LAB1 实验报告

PB22111639 马筱雅

1 实验目的

1. 学习 ALU，RF，存储器的实现思路并掌握其功能、时序及其应用
2. 利用状态机的思想，采用单端口存储器和 ALU 实现冒泡排序
3. 学会使用 ip 核生成存储器，并比较分布式和块式存储器的特性

2 逻辑设计

2.1 ALU 和 RF 设计

2.1.1 ALU

ALU 要求实现加减，有符号比较，无符号数比较，与，或，或非，异或，左移，逻辑右移，算术右移，赋值等功能。则对除有符号比较外的操作均可直接运用相应符号处理。对于有符号数比较，可以应用强制类型转换，如图所示

```
wire signed [31:0] su0, su1;
assign su0 = src0;
assign su1 = src1;
always @(*) begin
    case (op)
        4'd0: res = src0 + src1;
        4'd1: res = src0 - src1;
        4'd2: res = su0 < su1;
        4'd3: res = src0 < src1;
        4'd4: res = src0 & src1;
        4'd5: res = src0 | src1;
        4'd6: res = ~(src0 | src1);
        4'd7: res = src0 ^ src1;
        4'd8: res = src0 << src1[4:0];
        4'd9: res = src0 >> src1[4:0];
        4'd10: res = src0 >>> src1[4:0];
        4'd11: res = src1;
        default: res = 0;
    endcase
```

```
end
```

2.1.2 RF

RF 为写优先且不改变 R0 的值，先定义 32 个 32 位的寄存器，如果写使能为真且待读取寄存器和待写入寄存器相同，则读出的值为待写入值，其余情况，输出值为寄存器原有值，从而实现写优先。在时钟上升沿，只有写使能为真且待写入寄存器不为 R0，才能执行赋值操作，从而不改变 R0 的值。

```
assign rd0 = (wa == ra0 && we) ? wd : rf[ra0];
assign rd1 = (wa == ra1 && we) ? wd : rf[ra1];
always @(posedge clk) begin
    if(we && wa != 0)
        rf[wa] <= wd;
end
```

2.2 DRAM 和 BRAM 的设计和实现

方法：使用 ip 核的方式例化，并使用 coe 文件进行初始化

2.3 冒泡排序的设计

2.3.1 逻辑解释

- 冒泡排序模块使用单端口存储器，每次只能读出一个数据，则定义 data0 为读出的数据，data3 表示要存储的数据，用 src0 和 src1 分别表示每次进行比较的两个数据，同时作为 ALU 的两个操作数，point 表示指针，即在 DRAM 中取数据的位置，ps 表示每次循环终止时的指针位置。

- point 初始化为 0，ps 初始化为 1023，当一次循环初始时，src0 <= data0, 表示读出的第一个数据，同时 point+1，表示准备取出下一个数据，在下一个周期，src1 <= data0, 然后 src1 和 src0 进行比较，对比较结果是否满足顺序要求进行分类。

- 如果不满足，则写使能 we 变为真，令 point-1，表示要改变第一个索引对应的值，data3 <= src1，存储第一个值；在下一个周期, point + 1，data3 <= src0，存储第二个值，存储结束后写使能 we 变为 0，如果 point!=ps，则 point+1，表示继续取下一个数据进行比较，如果相等，则表示此次循环结束，则 point 变为 0，ps-1，然后进行下一次循环。

- 如果 src0 和 src1 满足顺序要求，则当 point!=ps 时，把 src1 赋值给 src0，上一次比较中的第二个数据成为下一次比较中的第一个数据，当

point==ps 时，point 变为 0，ps-1 表示此次循环结束，进行下一次循环。

```
if(!done) begin
    case (current_state)
      s0: begin
        point <= point + 1;
        if(ps == 0) point <= 0;  //表示排序结束，索引指针归零
      end
      s1:;
      s2: begin
        if((!res[0] && up)||(res[0] && !up)) begin //得出大小比较的结论，来判断是否需要写入
          we <= 1;
          point <= point - 1;              //表示要写入第一个数据
          data3 <= src1;                   //data3 表示目标数据
        end
        else begin
          if(point == ps) begin              //本次循环结束
            ps <= ps - 1;                  // ps - 1
            point <= 0;                    // point 从 0 开始
          end
          else point <= point + 1;           //否则继续下一个数据
        end
      end
      s3: begin
        point <= point + 1;                //存下一个数据
        data3 <= src0;
      end
      s4: begin
          we <= 0;                     //s4 之后存完，we 变为 0
         if(point == ps) begin          //判断本次循环是否结束
              ps <= ps - 1;
              point <= 0;
          end
        else point <= point + 1;
      end
      default:;
    endcase
```

