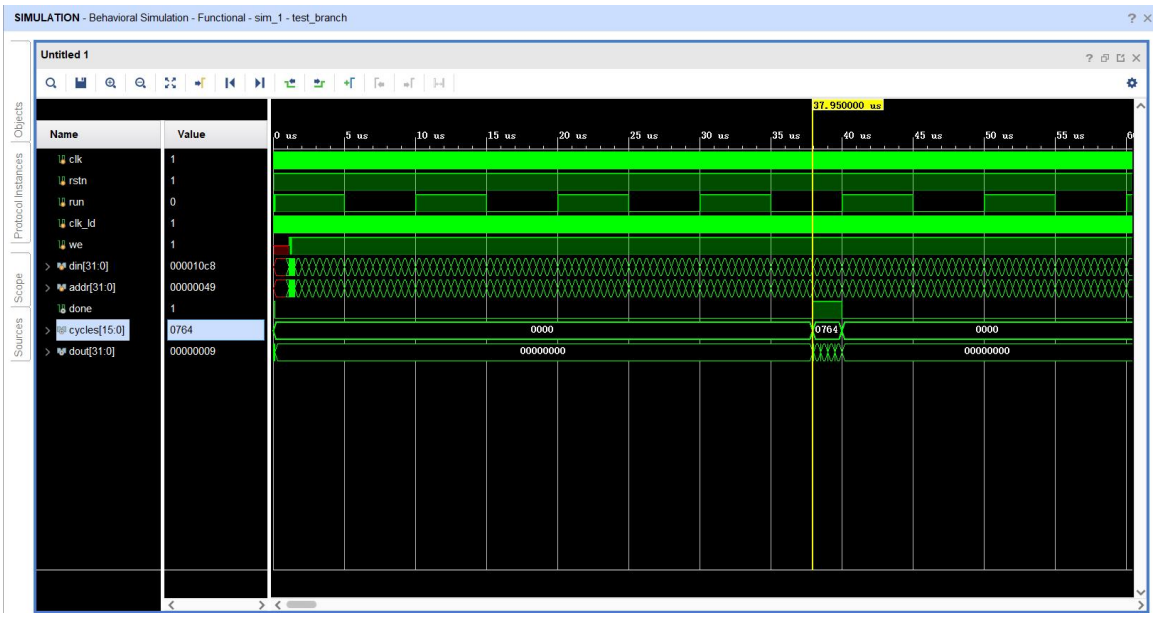
件跳转到 S5。S5 中主要进行了 j 的更新，如果 j>=sum 则跳至 S6，否则跳至 S2。S6 中进行了 i 的更新，如果 i>=sum-1 则跳至结束状态 S7，否则跳至 S2。S5 和 S6 两个状态的判断实现了冒泡排序的循环部分，S2 的大小比较实现了排序。

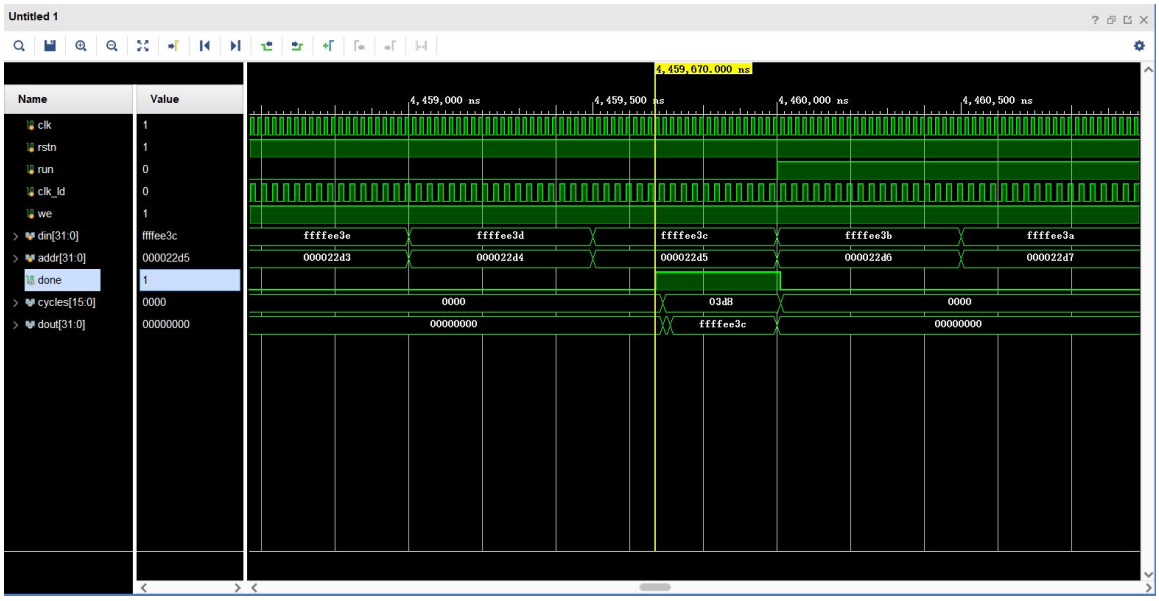
仿真结果与分析

Register 的仿真结果如下：



SRT 的仿真结果如下：





Register 的仿真文件为：

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
/
3.  // Company:
4.  // Engineer:
5.  //
6.  // Create Date: 2023/04/09 20:23:55
7.  // Design Name:
8.  // Module Name: test_branch
9.  // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
/
21.
22.
23. module test_branch();
24. reg clk,we;
25. reg [4:0] ra1,ra2,wa;
26. reg [31:0] wd;
27. wire [31:0] rd1,rd2;
```

