

### 1. 实验目的

1. 掌握CPU 的外设I/O 模块的设计方法。理解 I/O 地址空间的译码设计方法。
2. 掌握Vivado 仿真、实现、板级验证方式。
3. 通过扩展新指令的实现，深入理解CPU对指令的译码、执行原理和实现方式。

### 2. 实验平台及器材

1. 计算机 1 台 ( 尽可能达到 8G 及以上内存 ) ;
2. Xilinx 的 Vivado 开发套件 (2020.2 版本 ) ;
3. Xilinx 的 EGO1 FPGA 开发板。

### 3. 实验任务

1. 本次实验的主要任务是在实验三完成的单周期 CPU 核基础上进行外设拓展，增加 CPU 对 I/O 口的读写支持，以及对新指令的功能支持，并在 EGO1 实验板上完成板级测试验证。
2. 文件树

- ▼ ● **sc\_cpu\_iotest** (sc\_cpu\_iotest.v) (8)
  - clkgen : clock\_and\_mem\_clock (clock\_ar
  - inportone : in\_port (in\_port.v)
  - inporttwo : in\_port (in\_port.v)
  - ▼ ● computer\_main : sc\_computer\_main (sc\_
    - > ● cpu : sc\_cpu (sc\_cpu.v) (12)
    - ▼ ● imem : sc\_instmem (sc\_instmem.v) (
      - > □ irom : lpm\_rom\_irom (lpm\_rom
    - ▼ ● dmem : sc\_datamem (sc\_datamem.v)
      - io\_data\_mux : mux2x32 (mux2x3
      - > □ dram : lpm\_ram\_dq\_dram (lpm
      - io\_output\_reg : io\_output (io\_out
      - ▼ ● io\_input\_reg : io\_input (io\_input.v)
        - io\_input\_mux2x32 : io\_input
    - dec54 : out\_port\_hex2dec (out\_port\_hex:
    - dec32 : out\_port\_hex2dec (out\_port\_hex:
    - dec10 : out\_port\_hex2dec (out\_port\_hex:
    - > ● display : display (display.v) (1)

< Coefficient Files (2)

▼  Coefficient Files (2)

 lpm\_rom\_irom\_io.coe

 sc\_datamem.coe

#### ▼ Constraints (1)


constrs\_1 (1)

sc\_computer\_iotest.xdc

### Simulation Sources (3)

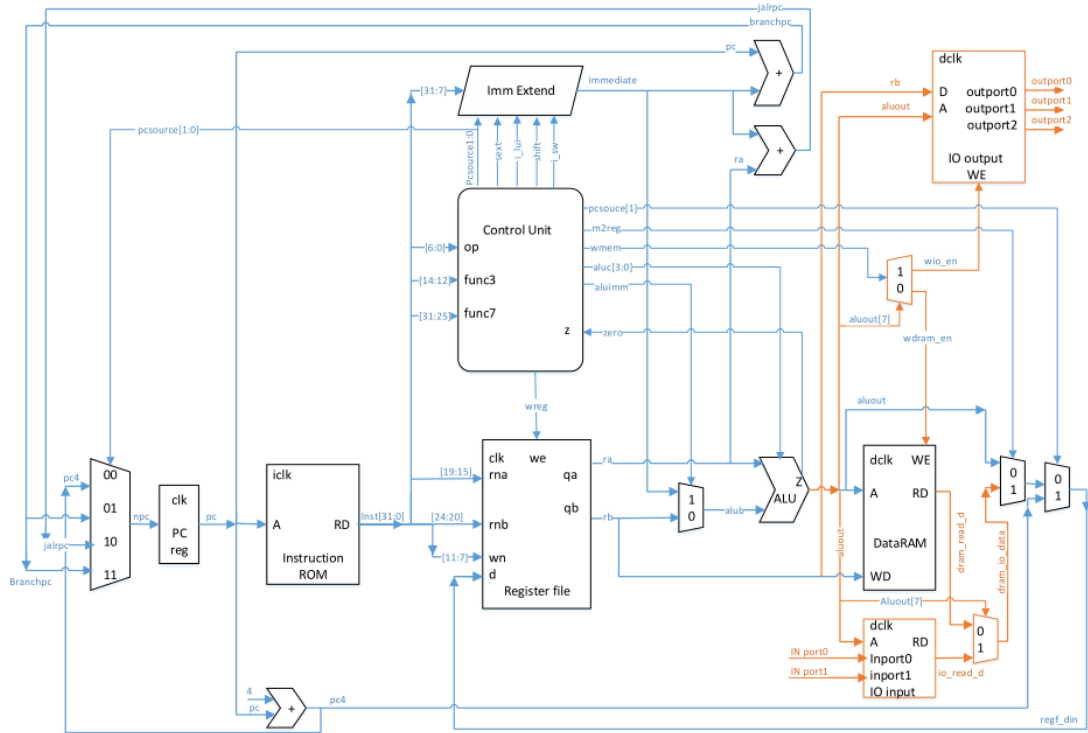
sim\_1 (3)

```
> TB_sc_computer_iotest (TB_sc.
```