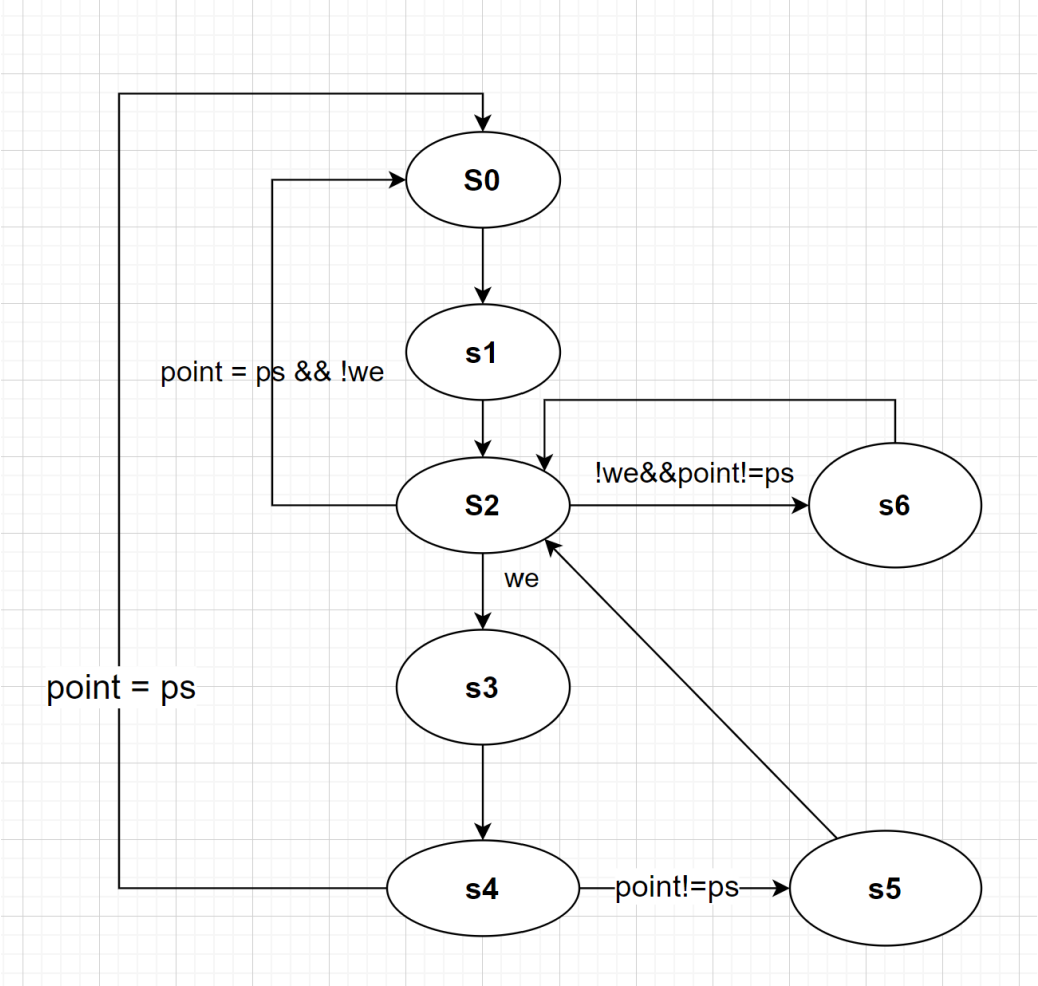
- 冒泡排序结束标志 done：当启动排序时 done<=0，当 ps 变为 0 时表示循环结束，done<=1

```
always @(posedge clk)
begin
  if(!rstn)
    done <= 1;
```

```
    else if(start)
      done <= 0;
    else if(ps == 0 && we == 0)
      done <= 1;
  end
```

