# 中国科学技术大学计算机学院

# 《数字电路实验》报告


实验题目：lab1_运算器与排序

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020 年 4 月 26 日

计算机实验教学中心制

2019 年 09 月

# 实验目的

1. 掌握算术逻辑单元（ALU）的功能，加/减运算时溢出、进位/借位、零标志的形成及其应用
2. 掌握数据通路和控制器的设计和描述方法理及使用

# 实验环境

1. PC 一台
2. Windows 或 Linux 操作系统
3. Vivado
4. vlab.ustc.edu.cn
5. Logisim
6. Nexys4DDR 开发板

# 实验设计

## 实验 1 运算器(ALU)

### 设计目标

待设计的 ALU 模块的逻辑符号如图 1 所示。该模块的功能是将两操作数（a，b）按照指定的操作方式（m）进行运算，产生运算结果（y）和相应的标志（f）。

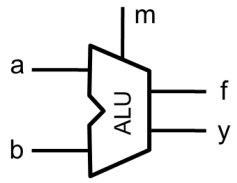


图 1

操作方式 m 的编码与 ALU 的功能对应关系如表 1 所示。表中标志 f 细化为进位/借位标志（cf）、溢出标志（of）和零标志（zf）；“*”表示根据运算结果设置相应值；“x”表示无关项，可取任意值。例如，加法运算后设置进位标志（cf）、of 和 zf，减法运算后设置借位标志（cf）、of 和 zf。

| m | y | cf | of | zf |
|---|---|---|---|---|
| 000 | a + b | * | * | * |
| 001 | a - b | * | * | * |
| 010 | a & b | x | x | * |
| 011 | a \| b | x | x | * |
| 100 | a ^ b | x | x | * |

| | | | | |
|---|---|---|---|---|
| 其他 | x | x | x | x |

表 1

## 端口声明

端口声明如下：

```
module alu #( parameter WIDTH = 32 )       //data width
       ( output reg [ WIDTH - 1: 0 ] y,    //output data
         output reg zf,                    //zero flag
         output reg cf,                    //carry flag
         output reg of,                    //overflow flag
         input [ WIDTH - 1: 0 ] a, b,      //input data
         input [ 2: 0 ] m );
```

## 设计实现

按照编码，运用 case 语句，对不同情况下的输出，赋对应的值即可，具体见核心代码。

## 核心代码

核心代码如下：

```
always @( * )
  begin
    case ( m )
      ADD:
        begin
          { cf, y } = a + b;
          of = ( ~a[ WIDTH - 1 ] & ~b[ WIDTH - 1 ] & y[ WIDTH - 1 ] ) | ( a[ WIDTH - 1 ] & b[
WIDTH - 1 ] & ~y[ WIDTH - 1 ] );
        end
      SUB:
        begin
          { cf, y } = a - b;
          of = ( ~a[ WIDTH - 1 ] & b[ WIDTH - 1 ] & y[ WIDTH - 1 ] ) | ( a[ WIDTH - 1 ] & ~b[
WIDTH - 1 ] & ~y[ WIDTH - 1 ] );
        end
      AND:
        y = a & b;
      OR:
        y = a | b;
      XOR:
        y = a ^ b;
      default:
        begin
          y = 0;
          cf = 0;
```

```
        of = 0;
      end
  endcase
  zf = ~|y;
end
```