>  Coefficient Files (2)

### 3. I/O模块代码

## 按框架图连接模块



### 1. 在 sc\_datamem 模块中完成子模块 io\_input 和 io\_output 的例化

```

io_output io_output_reg (
    .PAN(PAN),
    .out_port1(out_port1),
    .out_port2(out_port2),
    .addr(addr),
    .datain(datain),
    .write_io_enable(write_io_enable),
    .io_clk(dmem_clock),
    .resetn(resetn)
);

//IOinput , io_input , add here
io_input io_input_reg(
    .addr(addr),
    .io_clk(dmem_clock),
    .io_read_data(io_read_data),
    .in_port0(in_port0),
    .in_port1(in_port1)
);

```

2. 在软核顶层 sc\_computer\_main 模块中例化 sc\_instmem 和 sc\_datamem 两个存储器模块  
 需注意其中dmem\_clock,imem\_clock的传参，因为实验使用个独立的时钟生成模块  
 clock\_and\_mem\_clock 为指令存储器和数据存储器分别生成时间

```

//sc_instmem , instruction memory.add here
sc_instmem imem(
    .addr(pc),
    .inst(inst),
    .clock(clock),
    .imem_clock(imem_clock)
);

//sc_datamem data memory and IO module.add here
sc_datamem dmem (
    .resetn(resetn),
    .addr(aluout),
    .datain(data),
    .we(wmem),
    .dataout(memout),
    .clock(clock),
    .dmem_clock(dmem_clock),
    .PAN(PAN),
    .out_port1(out_port1),
    .out_port2(out_port2),
    .in_port0(in_port0),
    .in_port1(in_port1)
);

```

### 3. 在整个项目的顶层文件 sc\_cpu\_iotest 中例化 sc\_computer\_main 等下层模块

本实验模块层级较多，且部分变量会反复作为输入/输出被传递，例化时应特别注意不同文件中定义的变量名可能不同，要准确对应

```

sc_computer_main computer_main(
    .resetn(sys_rst_n),
    .clock(clock_out),
    .imem_clock(imem_clock),
    .dmem_clock(dmem_clock),
    .in_port0(inport0),
    .in_port1(inport1),
    .PAN(PAN),
    .out_port1(out_port1),
    .out_port2(out_port2)
);

```

#### 4. 扩展指令

先要求扩展 ALU 模块功能，使其支持求两个 32 比特数的汉明距离的操作。为此，我们自定义该操作为 R-type 指令 hamd rd, rs1, rs2，其编码中 func3=111，func7=0100000，op=0110011，则 hamd x16, x14, x15 的指令码为 0100000|01111|01110|111|10000|0110011 = 40f77833

1. sc\_cu 模块中为新指令（hamd）定义控制变量 i\_hamd，并为其分配不相冲突的 aluc 代码 1111

```
wire i_hamd = r_type & (func3 == 3'b111) & (op == 7'b0110011);
```