```
register_file test(
    .clk(clk),
    .ra1(ra1),
    .ra2(ra2),
    .rd1(rd1),
    .rd2(rd2),
    .wa(wa),
    .wd(wd),
    .we(we)
);

always #10 clk = ~clk;
always #40 we = ~we;

initial begin
    clk = 0; we = 0; ra1 = 5'b0; ra2 = 5'b0; wa = 5'b0; wd = 32'b0;
    #12 ra1 = 5'b1; ra2 = 5'b10001; wa = 5'b0; wd = 32'hff;
    #12 ra1 = 5'b10; ra2 = 5'b10010; wa = 5'b0; wd = 32'hff;
    #12 ra1 = 5'b1; ra2 = 5'b10001; wa = 5'b0; wd = 32'hff;
    #12 ra1 = 5'b0; ra2 = 5'b10001; wa = 5'b0; wd = 32'hff;
    #12 ra1 = 5'b0; ra2 = 5'b10011; wa = 5'b0; wd = 32'hff;
    #12 ra1 = 5'b100; ra2 = 5'b10100; wa = 5'b1; wd = 32'hfe;
    #12 ra1 = 5'b101; ra2 = 5'b10101; wa = 5'b10; wd = 32'hfd;
    #12 ra1 = 5'b110; ra2 = 5'b10110; wa = 5'b11; wd = 32'hfc;
    #12 ra1 = 5'b0; ra2 = 5'b1; wa = 5'b100; wd = 32'hfb;
    #12 ra1 = 5'b10; ra2 = 5'b11; wa = 5'b101; wd = 32'hfa;
    #12 ra1 = 5'b100; ra2 = 5'b101; wa = 5'b110; wd = 32'hf9;
end


endmodule
```

仿真文件中故意修改了 0 号寄存器的数值，并在修改完之后读取它，我们发现读取得到的数值依旧为 0，所以实验要求的寄存器堆的 0 号寄存器内容恒定为零实现了，同时通过观察仿真文件，写优先的读取模式也被实现。

SRT 的仿真文件为：

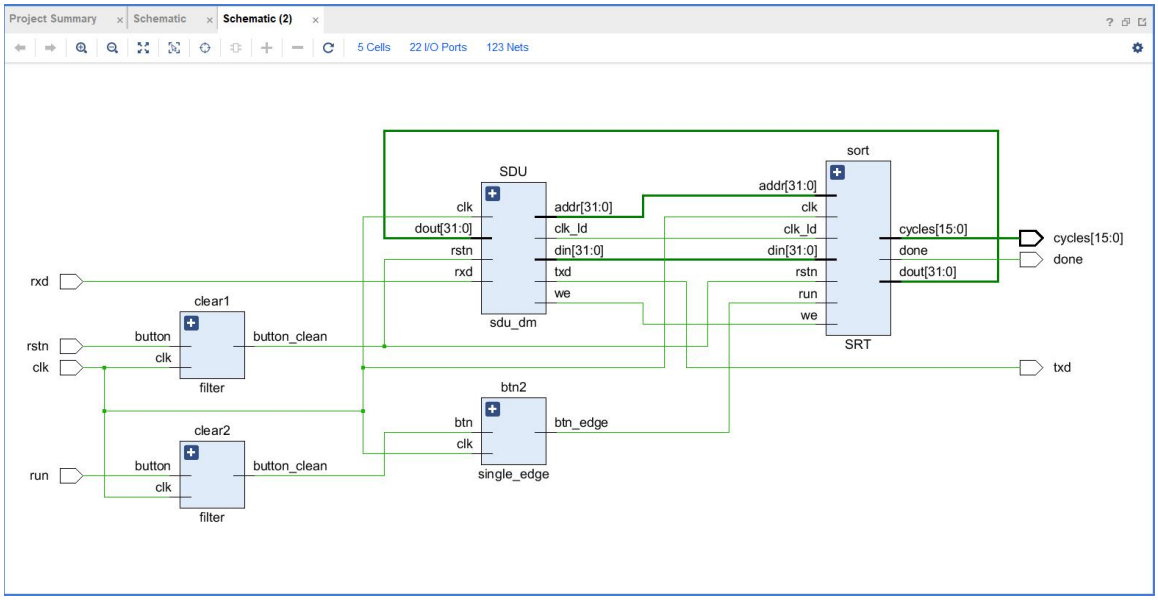
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2023/04/10 16:12:47
// Design Name:
```

```
// Module Name: test_branch
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/


module test_branch();
reg clk = 0,rstn,run,clk_ld = 1,we;
reg [31:0] din,addr;
wire done;
wire [15:0] cycles;
wire [31:0] dout;

SRT sort(
    .clk(clk),
    .rstn(rstn),
    .run(run),
    .done(done),
    .cycles(cycles),
    .addr(addr),
    .dout(dout),
    .din(din),
    .we(we),
    .clk_ld(clk_ld)
);

always #10 clk = ~clk;
always #15 clk_ld = ~clk_ld;
always #5000 run = ~run;
always begin #500 addr = addr + 1; din = din - 1; end
initial begin
    rstn = 1; run = 0;
    #100 run = 1;
    #1000 din = 32'hffff;addr = 32'h000f;we = 1;
    #40 din = 32'h1111;addr = 32'h0000;we = 1;
    #40 din = 32'haaaa;addr = 32'h0008;we = 1;
    #40 din = 32'h1111;addr = 32'h0000;we = 0;
    #40 we = 1;
```