### 仿真结果

仿真结果如下(图 2)：

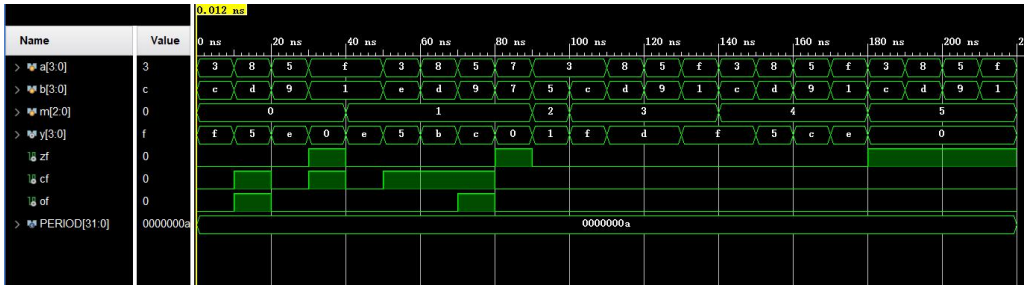


图 2

### 结果分析

仿真结果表明设计，运行效果与设计目标相同。

## 实验 2 排序

### 设计目标

#### 基本目标

利用前面设计的 ALU 模块，辅之以若干寄存器和数据选择器，以及适当的控制器，设计实现四个 4 位有符号数的排序电路，其逻辑符号如图 3 所示。

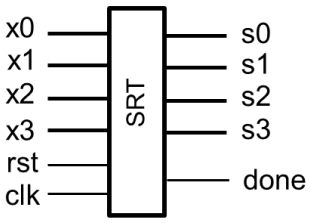


图 3

#### 进阶目标

实现输出结果为递减的排序器，以及运用两个 ALU 的排序器。

### 端口声明

端口声明如下：

```
module sort_new #( parameter N = 4 )                //data width
       ( output reg [ N - 1: 0 ] s0, s1, s2, s3,    //output data
         output reg done,                           //finish flag
         input [ N - 1: 0 ] x0, x1, x2, x3,         //input data
         input clk, rst,                            //clock,reset
         input opr                                  //1(descending) 0(incremental)
       );
```

输入部分中的 opr，为 1 时输出的排序结果为递减，为 0 时排序结果为递增（在基本目标中不含 opr 端口）。

## 设计实现

### 基本目标

数据通路如下：

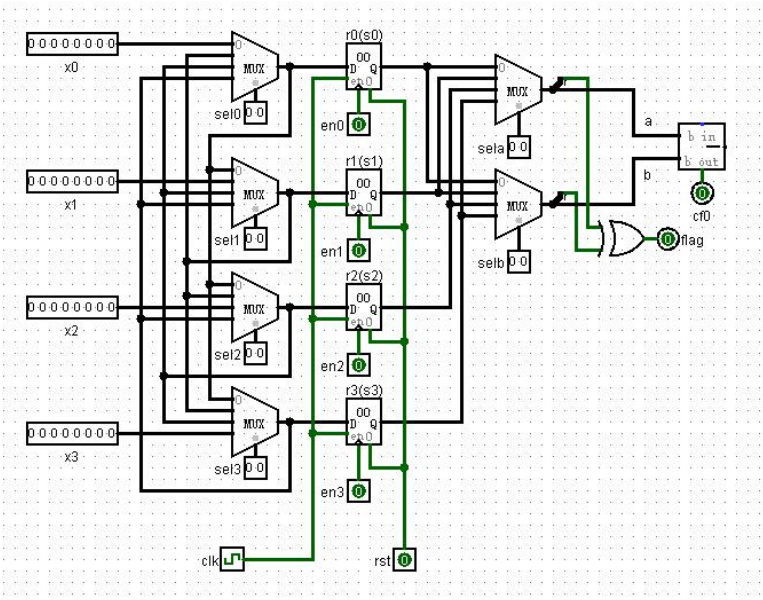


图 4（注：flag=a[N-1]^b[N-1]）

实现思路为在 LOAD 状态，将数据 x0-x3 装入 r0-r3；在 CX 阶段，每次选择两个数送入 ALU，做减法，根据 cf 以及比较的两个数的符号位（a[N-1]、b[N-1]）判断是否交换（确定 en0-3），排序思路为冒泡排序，每轮完成一个数的排序，总共需要比较 6 次；CX 段结束后，进入 HLT 状态，排序完成。

