CODH - 运算器及其应用 实验报告

院系: 姓名: 学号:

2023 年 7 月 13 日

1 实验目的

- 熟练掌握算术逻辑单元 (ALU) 的功能
- 掌握数据通路和控制器的设计方法
- 掌握组合电路和时序电路，以及参数化和结构化的 Verilog 描述方法
- 了解查看电路性能和资源使用情况

2 实验环境

- vlab.ustc.edu.cn
- Vivado 2016.3
- Nexys 4 DDR 开发板

3 实验内容

3.1 实现算术逻辑单元 (ALU)

- f：功能选择，加、减、与、或、异或、逻辑左移、逻辑右移、算术右移等运算
- a, b：两个操作数
- y：运算结果，和、差……
- t：比较标志，相等 (eq)，小于 (lt, ltu)

3.2 ALU 的应用：计算移动/滑动平均 (MAV)

- d：输入数据
- en：输入使能
- m：输出数据
- clk, rstn：时钟，复位信号

4 逻辑设计 / 核心代码

4.1 算术逻辑单元 (ALU)

4.1.1 逻辑设计

只需要使用 Verilog 基本运算以及组合逻辑即可实现，注意 `f=3'b000`（减法计算）时需要对 `t` 进行特殊计算。（相等时 `t[0] = 1`，无符号小于时 `t[2] = 1`，有符号小于时 `t[1] = 1`）。

4.1.2 核心代码

```
always @ (*)
begin
  t = 3'b000;
  case(f)
  3'b000:
  //t[0] = 1 if a eq b
  //t[1] = 1 if a < b (signed)
  //t[2] = 1 if a < b (unsigned)
  begin
      y = a - b;
      if(y == 0)
          t[0] = 1;
      if(a < b)
          t[2] = 1;
      if($signed(a) < $signed(b))
          t[1] = 1;
  end
  3'b001: y = a + b;
  3'b010: y = a & b;
  3'b011: y = a | b;
  3'b100: y = a ^ b;
  3'b101: y = a >> b;
  3'b110: y = a << b;
  3'b111: y = $signed(a) >>> b;
  endcase
end
```

alu.v

4.1.3 模块仿真

直接对默认的 32 位 ALU 进行仿真，使用系统的 `$random` 函数生成随机的 a 和 b，对每一种 f 进行仿真。testbench 如下:

```
...
alu alu_inst(a, b, s, y, f);
initial
begin
    a = $random;
    b = $random;
    s = 0;
end

initial
begin
```

```
12          for(i=0; i<8; i=i+1)
13          begin
14              #5 s = s + 1;
15          end
16      $finish;
17      end
```

alu.v

在 Vivado 仿真后得到波形图：

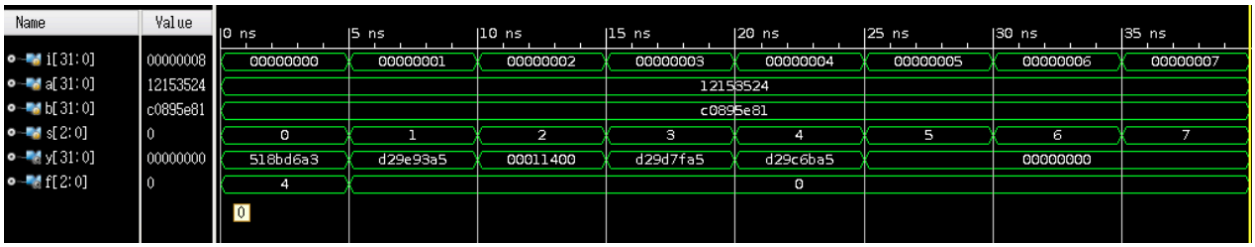

图 1: ALU 仿真波形图

注意到最后有关移位的部分，因为 b 过大导致结果都为 0，实际上正确性没有问题。如果可以对 b 进行限制（取模），可能结果会更加直观。

4.1.4 下载测试

由于一开始无法烧写板子，故首先在 FPGAOL 平台上进行了调试，之后编写 alu_top 进行测试，将 ALU 的输入输出接到开发板上，使用开发板上的按键和 LED 来进行 ALU 的输入和输出，结果与设计相符合。因为仅仅是一个组合逻辑电路，没有加入时钟，因此没有时间性能报告。

