Lab 1 运算器及其应用 报告

PB21111681 朱炜荣

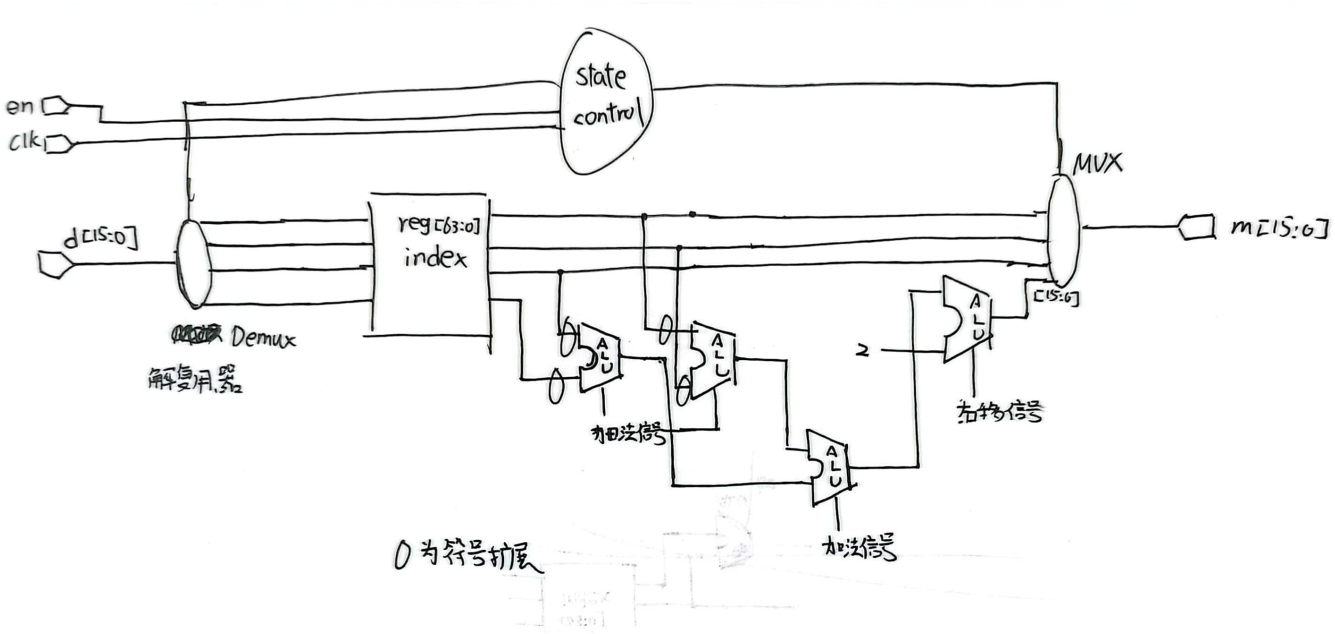
实验目的与内容

本次实验的主要目的在于帮助大家回忆 Verilog 的相关语法、Verilog 的编程结构和技巧。同时给第一次接触到板子的同学一个熟悉使用板子的机会。掌握数据通路和控制器的设计方法。

实验的内容主要包括自己编写一个 ALU（算术逻辑单元），并进行仿真测试；完成 MAV 的逻辑设计、仿真和下载测试，并查看 RTL 电路图，以及实现电路资源和时间性能报告。