状态图如下：

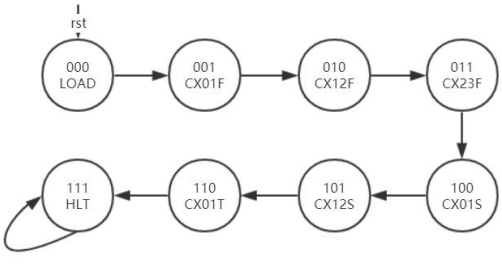


图 5

其中，CX01F 表示 r0，r1 第一次(First time)比较,其余以此类推。

控制单元(输入还包含 clk,在 Logisim 中封装后，并未表示)示意如下：

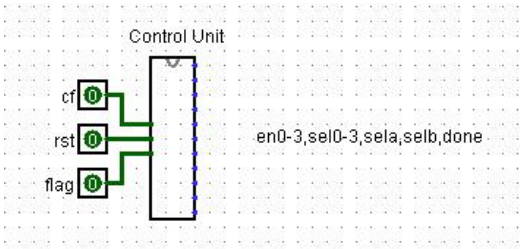


图 6（注：flag=a[N-1]^b[N-1]）

各个状态控制信号如下：

| State | en0 | en1 | en2 | en3 | sel0 | sel1 | sel2 | sel3 | sela | selb | done |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 000(LOAD) | 1 | 1 | 1 | 1 | 00 | 01 | 10 | 11 | x | x | 0 |
| 001(CX01F) | En | En | 0 | 0 | 01 | 00 | x | x | 00 | 01 | 0 |
| 010(CX12F) | 0 | En | En | 0 | x | 10 | 01 | x | 01 | 10 | 0 |
| 011(CX23F) | 0 | 0 | En | En | x | x | 11 | 10 | 10 | 11 | 0 |
| 100(CX01S) | En | En | 0 | 0 | 01 | 00 | x | x | 00 | 01 | 0 |
| 101(CX12S) | 0 | En | En | 0 | x | 10 | 01 | x | 01 | 10 | 0 |
| 110(CX01T) | En | En | 0 | 0 | 01 | 00 | x | x | 00 | 01 | 0 |
| 111(HLT) | 0 | 0 | 0 | 0 | x | x | x | x | x | x | 1 |

表 2 （注：En=~cf^flag）

## 进阶目标

在可选择输出结果为递增或递减时，只需将表 2 中的控制信号中的所有 En 改为(opr^En)即可，状态图和数据通路与基本目标中相同。

控制单元增加了输入 opr，如下(略去 clk 输入)：

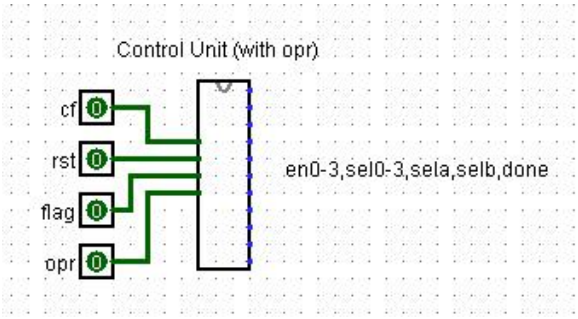


图 7 （注：flag=a[N-1]^b[N-1]）

在使用两个 ALU 的排序器中，可以将 6 个比较周期，缩短至 3 个周期，具体实现方式为：①将 r0 与 r1,r2 与 r3 排好序；②将①中排过序的 r0 与 r3，r1 与 r2 排好序；③将②中排过序的 r0 与 r1,r2 与 r3 排好序。三个周期后，得到的 r0-r3 便为有序输出。