2. alu 模块增添汉明距离运算操作

循环语句应使用 always 过程赋值，且变量类型相应更改为 reg 类型

与之前 assign 对应 wire 不同

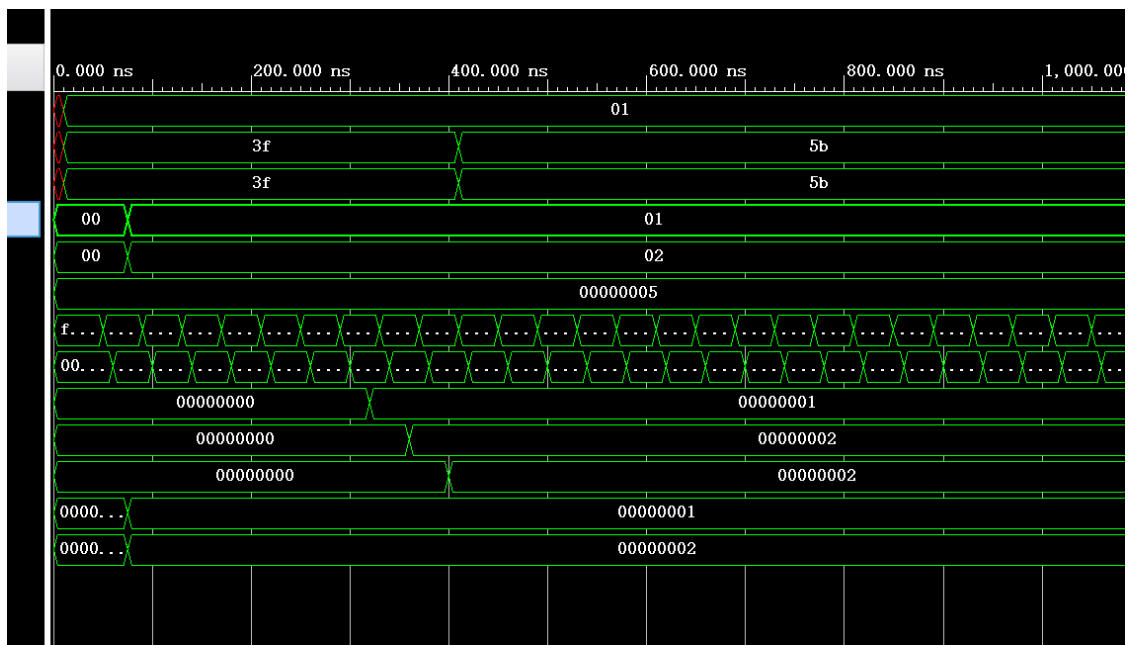
```

if(aluc == 4'b1111) begin
    cnt = 0;
    for(i=0; i<31; i=i+1)begin
        if(a[i]!=b[i]) cnt = cnt + 1;
    end
    s = cnt;
end
if (s == 0 ) z = 1;
else z = 0;

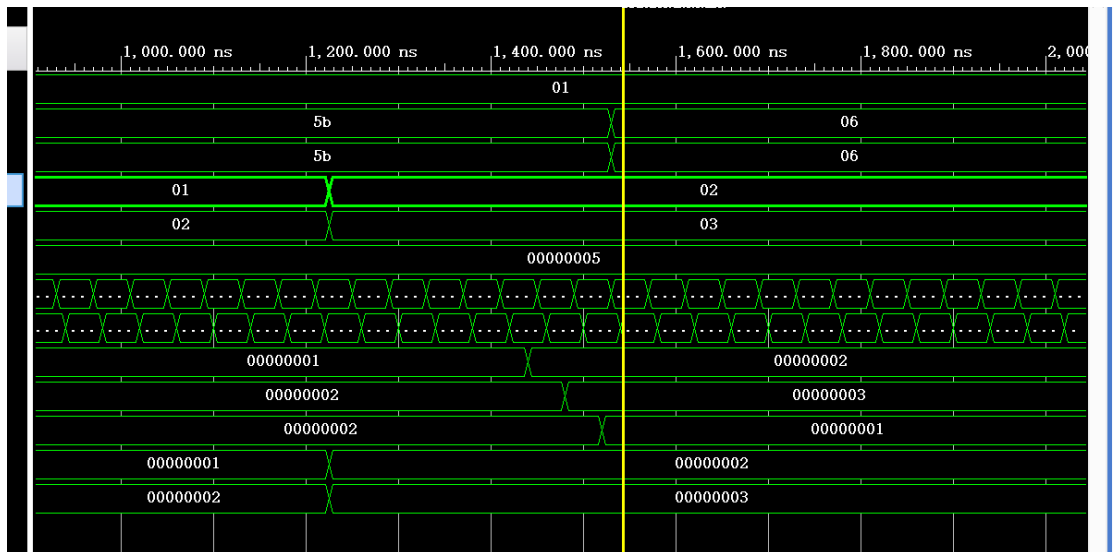
```