2.3.2 状态机的设计

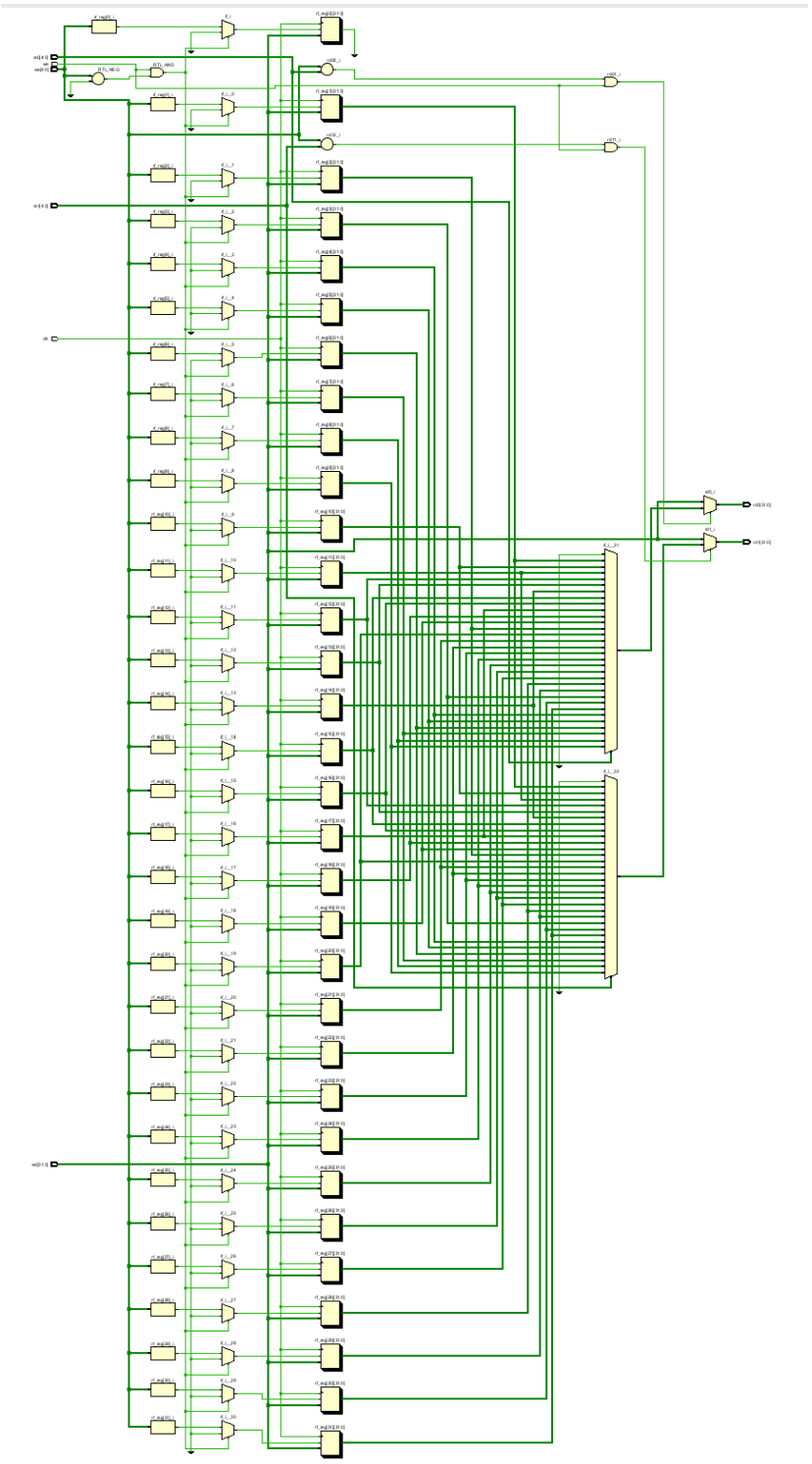
s0：一次循环初始，此时指针 point = 0，取第一个数据 src0，而后 point + 1，取下一数据
s1：取第二个数据，即 src1
s2：判断 src0 和 src1 的大小是否符合升序/降序，从而进行状态转换
s3：存储第一个数据
s4：存储第二个数据
s5：保持数据不变
s6：src0 <= src1，从而比较下一组相邻数据