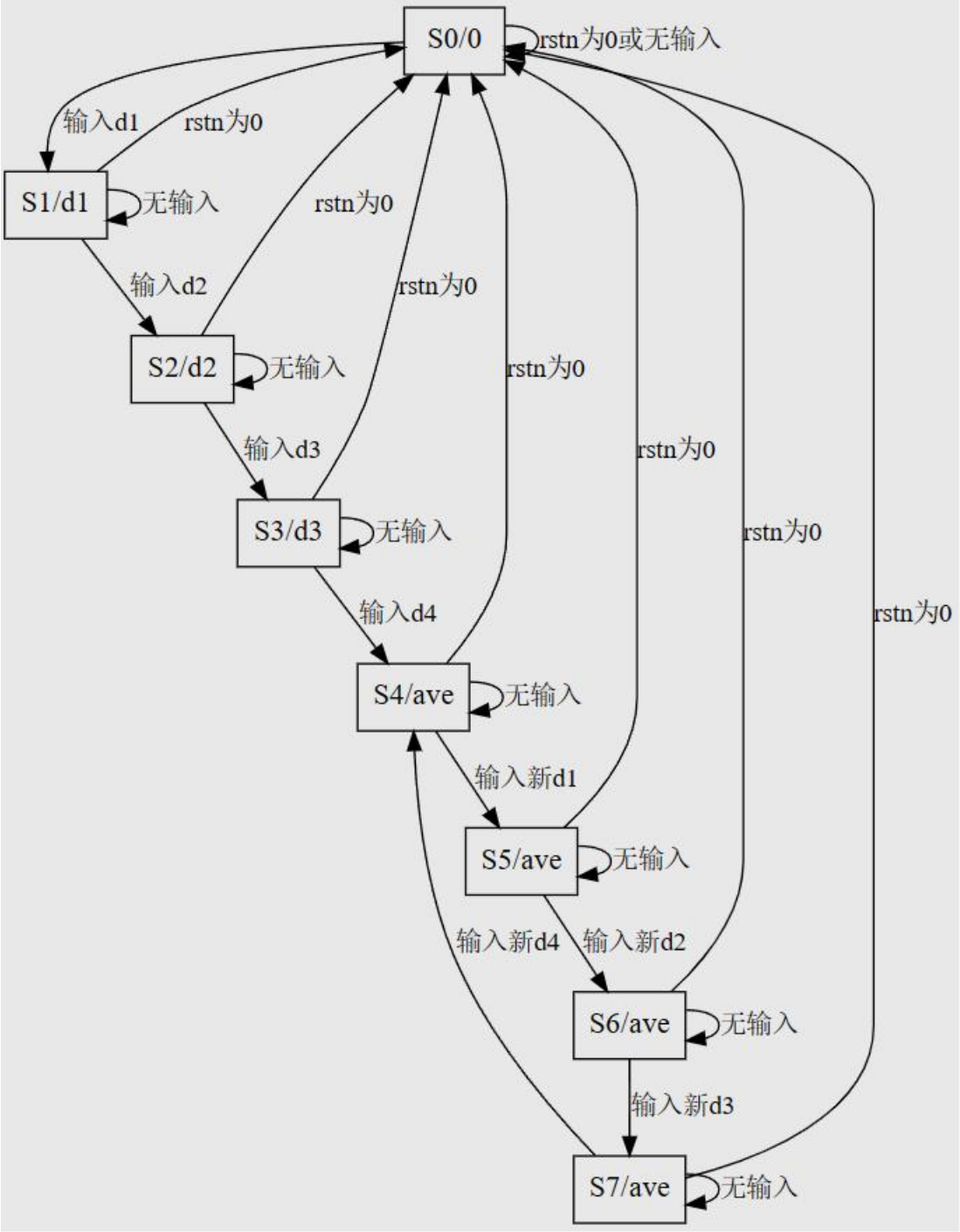
逻辑设计

模块的框图和数据通路如下所示：



其中 state_control 是一个由时钟和 en 控制的时序逻辑程序，用来控制输入和输出的数据选择，index 为 64 位的寄存器用来记录每次读入的数据值。选择用四个 ALU 来完成程序是因为题目中要求了计算在单周期内完成。