#### 5. 仿真

400 ns 时，正确完成第一轮循环，out\_port0 和 out\_port1 正确显示了 in\_port0 和 in\_port1 的数据，并在 out\_port2 中正确得到了二者加和的结果

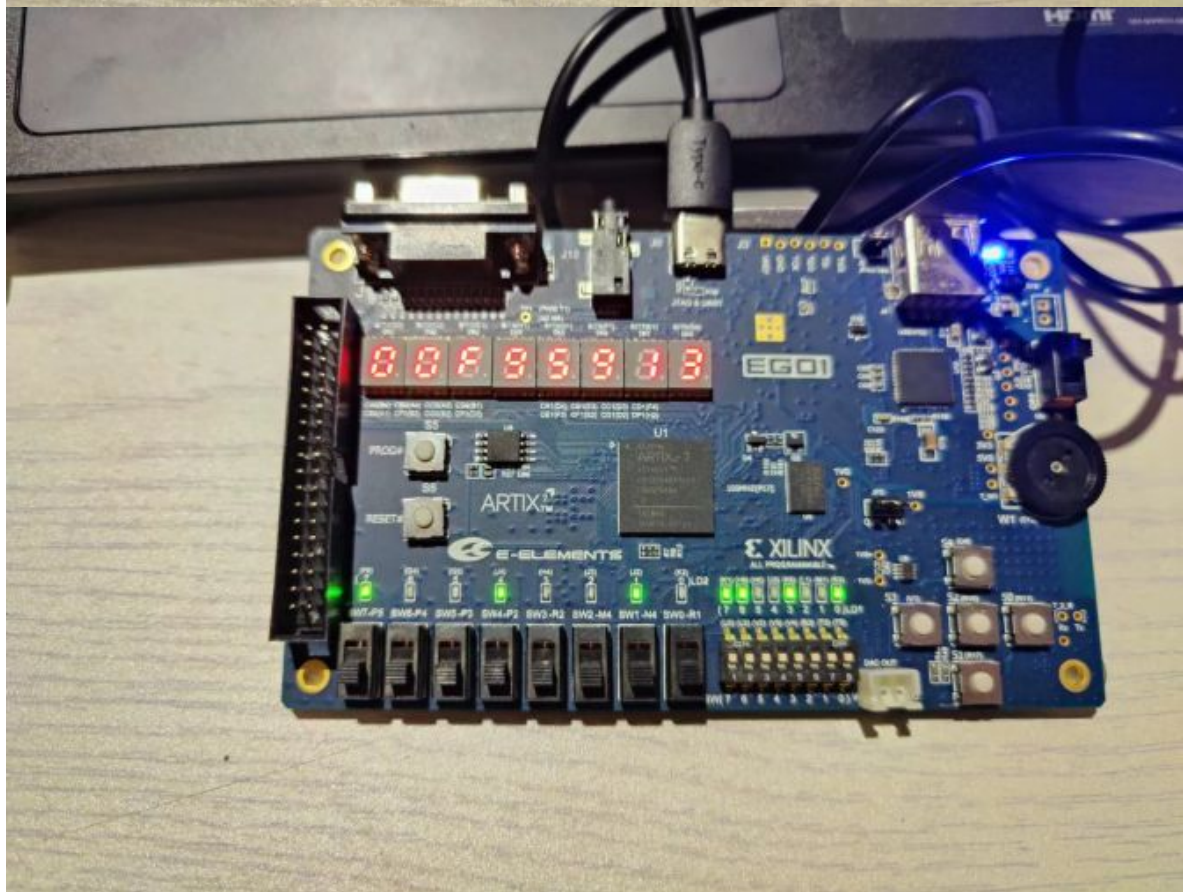