3 电路图

3.1 ALU 和 RF

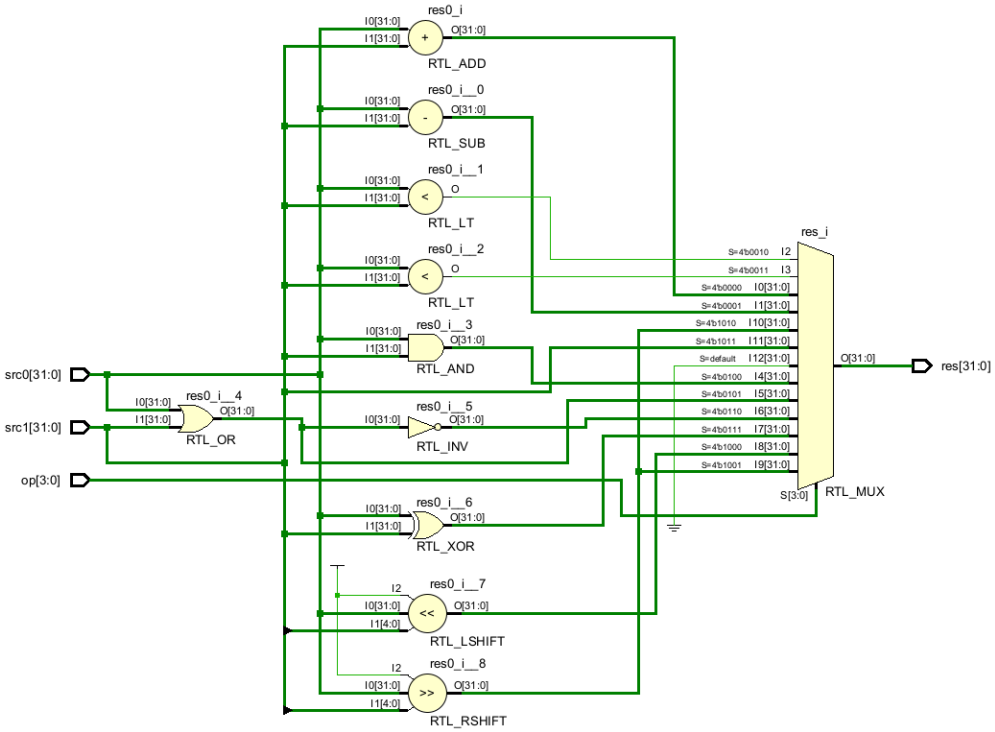
3.1.1 RTL 电路图



资源使用量

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxe	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
N RegFile	611	992	256	128	113	1