程序实现的状态转换图如下：



其中较为核心的代码为两段式摩尔型状态机的 Verilog 实现。

```
always @(posedge clk,posedge rstn_clear) begin
    if(!rstn_clear) begin
        current_state <= S0;
    end
    else begin
        current_state <= next_state;
```

```
    end
end

always@(*) begin
     next_state = current_state;
     if(!rstn_clear) begin
        index = 64'b0;
        m0 = 16'b0;
        next_state = S0;
    end
    else
    case (current_state)
        S0:begin    //复位设置数值
            if(en_clear) begin
                m0 = d;
                index[63:48] = d;
                next_state = S1;
           end
           else begin
                m0 = 16'b0;
                next_state = S0;
            end
        end
        S1:begin    //第一次输入成功
            if(en_clear)begin
                m0 = d;
                index[47:32] = d;
                next_state = S2;
            end
            else begin
                m0 = index[63:48];
                index = index;
                next_state = S1;
            end
        end
        S2:begin    //第二次输入成功
            // next_state = current_state;
            // index = {index[63:32],32'b0};
            if(en_clear)begin
                m0 = d;
               index[31:16] = d;
               next_state = S3;
            end
           else begin
                m0 = index[47:32];
                index = index;
                next_state = S2;
            end
```

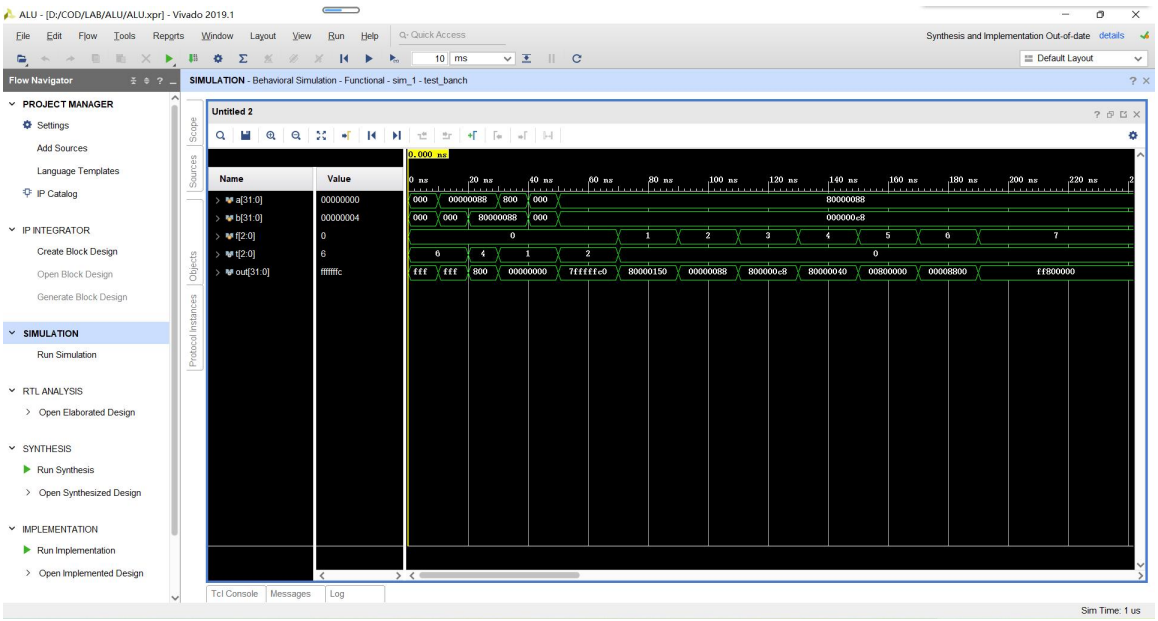
```
        end
        S3:begin
            // next_state = current_state;
            // index = {index[63:16],16'b0};
            if(en_clear) begin
                m0 = d;
                index[15:0] = d;
                next_state = S4;
            end
            else begin
                m0 = index[31:16];
                index = index;
                next_state = S3;
            end
        end
        S4:begin
            if(en_clear) begin
                index[63:48] = d;
                m0 = out[15:0];
                next_state = S5;
            end
            else begin
                m0 = out[15:0];
                index = index;
                next_state = S4;
            end
        end
        S5: begin
            if(en_clear) begin
                index[47:32] = d;
                m0 = out[15:0];
                next_state = S6;
            end
            else begin
                m0 = out[15:0];
                index = index;
                next_state = S5;
            end

        end
        S6: begin
            if(en_clear) begin
                index[31:16] = d;
                m0 = out[15:0];
                next_state = S7;
            end
             else begin
                next_state = S6;
```

```
                index = index;
                m0 = out[15:0];
            end
        end
        S7: begin
            if(en_clear) begin
                index[15:0] = d;
                m0 = out[15:0];
                next_state = S4;
            end
             else begin
                m0 = out[15:0];
                index = index;
                next_state = S7;
            end
        end
    endcase
end
```