状态图如下：

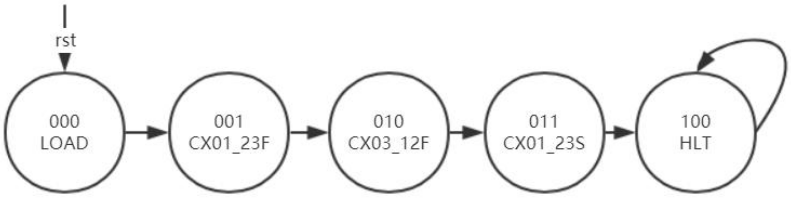


图 8

控制单元如下(略去 clk 输入)：

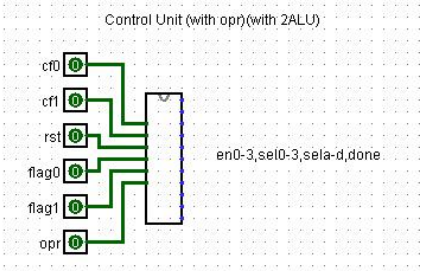


图 9

数据通路，如下：

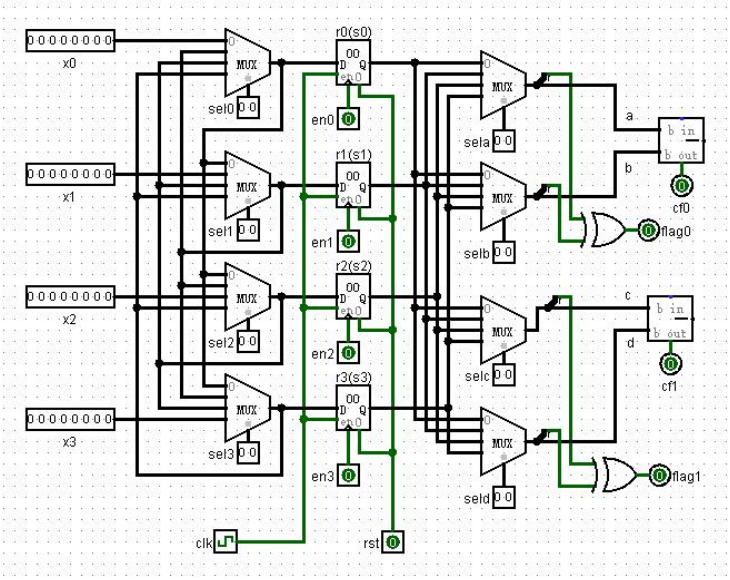


图 10

各个状态控制信号如下：

| State | en0 | en1 | en2 | en3 | sel0 | sel1 | sel2 | sel3 | sela | selb | selc | seld | done |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000(LOAD) | 1 | 1 | 1 | 1 | 00 | 01 | 10 | 11 | x | x | x | x | 0 |
| 001(CX01_23F) | En0 | En0 | En1 | En1 | 01 | 00 | 11 | 10 | 00 | 01 | 10 | 11 | 0 |
| 010(CX03_12F) | En0 | En1 | En1 | En0 | 11 | 10 | 01 | 00 | 00 | 11 | 01 | 10 | 0 |
| 011(CX01_23S) | En0 | En0 | En1 | En1 | 01 | 00 | 11 | 10 | 00 | 01 | 10 | 11 | 0 |
| 100(HLT) | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 1 |

表 3（注：En0=opr^~cf0^flag0,En1=opr^~cf1^flag1）

## 核心代码

### 基本目标

控制单元核心代码如下：

```
always@( * )
  begin
    { en0, en1, en2, en3, done } = 5'b0;
    flag = a[ N - 1 ] ^ b[ N - 1 ];
    case ( current_state )
      LOAD:
        begin
          { sel0, sel1, sel2, sel3 } = 8'b00011011;
          { en0, en1, en2, en3 } = 4'b1111;
        end
      CX01F, CX01S, CX01T:
        begin
          { sela, selb } = 4'b0001;
          { sel0, sel1 } = 4'b0100;
          en0 = ~cf ^ flag;
          en1 = ~cf ^ flag;
        end
      CX12F, CX12S:
        begin
          { sela, selb } = 4'b0110;
          { sel1, sel2 } = 4'b1001;
          en1 = ~cf ^ flag;
          en2 = ~cf ^ flag;
        end
      CX23F:
        begin
          { sela, selb } = 4'b1011;
          { sel2, sel3 } = 4'b1110;
          en2 = ~cf ^ flag;
          en3 = ~cf ^ flag;
        end
      HLT:
        done = 1;
    endcase
  end
```

### 进阶目标

增加 opr(可选输出递增/递减)后的核心代码，只需将基本目标中的~cf 改为 opr^~cf 即可。

使用 2 个 ALU 的排序器的核心代码，如下：

```
always@( * )
  begin
    { en0, en1, en2, en3, done } = 5'b0;
    flag0 = a[ N - 1 ] ^ b[ N - 1 ];
    flag1 = c[ N - 1 ] ^ d[ N - 1 ];
    case ( current_state )
      LOAD:
        begin
          { sel0, sel1, sel2, sel3 } = 8'b00011011;
          { en0, en1, en2, en3 } = 4'b1111;
        end
      CX01_23F, CX01_23S:
        begin
          { sela, selb , selc, seld } = 8'b00011011;
          { sel0, sel1 , sel2, sel3 } = 8'b01001110;
          en0 = opr ^ ~cf0 ^ flag0;
          en1 = opr ^ ~cf0 ^ flag0;
          en2 = opr ^ ~cf1 ^ flag1;
          en3 = opr ^ ~cf1 ^ flag1;
        end
      CX03_12F:
        begin
          { sela, selb , selc, seld } = 8'b00110110;
          { sel0, sel1 , sel2, sel3 } = 8'b11100100;
          en0 = opr ^ ~cf0 ^ flag0;
          en1 = opr ^ ~cf1 ^ flag1;
          en2 = opr ^ ~cf1 ^ flag1;
          en3 = opr ^ ~cf0 ^ flag0;
        end
      HLT:
        done = 1;
    endcase
  end
```