3.1.2 ALU 电路图



ALU 资源使用量

Name	Slice LUTs (63400)	F7 Muxes (31700)	Bonded IOB (210)
N ALU	364	1	100

3.2 DRAM 和 BRAM

3.2.1 BRAM 资源使用量

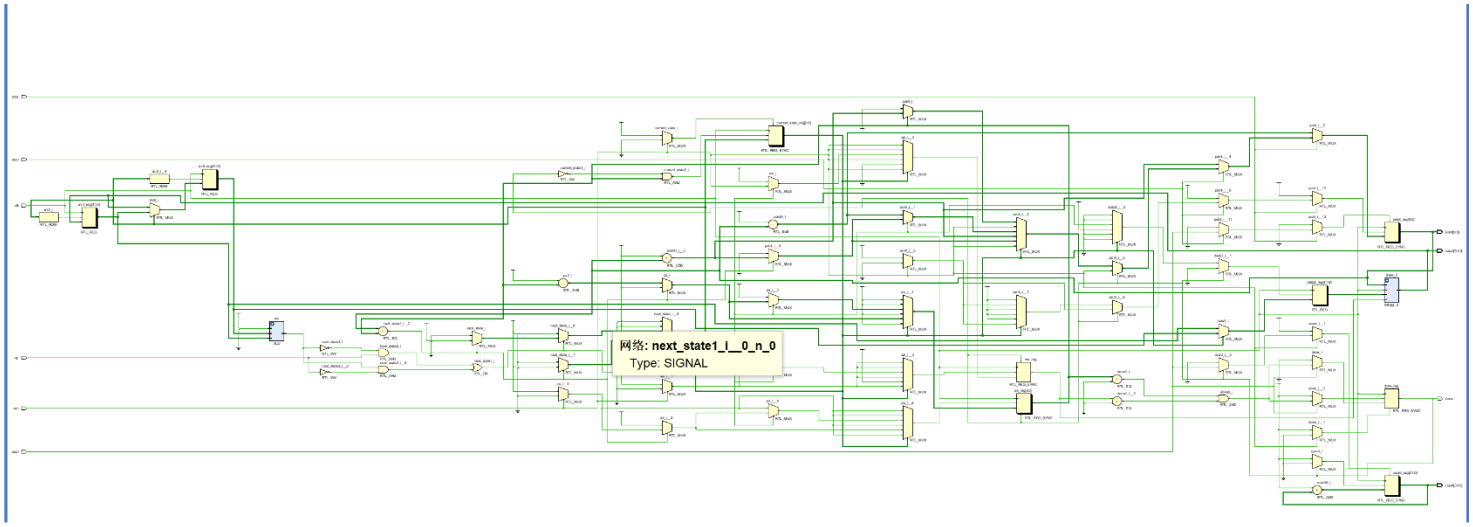
Name	Block RAM Tile (135)	Bonded IOB (210)
N BRAM_T	1	77
bram (BRAM)	1	0

3.2.2 DRAM 资源使用量

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)
N DRAM_T	548	32	256	128	76
dram (DRAM_0)	548	32	256	128	0

3.3 SRT

电路图



资源使用量

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
N BubbleSort	695	185	256	128	81	1
alu (ALU)	139	0	0	0	0	0
dram_0 (DRAM_0)	548	32	256	128	0	0