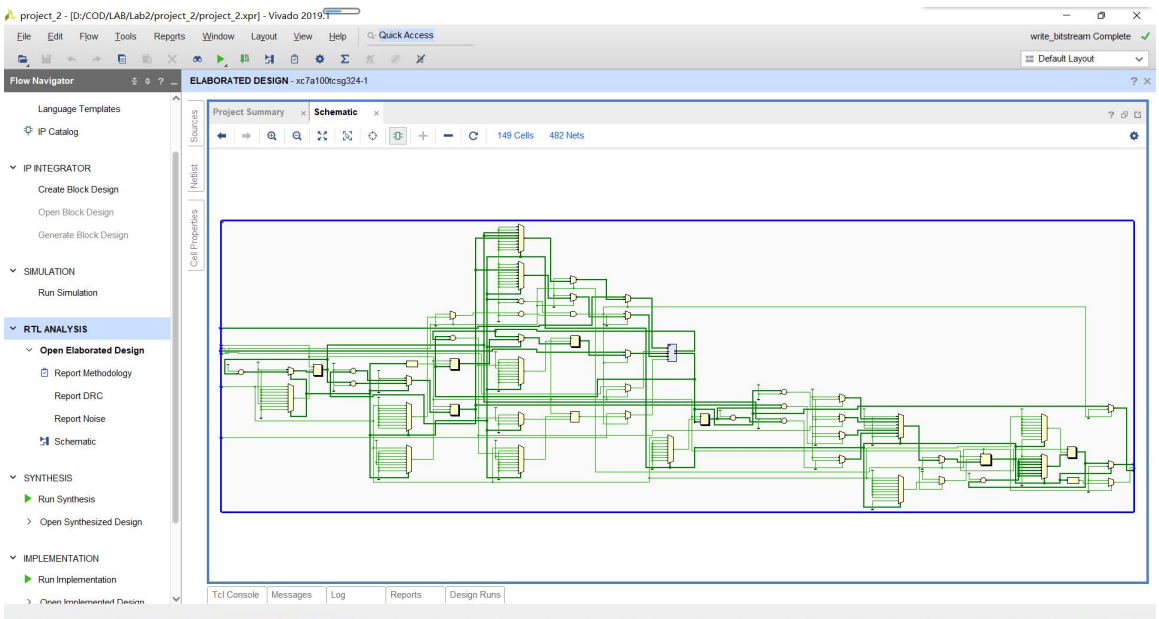
```
55.  end
56.
57.  endmodule
```

其中我们先对例化的数据进行排序，并得到一个 cycles 数值，之后我们再利用 SDU 传导的数值改写数组中的数值，然后再进行排序查看 cycles 是否有所改变，同时在最后我们保持所有的数据不变，再查看 cycles 的数值是否是一致的。