## 仿真结果

### 基本目标

仿真结果如下：

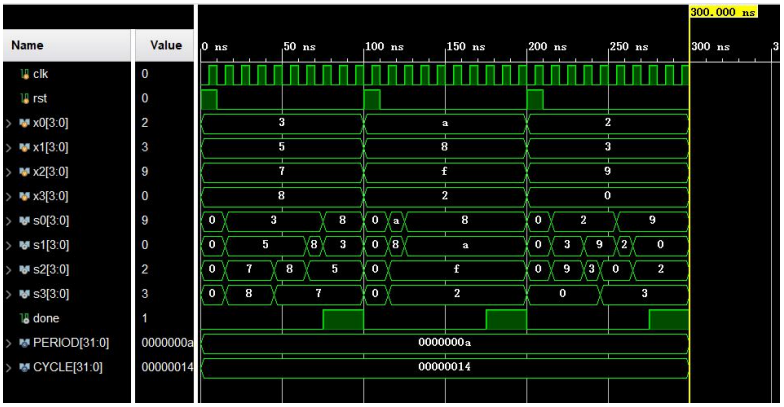


图 11

## 进阶目标

可选择输出结果为递增或递减的排序器，仿真结果如下：

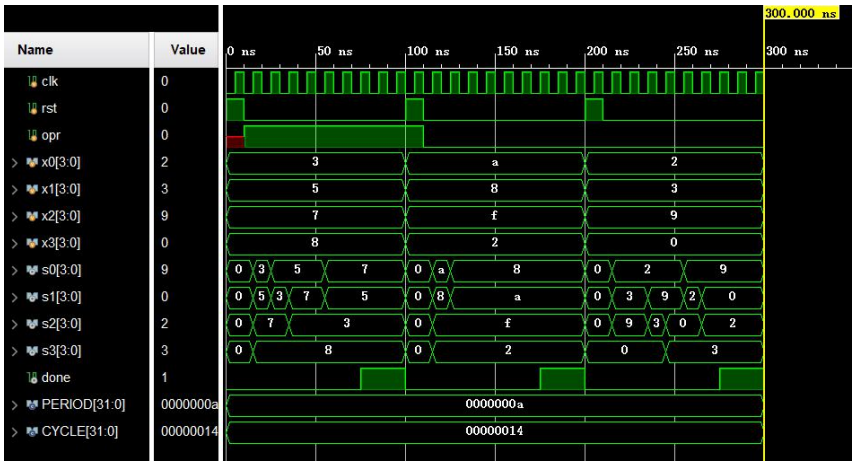


图 12

使用 2 个 ALU 的排序器，仿真结果如下：

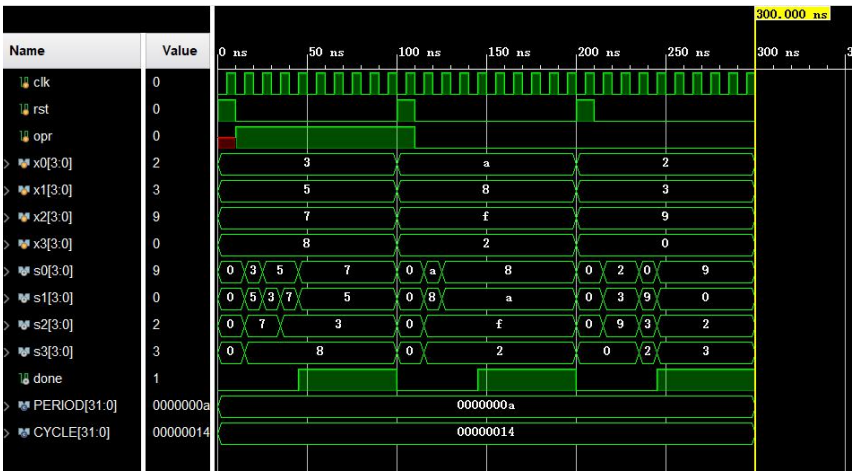


图 13

## 结果分析

ALU 及排序器的仿真结果均符合设计目标。在使用 2 个 ALU 的设计方案中，也充分利用了提供的器件，使得比较阶段由 6 个时钟周期缩短至 3 个时钟周期，达到了空间换时间的目的。

## 实验总结

本次实验复习了上学期所学的 Verilog 的基本知识，并通过 ALU 及排序器，实现了对组合逻辑电路和时序逻辑电路的巩固，为后半学期的 CPU 设计打下基础。

## 意见建议

建议缩短实验讲解环节的时间。

## 思考题

已在实验报告中的进阶目标中体现。