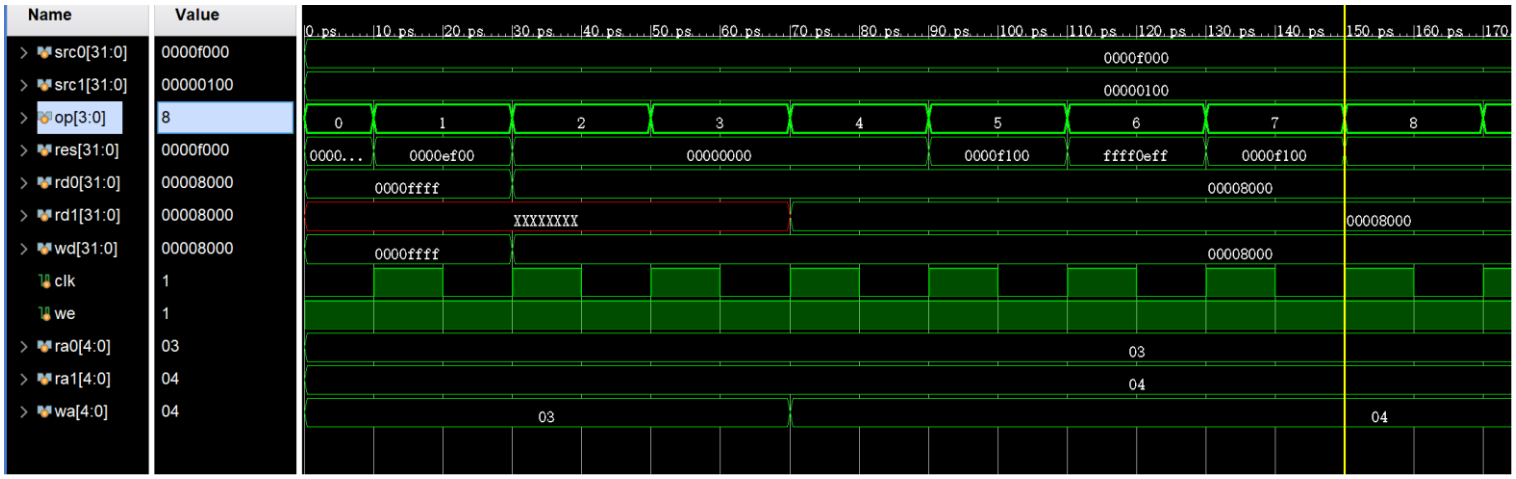
性能：

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Require
Path 1	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[0]/CE	4.544	1.972	2.572	
Path 2	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[1]/CE	4.544	1.972	2.572	
Path 3	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[2]/CE	4.544	1.972	2.572	
Path 4	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[3]/CE	4.544	1.972	2.572	
Path 5	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[4]/CE	4.544	1.972	2.572	
Path 6	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[5]/CE	4.544	1.972	2.572	
Path 7	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[6]/CE	4.544	1.972	2.572	
Path 8	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[7]/CE	4.544	1.972	2.572	
Path 9	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[8]/CE	4.544	1.972	2.572	
Path 10	15.074	7	4	12	bubbleSort/src1_reg[1]/C	bubbleSort/ps_reg[9]/CE	4.544	1.972	2.572	

4 仿真结果和分析

4.1 ALU 和 RF

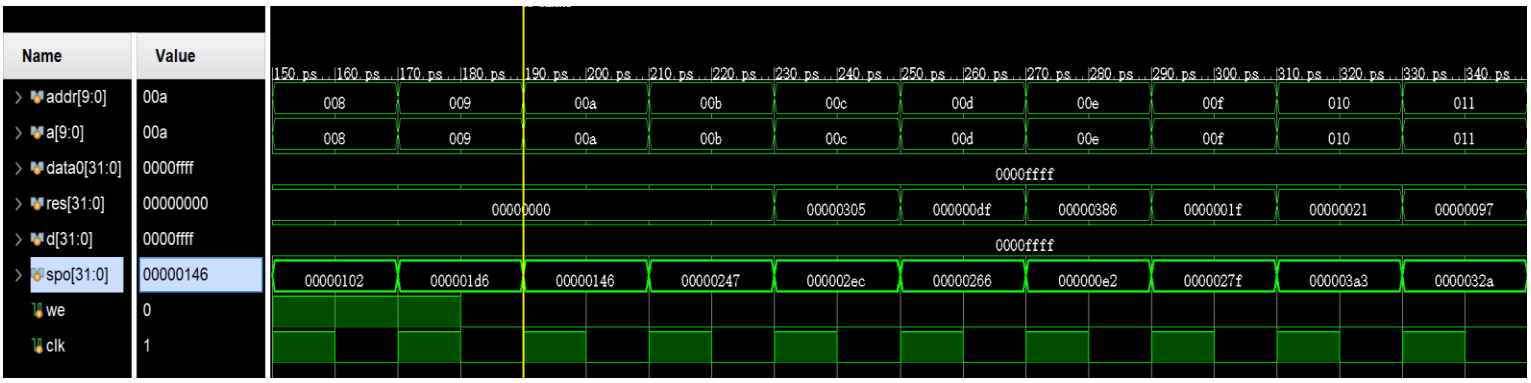
- 对 ALU 和 RF 进行同步仿真，仿真波形图中，src0 和 src1 为 ALU 的两个操作数，op 为操作码，res 为运行结果，由图可知，运行结果符合定义运算
- RF 两个读端口为 ra0，ra1，读出数据端口为 rd0，rd1，写入数据为 wd，写使能为 we，写入地址为 wa。
- 由波形图，可知 we = 1，此时 wa=3，ra0 为 3，wd=0000ffff，由于写优先，所以此时读出 rd0 = 0000ffff，当 wd 改变为 00008000 时，rd0 同时改变。当 wa=4，ra1=4 时，此时 wd = 00008000，则 rd1 也为 00008000，体现了写优先。