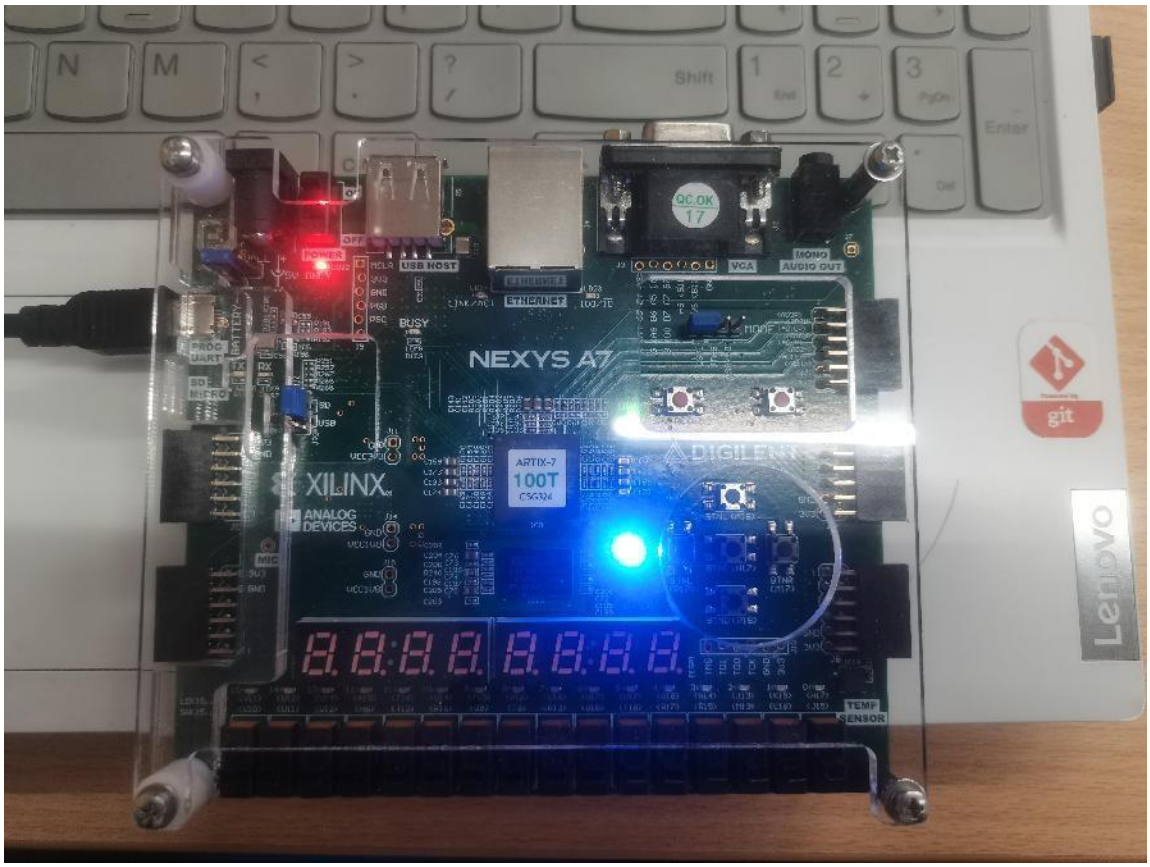
电路设计与分析

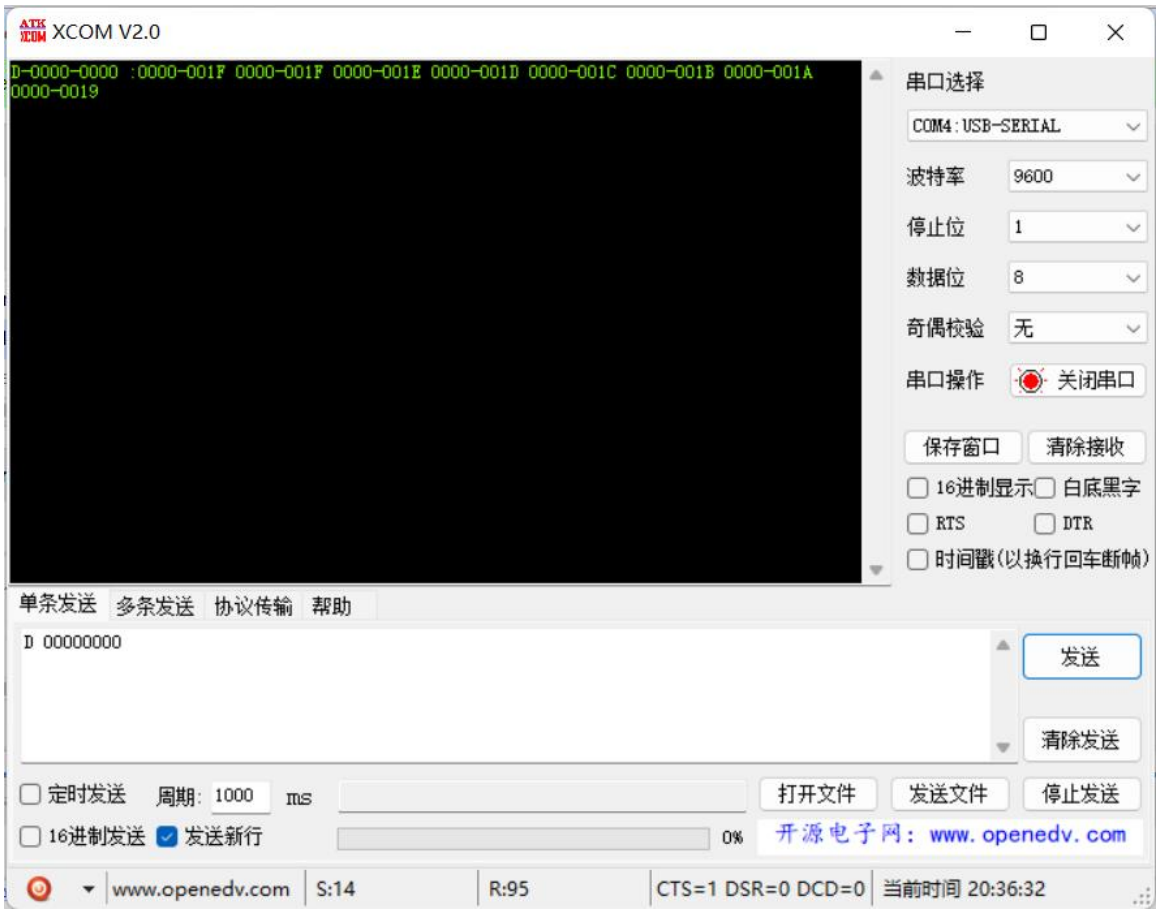


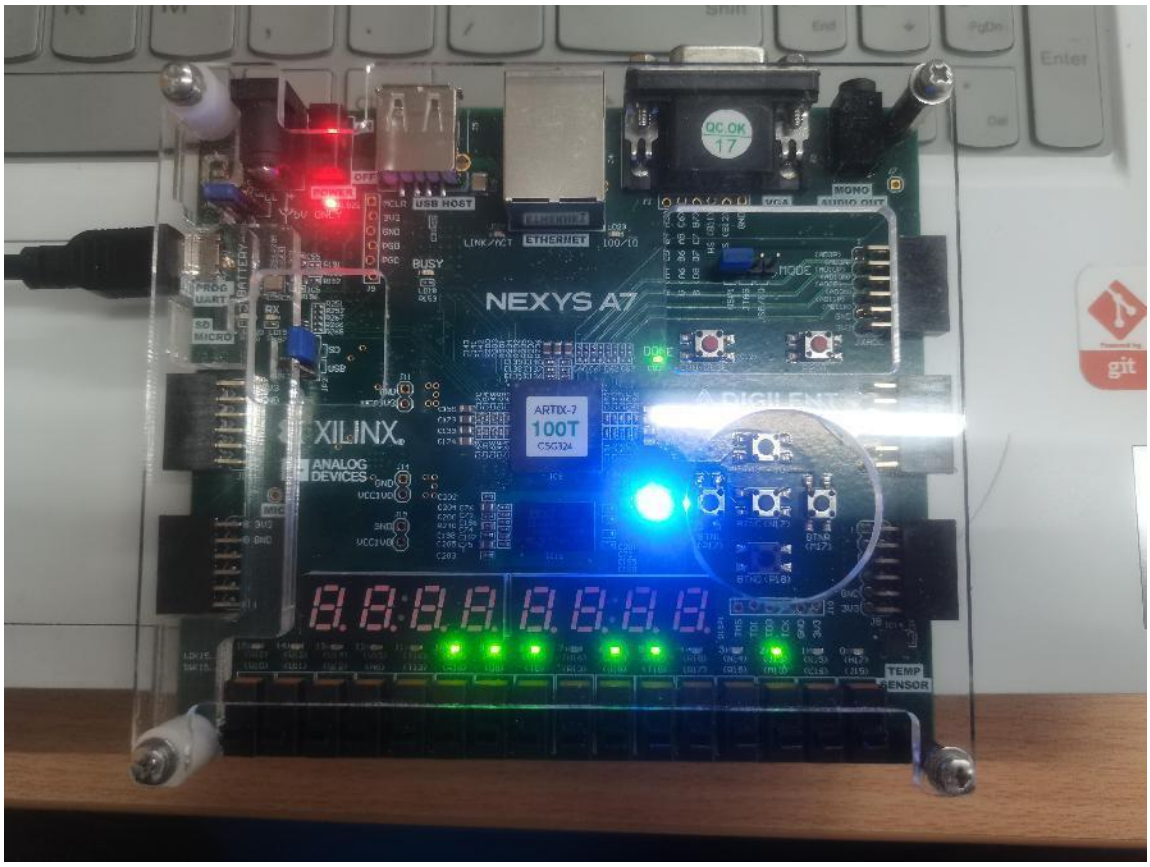
以下为 SRT 模块的 RTL 电路图：

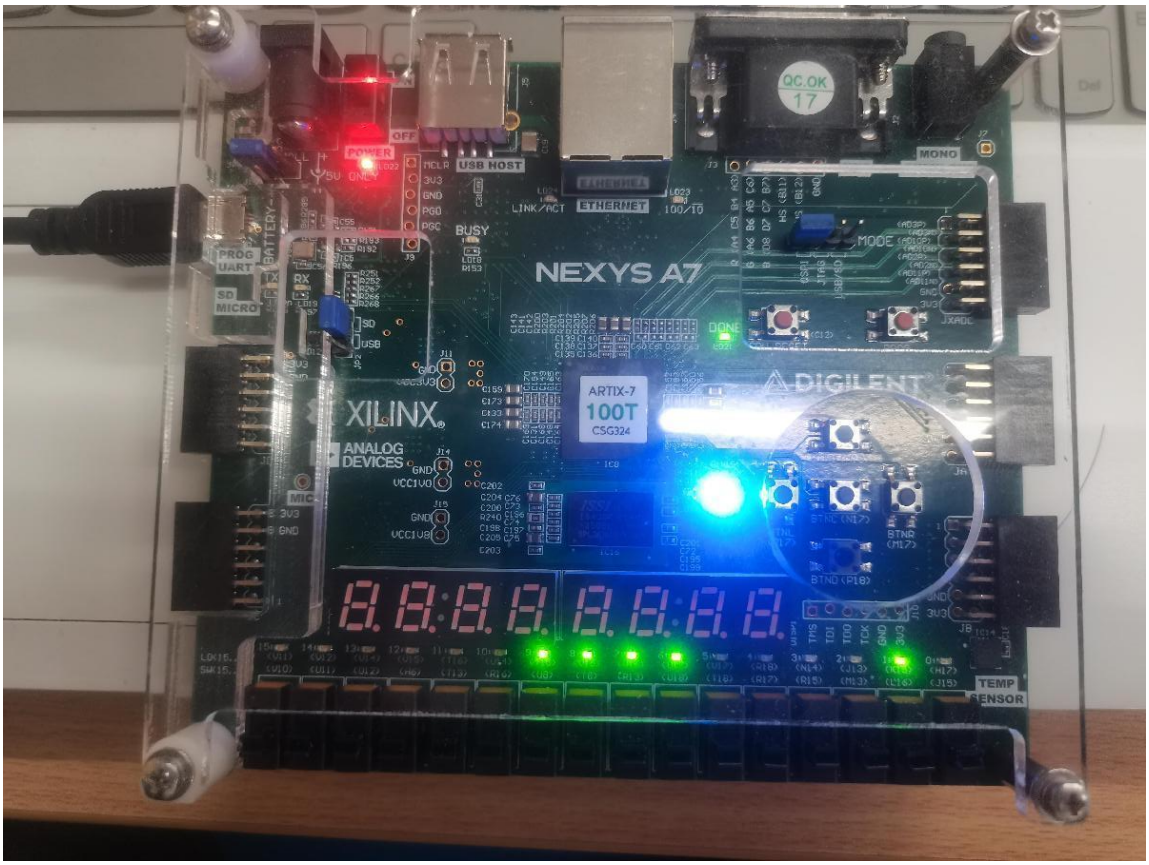