RTL 电路图：

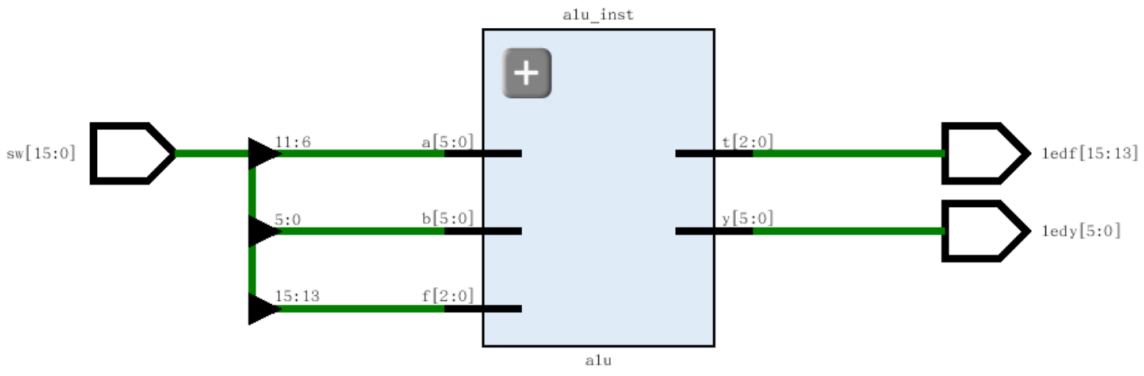

图 2: ALU RTL 电路图

电路资源使用情况：

Hierarchy

Name	Slice LUTs (63400)	F7 Muxes (31700)	Bonded IOB (210)
alu_top	60	4	24
alu_inst (alu)	17	3	0

图 3: ALU 电路资源使用情况

4.2 计算移动/滑动平均 (MAV)

4.2.1 逻辑设计

状态机方面，共设计五个状态，需要 3 位二进制码表示：未输入数字（IDLE）、已输入一个数字（M1）、已输入两个数字（M2）、已输入三个数字（M3）、已输入四个数字（M4）。使用四个寄存器（m0、m1、m2、m3）来存储之前输入的四个数字。

初始状态记为 IDLE，此时初始化 m0、m1、m2、m3 为 0，输出数据（m）为 0。

对于每个状态，当时针上升沿到来时，如果使能信号（EN）为 1，则将寄存器内容向后移动，将输入数据（d）存入第一个寄存器（m0）；如果使能信号为 0，则不改变数据寄存器。

对于前四个状态，输出数据（m）为之前的输入数据（d），当时针上升沿到来时，如果使能信号为 1，则切换至下一个状态，否则保持当前状态。

最后一个状态输出的是对四个寄存器内数据的平均值，借用 ALU，可以实现累计，并且可以通过向右位移 2 位来实现除以 4 的效果。

状态转换图如下：

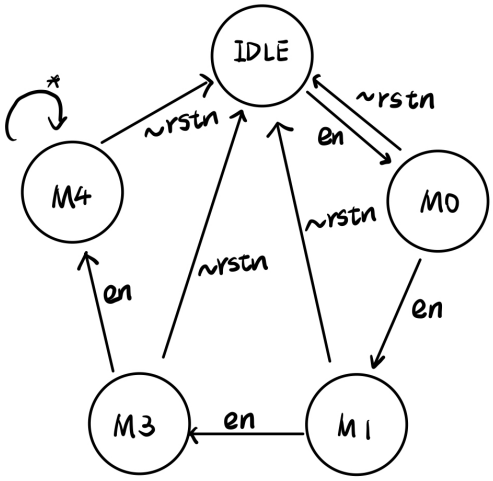

图 4: MAV 状态转换图

4.2.2 核心代码

```
    ...
    //三段式状态机
    //此处四个例化ALU用于计算(m0+m1+m2+m3)/4
    alu #(.WIDTH(16)) alu_inst1(d, m0, 3'b001, temp1);
    alu #(.WIDTH(16)) alu_inst2(m1, m2, 3'b001, temp2);
    alu #(.WIDTH(16)) alu_inst3(temp1, temp2, 3'b001, temp3);
    alu #(.WIDTH(16)) alu_inst4(temp3, 16'o2, 3'b111, res);
    //状态切换
    always@(posedge clk or negedge rstn) begin
        if(~rstn)
            cs <= 3'b000;
        else
            cs <= ns;
    end
    //决定次态
    always@(*) begin
        ns = cs;
        if(en) begin
            case(cs)
                IDLE: begin ns = M1; end
                M1: begin ns = M2; end
                M2: begin ns = M3; end
                M3: begin ns = M4; end
                M4: begin ns = M4; end
                 default begin ns = IDLE; end
        endcase
        end
    end
    //输出部分
    always@(posedge clk or negedge rstn) begin
        if(~rstn) begin
                m0 <= 16'b0;
                m1 <= 16'b0;
                m2 <= 16'b0;
                m3 <= 16'b0;
                m <= 16'b0;
        end
        else if(en) begin
            m0 <= d;
            m1 <= m0;
            m2 <= m1;
            m3 <= m2;
             if(ns == 3'b100) begin
                m <= res;
            end
             else
                m <= d;
        end
    end
```