4.2 DRAM 和 BRAM

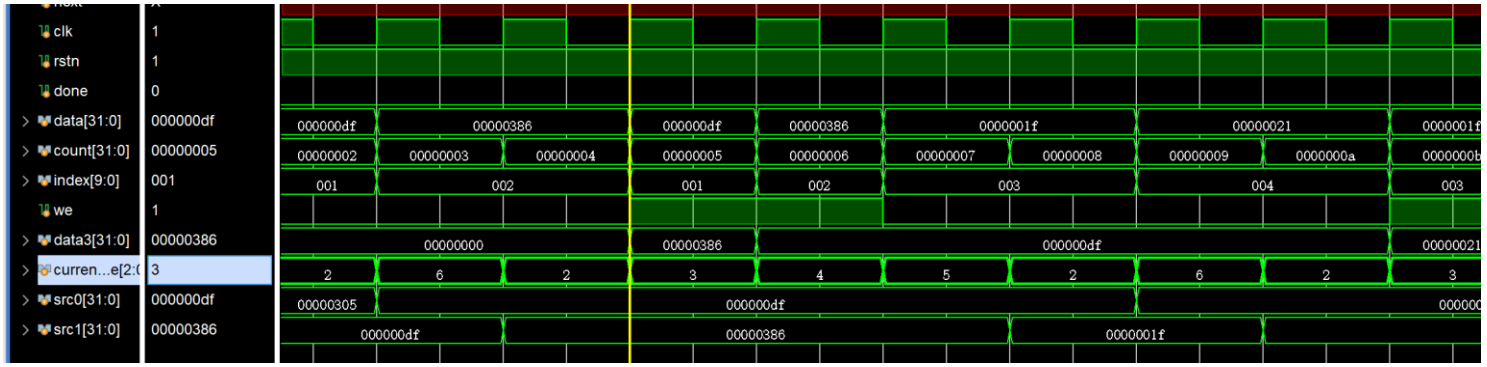
- 对 BRAM 和 DRAM 在同一个仿真文件中进行仿真，其中 addr 是 bram 的取数地址，res 为取出的数据，data0 为写入数据，we 为 bram 和 dram 的写入信号，a，d，spo 分别为 darm 的地址，写入数据，所取数据。每个周期 a 和 addr 均加 1。

- 根据仿真波形可以看出，bram 相对于 dram 有较大的延迟，dram 在给出取数地址时即可取出数据，但 bram 需要时钟才能读出数据



4.3 SRT 仿真测试

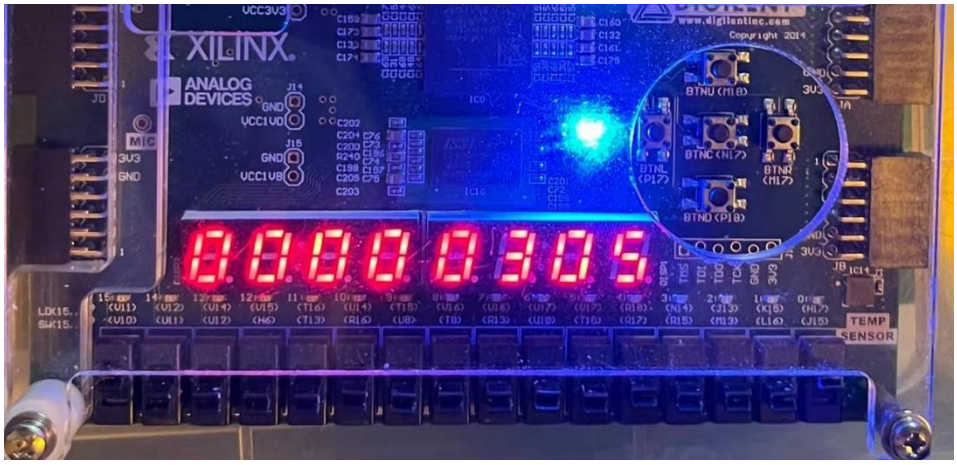
- 解析仿真图：index 为指针，在 current_state 为 6 时，说明 上一次比较的两个数不需要交换，则此时 src0 <=src1，src0 为 000000df，当 index=1 时，读出数据为 data，src1<=data,此时 src1 为 00000386，比较两个数据，需要交换，于是下周期 we = 1，data3 = src1，index - 1，进行存储 src1，接着 point +1 存储 src0，存储完毕 we 变为 0，index + 1，接下来进行下一次比较。运行结果符合预期。