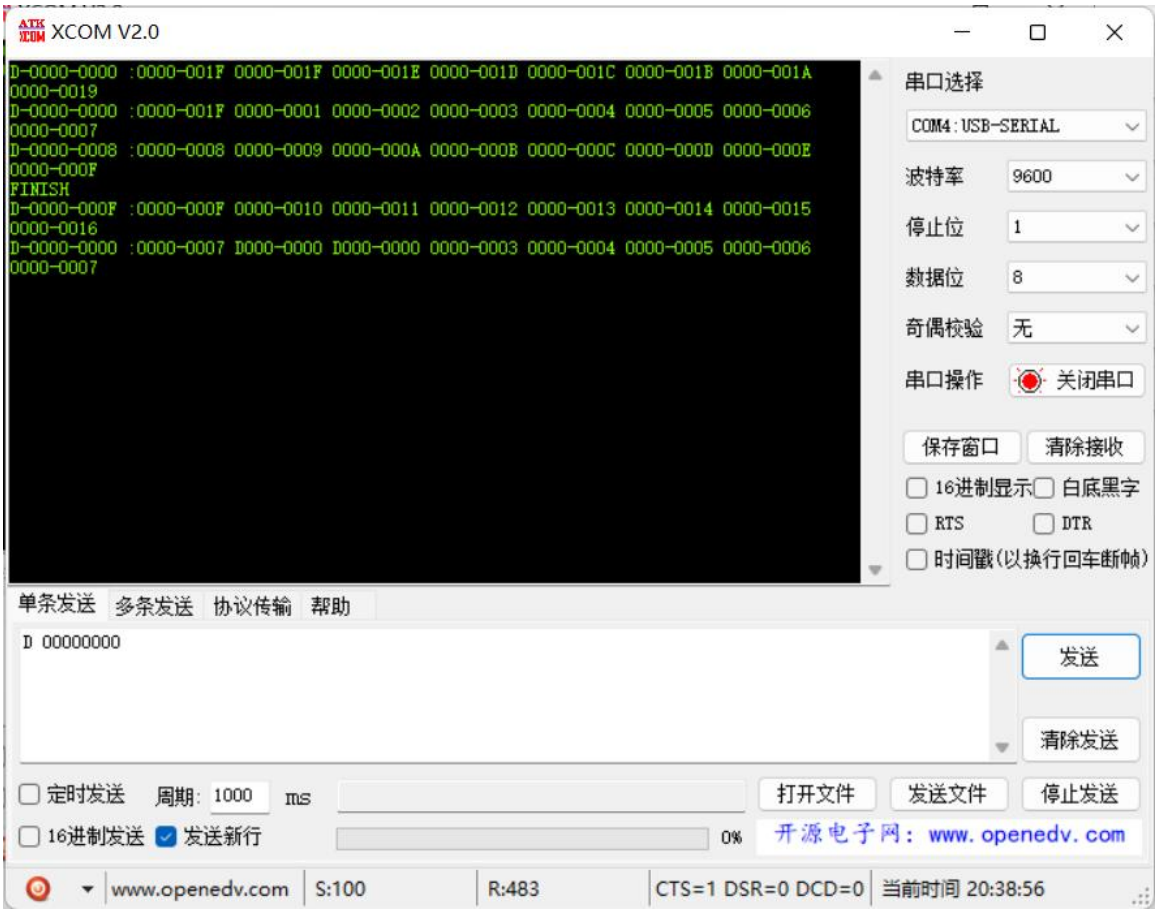
测试结果与分析

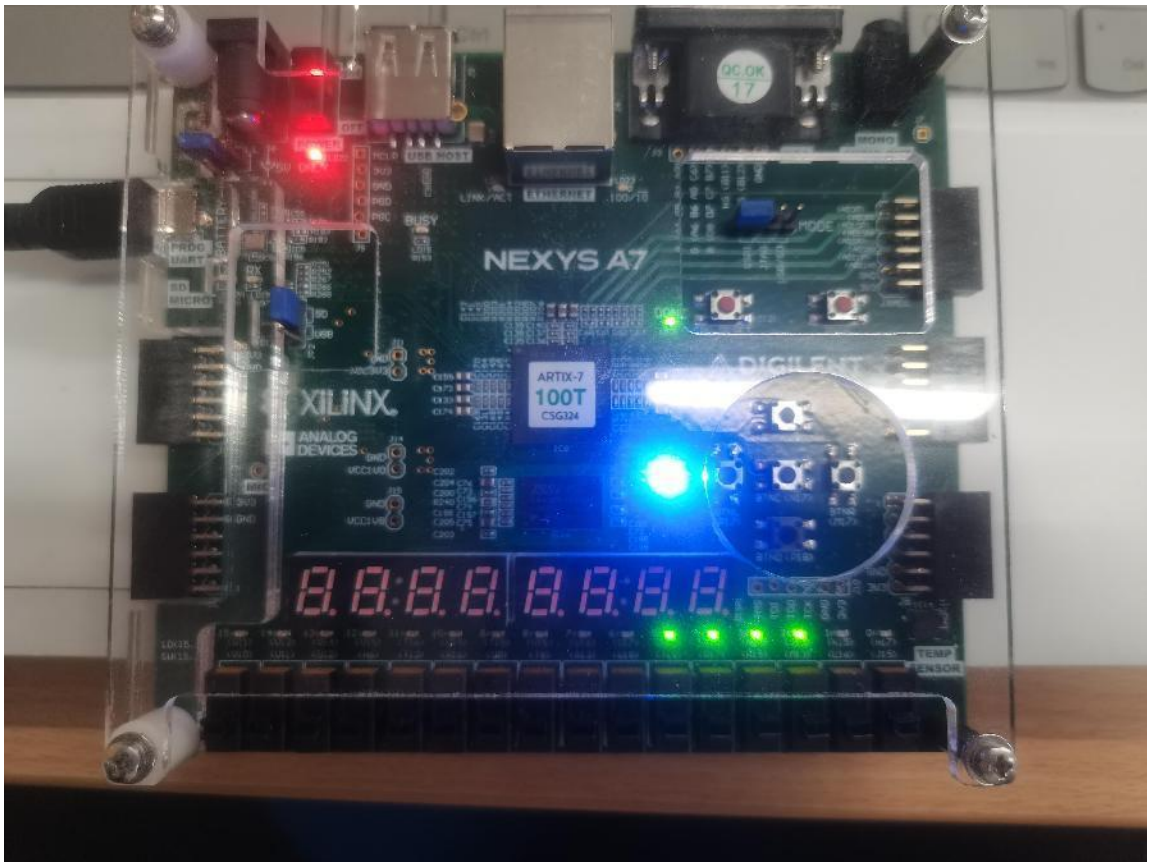

NEXYS A7
XILINX
DIGILENT
ANALOG DEVICES
ARTIX-7
100T
Lenovo

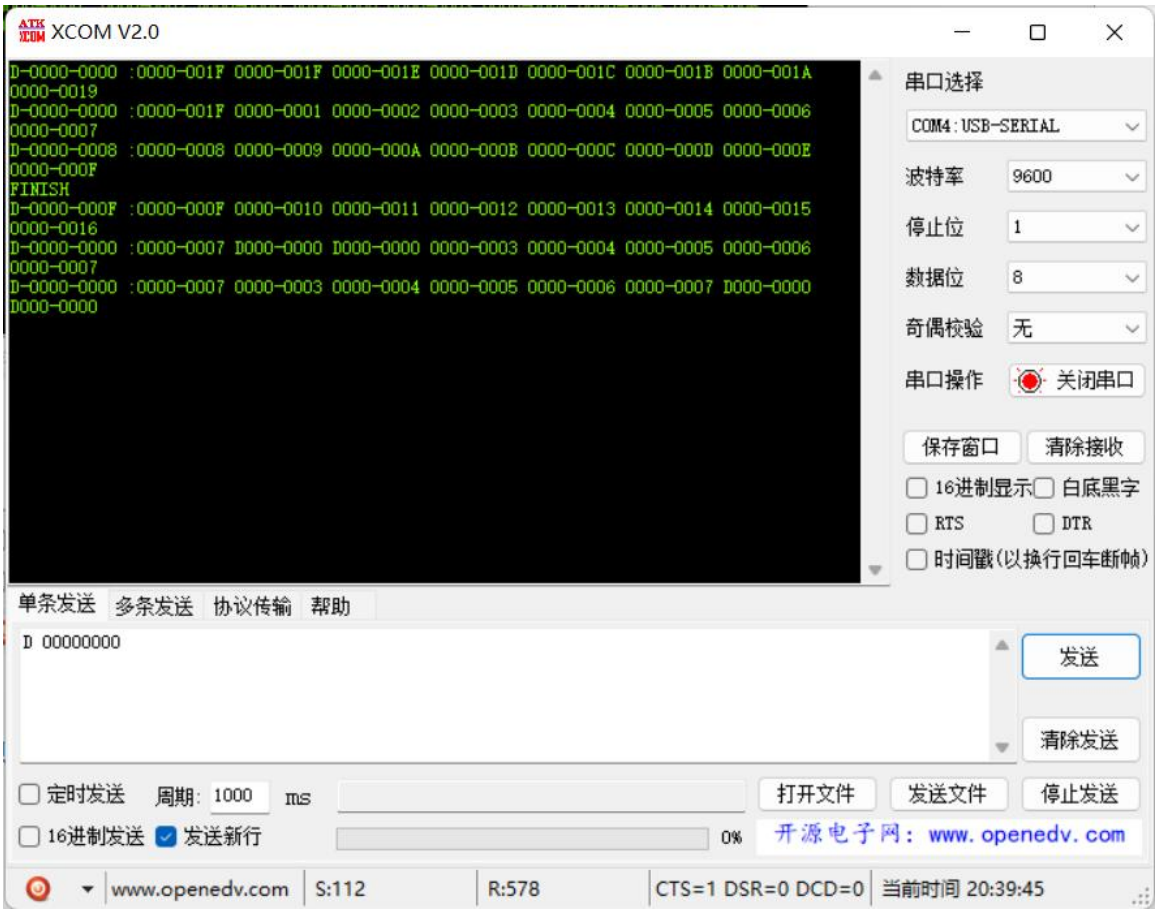

XCOM V2.0
D-0000-0000 :0000-001F 0000-001F 0000-001E 0000-001D 0000-001C 0000-001B 0000-001A
0000-0019
串口选择
COM4:USB-SERIAL
波特率
9600
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
16进制显示
白底黑字
RTS
DTR
时间戳(以换行回车断帧)
单条发送
多条发送
协议传输
帮助
D 00000000
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网：www.openedv.com
www.openedv.com
S:14
R:95
CTS=1 DSR=0 DCD=0
当前时间 20:36:32