mav.v

4.2.3 模块仿真

对照老师提供的波形图编写 testbench:

```
  ...
  mav mav_inst(clk, rstn, en, d, m);
```

```
3
4   initial
5   begin
6       rstn = 0;
7       #7 rstn = ~rstn;
8   end
9
10  initial
11  begin
12      clk = 0;
13      forever
14          #5 clk = ~clk;
15          #5 clk = ~clk;
16          en = ~en;
17  end
18  initial
19      begin
20          en = 0;
21          forever
22              #10 en = ~en;
23      end
24  initial
25  begin
26      d = 31'o2;
27      #20 d = 31'o3;
28      #20 d = 31'o4;
29      #20 d = 31'o5;
30      #20 d = 31'o6;
31      #30 $finish;
32  end
```

mav.v

在 Vivado 进行仿真，波形图如下：

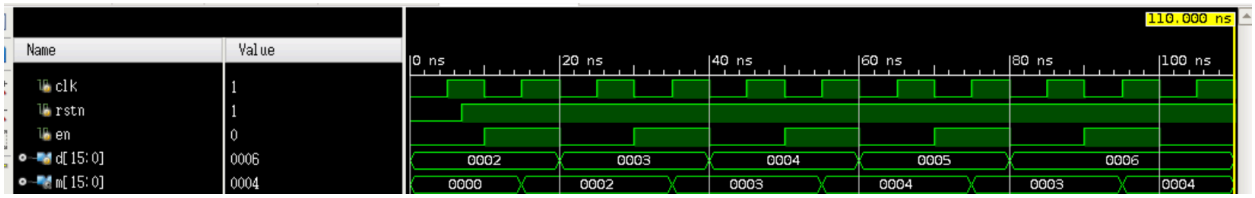

图 5: MAV 仿真波形图

可以发现 MAV 模块很好地完成了任务。

4.2.4 下载测试

首先需要编写取边沿，去抖动的模块，对 en 进行处理：

```
1   ...
2   always@(posedge clk)
3     begin
4         if (en == 0)
5             cnt <= 0;
6         else if (cnt < 16'h8000)
7             cnt <= cnt + 1;
8         en1 <= cnt[15];
```

```
        en2 <= en1;
    end
  assign out = en1 & ~en2;
```

edge.v

然后编写 mav_top 模块，同时为了更直观的看到结果，除了用 16 位 LED 作为输出之外还添加了 4 位 7 短管来显示 4 位 16 进制的输出。这里实例化 mav 模块，使用了 edge 模块来处理使能信号，encode 模块来将输出数据 m 的每四位转化为 7 段管的显示码，使用了分时轮流显示 4 位 7 段管来达到同时显示的效果。

```
module mav_top (
    input  clk,
    input  rstn,
    input  en,
    input  [15:0]  d,
    output [15:0]  m,
    output [7:0] AN,
    output CA, CB, CC, CD, CE, CF, CG
);
    wire out;
    integer t = 0;
    reg [7:0] in_temp, AN_temp;
    fix_edge edge_inst1(clk, en, out);
    mav mav_inst1(clk, rstn, out, d, m);

    encode encode_inst(
        .in(in_temp),
        .out1(CA),
        .out2(CB),
        .out3(CC),
        .out4(CD),
        .out5(CE),
        .out6(CF),
        .out7(CG)
    );

    always @ (posedge clk)
    begin
        if(t % 100000 == 0)
        begin
            t = 0;
            case(AN_temp)
                8'b11111110: begin AN_temp = 8'b11111101; in_temp = m[7:4]; end
                8'b11111101: begin AN_temp = 8'b11111011; in_temp = m[11:8]; end
                8'b11111011: begin AN_temp = 8'b11110111; in_temp = m[15:12]; end
                default: begin AN_temp = 8'b11111110; in_temp = m[3:0]; end
           endcase
        end
        t = t + 1;
    end
    assign AN = AN_temp;
endmodule
```