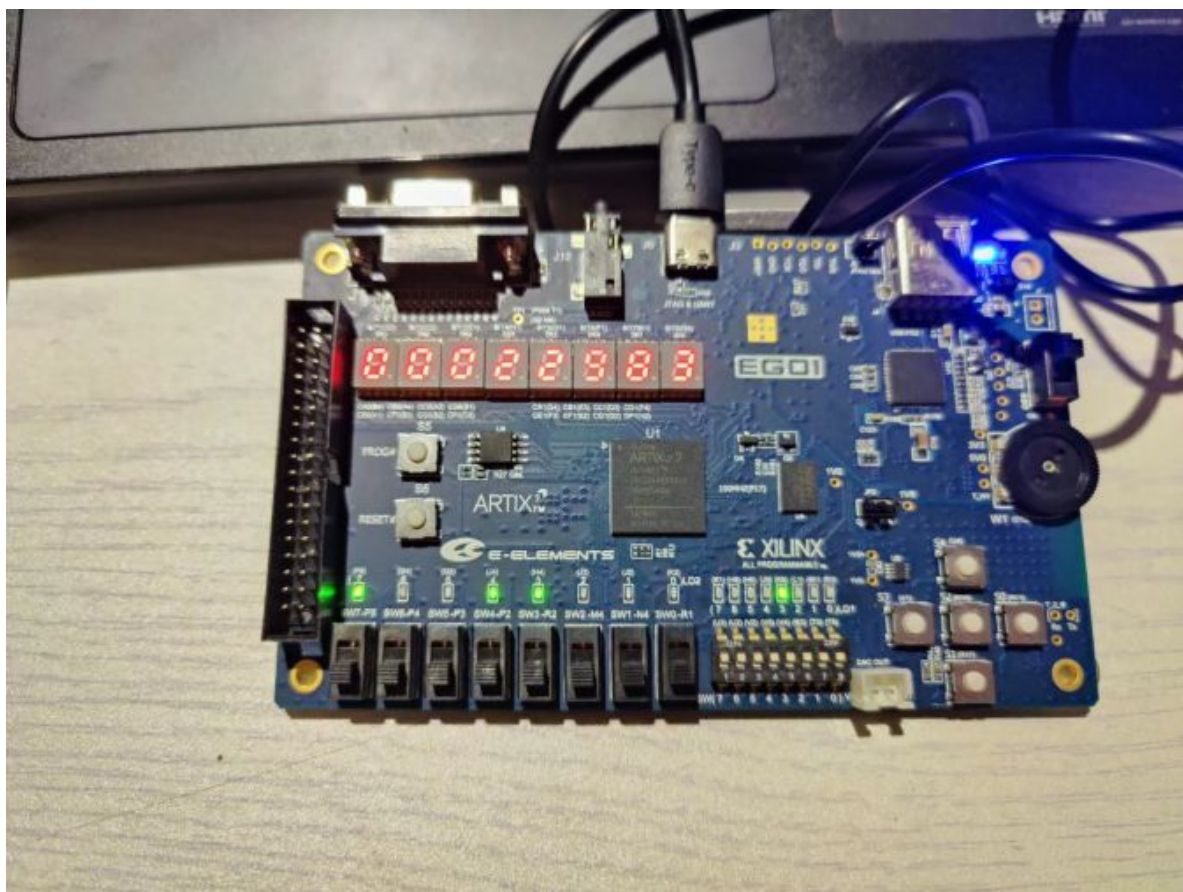


程序正确完成循环， out\_port0 和 out\_port1 正确显示了 in\_port0 和 in\_port1 的数据，并在 out\_port2 中正确得到了二者求汉明距离的结果



## 6. 板上验证





汉明距离正确