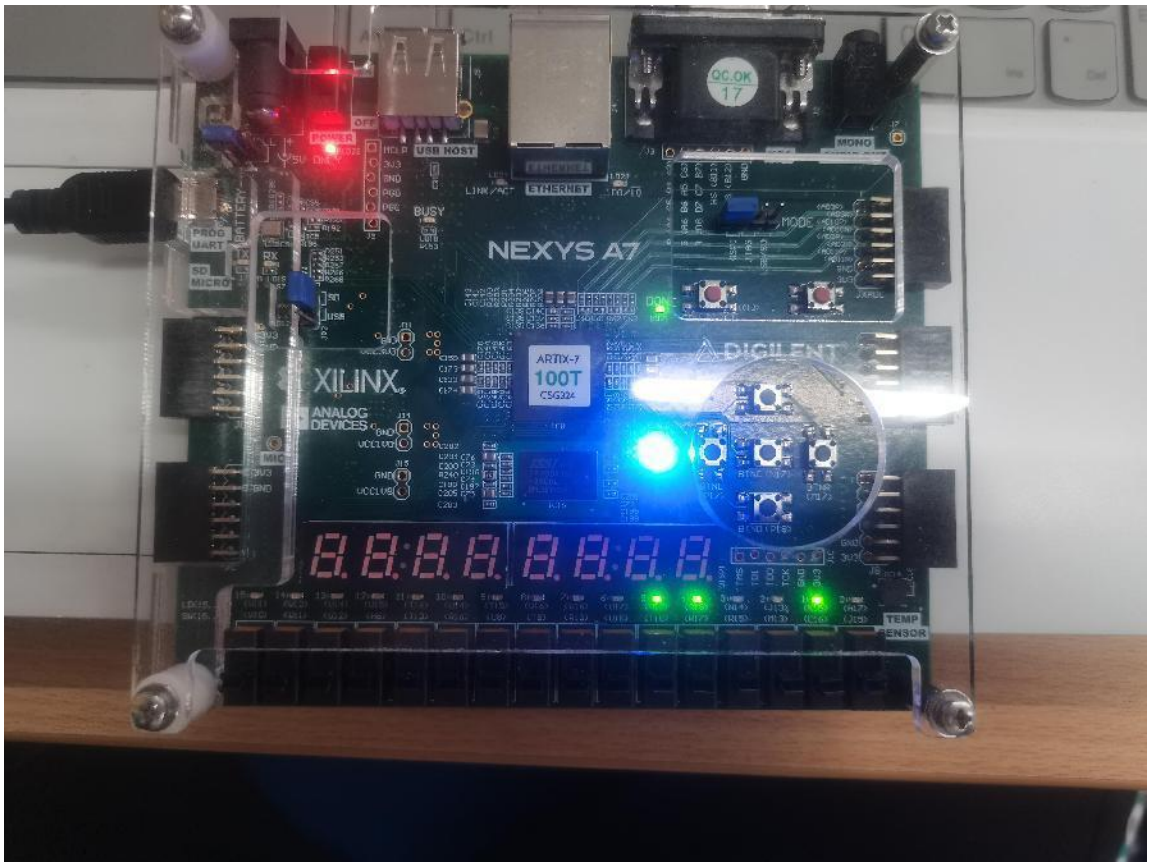

NEXYS A7
XILINX
ANALOG DEVICES
ARTIX-7
100T
CSG324
USB HOST
ETHERNET
PROG UART
SD MICRO
BUSY
DONE
MODE
TEMP SENSOR
QC.OK
17


NEXYS A7
XILINX
ARTIX-7
100T
CSG324
DIGILENT
ANALOG DEVICES
ETHERNET
USB HOST
PROG UART
SD MICRO
BUSY
DONE
MODE
TEMP SENSOR


XCOM V2.0
D-0000-0000 :0000-001F 0000-001F 0000-001E 0000-001D 0000-001C 0000-001B 0000-001A
0000-0019
D-0000-0000 :0000-001F 0000-0001 0000-0002 0000-0003 0000-0004 0000-0005 0000-0006
0000-0007
D-0000-0008 :0000-0008 0000-0009 0000-000A 0000-000B 0000-000C 0000-000D 0000-000E
0000-000F
FINISH
D-0000-000F :0000-000F 0000-0010 0000-0011 0000-0012 0000-0013 0000-0014 0000-0015
0000-0016
D-0000-0000 :0000-0007 D000-0000 D000-0000 0000-0003 0000-0004 0000-0005 0000-0006
0000-0007
串口选择
COM4:USB-SERIAL
波特率
9600
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
16进制显示
白底黑字
RTS
DTR
时间戳(以换行回车断帧)
单条发送
多条发送
协议传输
帮助
D 00000000
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网：www.openedv.com
www.openedv.com
S:100
R:483
CTS=1 DSR=0 DCD=0
当前时间 20:38:56


NEXYS A7
XILINX
ARTIX-7
100T
CSG324
DIGILENT
ANALOG DEVICES
USB HOST
ETHERNET
PROG UART
SD MICRO
MODE
TEMP SENSOR


XCOM V2.0
D-0000-0000 :0000-001F 0000-001F 0000-001E 0000-001D 0000-001C 0000-001B 0000-001A
0000-0019
D-0000-0000 :0000-001F 0000-0001 0000-0002 0000-0003 0000-0004 0000-0005 0000-0006
0000-0007
D-0000-0008 :0000-0008 0000-0009 0000-000A 0000-000B 0000-000C 0000-000D 0000-000E
0000-000F
FINISH
D-0000-000F :0000-000F 0000-0010 0000-0011 0000-0012 0000-0013 0000-0014 0000-0015
0000-0016
D-0000-0000 :0000-0007 D000-0000 D000-0000 0000-0003 0000-0004 0000-0005 0000-0006
0000-0007
D-0000-0000 :0000-0007 0000-0003 0000-0004 0000-0005 0000-0006 0000-0007 D000-0000
D000-0000
串口选择
COM4:USB-SERIAL
波特率
9600
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
16进制显示
白底黑字
RTS
DTR
时间戳(以换行回车断帧)
单条发送
多条发送
协议传输
帮助
D 00000000
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网：www.openedv.com
www.openedv.com
S:112
R:578
CTS=1 DSR=0 DCD=0
当前时间 20:39:45



首先第一张图展示了上板子的初始情况。分布式存储器在例化时存储的第一位是 31，代表我要排 31 个数的数组。然后接下来是 31 到 1 降序排列的数组。通过串口也可以看出存储情况。然后我们运行 run 的指令，板子上通过 16 位的 led 显示了 cycles 的情况。利用串口的 D 指令，我们发现 31 位的数组已经变成了升序排列的顺序。当我们再按几次 run 按钮后，发现 cycles 的大小一致，同时远远小于第一次排序的 cycles 大小。接着我们利用串口修改地址的数值，改为要排 7 个数，并修改了前两位数的大小，利用串口展示修改情况。然后再按一次 run 按钮，通过板子我们可以发现 cycles 的大小又一次发生了变化，同时会比排 31 个数要小很多。排序完成之后利用串口发现排序又完成了，同时再按几次 run 按钮，cycles 也不再变化，且比第一次不再变化的数值小很多。

总结

总结：在本次实验中，我加深了对于分布式存储器的学习理解、同时也进一步了解了关于一些多端口存储器堆端口的含义。同时逐渐回忆起了有关 IP 核的相关知识。初步简单地了解了有关串口和 SDU 工作原理和指令的有关知识。本次实验我进一步理解了数据通路的美妙，以及顶端封装对于一名普通程序员的友好之处。

本次实验的体验比实验一好一些（也可能是这次检查实验较早的缘故，有时间改掉程序的 bug），不过在实验过程中，我还是自己写出了锁存器的 bug，同时收到 ppt 实验文档的影响，我接错了 rxd 和 txd 的串口，在很长的一段时间里我都以

为是自己程序的逻辑问题，直到我试着修改了输入输出接口的顺序。一切都变得正常了。上次实验中让自己陷入困境的 btn 信号去毛刺的问题，在本次实验中迎刃而解。果然人就是在不断地磨砺中一步步成长的吧。在这里依旧小小的吐槽一下实验文档上下文连接性的问题，能否再详尽一些，以及拼写错误这样的错误能否有人专门负责矫正一下，否则因此而昏头转向的同学们真的失去了实验的体验。