mav_top.v

七段管编码显示，encode 模块：

```
    ...
```

```
    case (in)
        4'b0000: begin out = 7'b1000000; end
        4'b0001: begin out = 7'b1111001; end
        4'b0010: begin out = 7'b0100100; end
        4'b0011: begin out = 7'b0110000; end
        4'b0100: begin out = 7'b0011001; end
        4'b0101: begin out = 7'b0010010; end
        4'b0110: begin out = 7'b0000010; end
        4'b0111: begin out = 7'b1111000; end
        4'b1000: begin out = 7'b0000000; end
        4'b1001: begin out = 7'b0010000; end
        4'b1010: begin out = 7'b0001000; end
        4'b1011: begin out = 7'b0000011; end
        4'b1100: begin out = 7'b1000110; end
        4'b1101: begin out = 7'b0100001; end
        4'b1110: begin out = 7'b0000110; end
        4'b1111: begin out = 7'b0001110; end
        default: begin out = 7'b1111111; end
        endcase
    end
    assign {out7, out6, out5, out4, out3, out2, out1} = {out};
    ...
```

encode.v

然后生成 bit 流进行测试。由于笔记本不兼容，一开始先在 FPGAOL 平台上对简化版的 mav_top 进行了测试，测试结果一切正常。

之后再烧写到 Nexys 4 DDR 上进行测试，测试结果与预期相符，已供助教检查。助教同时指出了一些可以简化的部分，因此检查结束后又改进了部分代码（相较于检查时添加了 7 段数码管显示、简化了状态机的转换）。