5 上板结果

coe 文件数据为随机排列的 0-1023，用 sw1=1 表示升序，否则为降序；led9-led0 为 index 索引

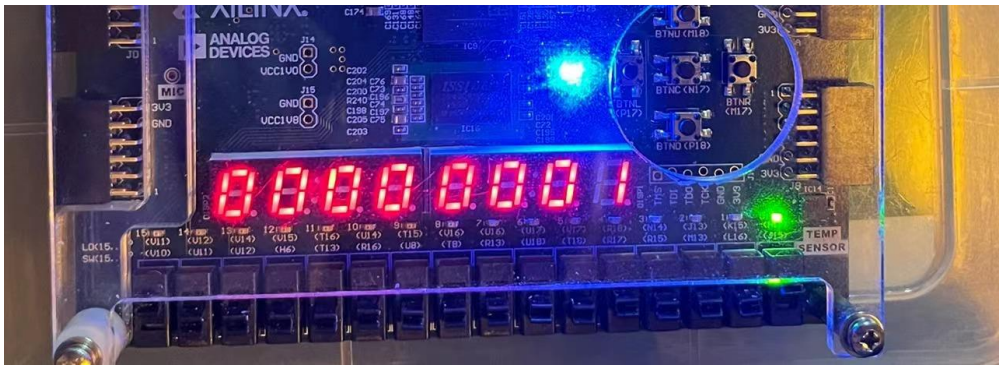
- 未排序时



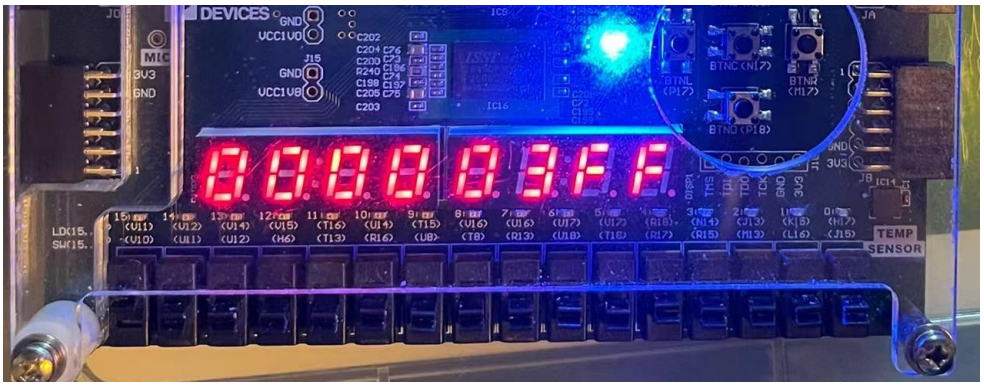
索引为 0 的值为 32’h305

- 升序排列时
  - 索引为 0

  - 索引为 1



- 降序排列时



- 周期数

6 遇到的问题

1. index 的问题，上板运行后，每次重新进行排序会导致少一个数据
   原因：如果 index 不为 0，由于 index 是指针，每次重新排序时取出索引为 index 的数据，导致索引小于 index 的数据未参加排序。
   解决方法：在每次重新排序，即按下 enter 键时，令 index 为 0.
2. dram 的问题，we 和数据转移的状态不对，还有状态机设置不合理，中间尝试去除 s0 和 s5 状态，发现出错。
   原因：忽略了 dram 取数的特性，dram 取数不依赖时钟上升沿，导致每次在时钟上升沿时当前状态分别为 s1 和 s2 时取第一个和第二个操作数的时间晚，从而使得 we 晚了一个周期，存的数字不对。
   解决方法：在 next_state 分别为 s1 和 s2 时，赋值第一个和第二个数据，并保留状态数 s0 和 s6