4.2.5 结果分析

RTL 电路：寄存器转换级电路，根据硬件寄存器之间的数字信号 (数据) 流以及对这些信号执行的逻辑操作来模拟同步数字电路。

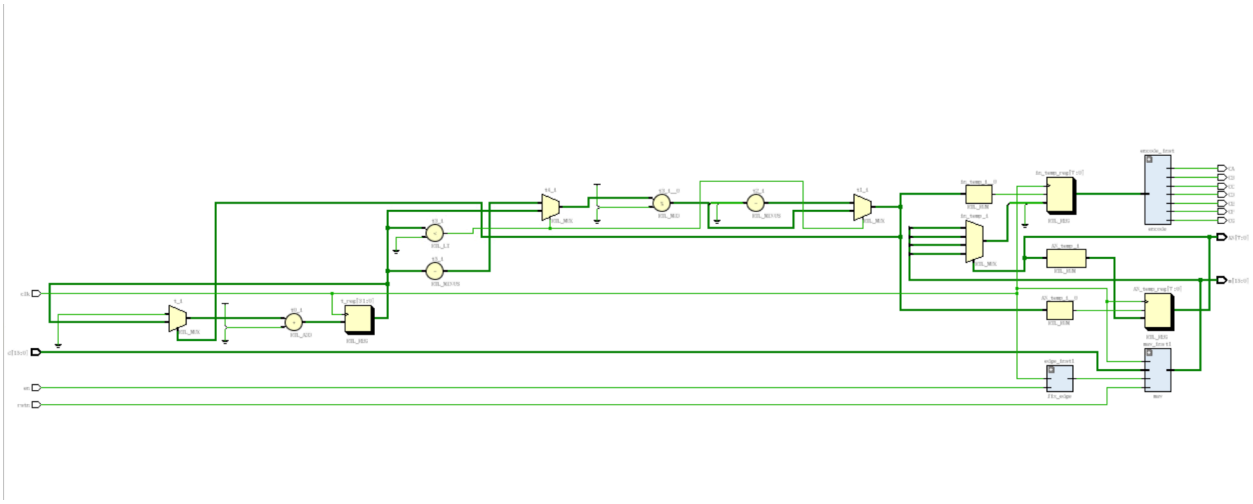

图 6: RTL 电路图

可以发现由于 mav_top 对 7 段数码管输出的支持使得 RTL 电路图变得很复杂，删去这一部分后

再查看：

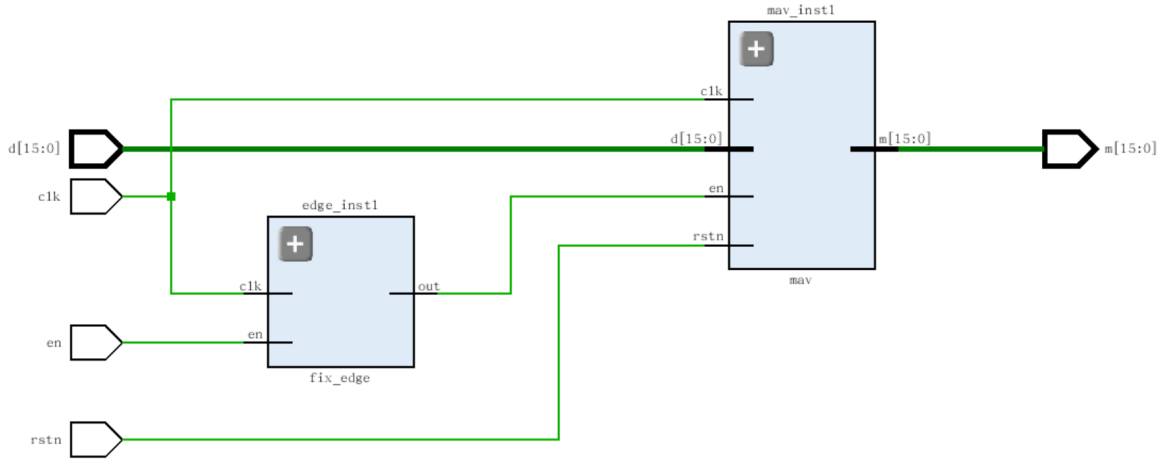

图 7: RTL 电路图

电路资源使用情况：

Name	Slice LUTs (63400)	Slice Registers (126800)	Bonded IOB (210)	BUFGCTRL (32)
mav_top	510	125	50	1
edge_inst1 (fix_edge)	19	18	0	0
encode_inst (encode)	4	0	0	0
mav_inst1 (mav)	71	67	0	0

图 8: 电路资源使用情况

时间性能报告：

Intra-Clock Paths - sys_clk - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	5.217	8	3	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[13]/D	4.647	3.237	1.410	10.000	sys_clk	sys_clk	
Path 2	5.217	8	3	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[14]/D	4.647	3.237	1.410	10.000	sys_clk	sys_clk	
Path 3	5.217	8	3	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[15]/D	4.647	3.237	1.410	10.000	sys_clk	sys_clk	
Path 4	5.312	8	2	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[12]/D	4.552	3.156	1.396	10.000	sys_clk	sys_clk	
Path 5	5.352	7	2	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[9]/D	4.512	3.120	1.392	10.000	sys_clk	sys_clk	
Path 6	5.366	8	2	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[11]/D	4.498	3.242	1.256	10.000	sys_clk	sys_clk	
Path 7	5.429	7	2	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[8]/D	4.435	3.039	1.396	10.000	sys_clk	sys_clk	
Path 8	5.483	7	2	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[7]/D	4.381	3.125	1.256	10.000	sys_clk	sys_clk	
Path 9	5.483	8	2	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[10]/D	4.381	3.126	1.255	10.000	sys_clk	sys_clk	
Path 10	5.573	6	2	mav_inst1/m0_reg[1]/C	mav_inst1/m_reg[5]/D	4.291	2.899	1.392	10.000	sys_clk	sys_clk	

图 9: 时间性能报告

5 结果分析

每个 Xilinx 7 系列 FPGA 的 Slice 包含 4 个 LUT 查找表和 8 个触发器，还可以包含寄存器、进位链和多个多数选择器。

关于电路资源使用情况：这里前两列显示的就是模块占用的 LUT 以及 Register 的数目；Bonded IOB 指占用的可编程输入输出单元数目，BUFGCTRL 指占用的时钟缓冲器数目。

关于时间性能报告：Timing Summary 报告把路径按照时钟域（例如 Intra-Clock，同一时钟域内部路径）分类，每个组别下缺省会报告 Setup、Hold 以及 Pulse Width 检查最差的各 10 条路径，其中一些参数意义如下：

1. slack：建立时间裕量
2. level：逻辑级数，这里 1 就表示在两个寄存器之间仅存在 1 个组合逻辑器件
3. fanout：表示从这一点连接到了几个目的端点，fanout = 1 就表示连接了 1 个目的端点
4. from to：表示是哪两者之间的时序
5. logic delay：逻辑器件延时
6. net delay：布线延时

6 实验总结

1. 本次实验尽管在开始时由于我的硬件条件不支持对烧录程序造成了一些麻烦，但依然很好地锻炼了我设计状态图，从底向上分模块构建代码完成试验任务的能力；同时也磨练了调试代码的技术。
2. 加深了对开发板的理解，意识到了仿真的重要性；完善的仿真模拟可以减少很多的硬件调试，效率会高上不少。
3. 学会了更好地设计状态机，在助教的指导下检查出状态间重复可以合并的部分，使得代码更加简洁。
4. 了解到了如何使用 7 段数码管显示数字，使得输出更加直观。
5. 学会了查看 RTL 电路，电路资源使用情况，时间性能报告等。并且通过搜索引擎了解到了这些后续分析的意义所在。

7 意见/建议

如果可能的话，建议增加实验室开放时间。由于硬件条件不兼容，第一次实验可能耗费了许多时间在无用的尝试之上，如果平时也能够在实验室进行实验，那么效率会高上不少。