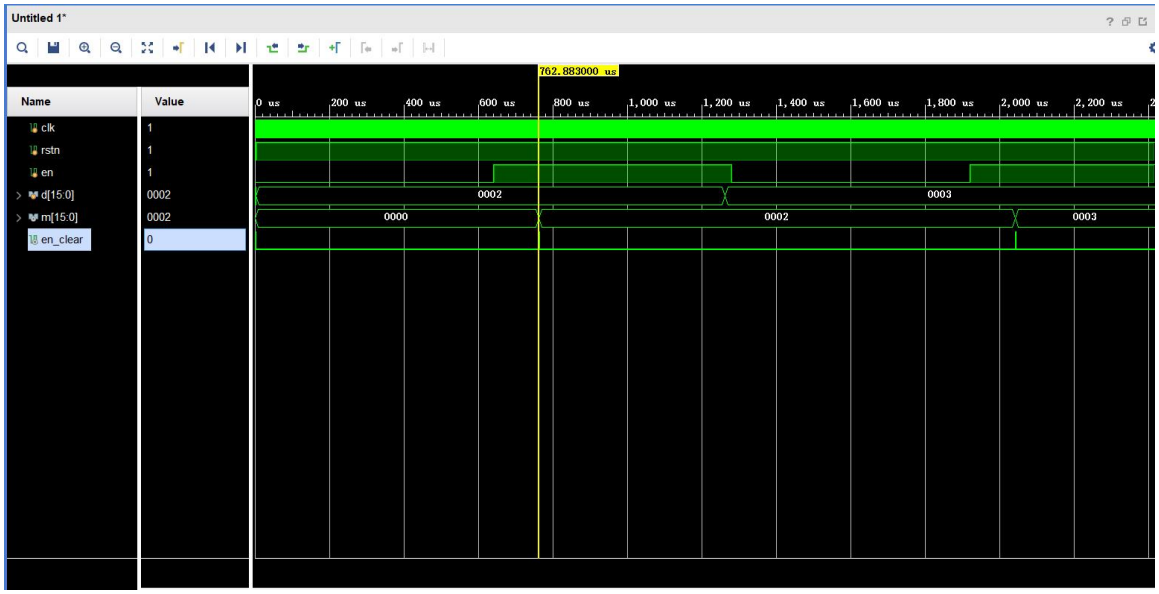
两段式的有限状态机实现中只使用两个 always 模块，一个用来处理状态转移的时序逻辑，即处理状态的变化和复位设值；另一个用来描述计算和赋值的组合逻辑，主要包括将读入数据保存起来，并在合适的时候对 4 个 16 位数进行求和取平均值，同时根据输入信号的变化来判断下一个周期的状态应该是什么。

仿真结果与分析

以下为实验两个部分的仿真结果：



ALU 仿真 1



MAV 仿真 1

ALU 的仿真文件代码：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2023/03/30 17:22:24
// Design Name:
// Module Name: test_banch
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


```

```
23.  module test_banch();
24.
25.  reg [31:0] a = 32'b0;
26.  reg [31:0] b = 32'b100;
27.  reg [2:0] f = 3'b000;
28.  wire [2:0]t;
29.  wire [31:0]out;
30.
31.  ALU test(.a(a),.b(b),.f(f),.t(t),.y(out));
32.
33.  initial begin
34.      #10 a = 32'b0000_0000_0000_0000_0000_0000_1000_1000; b = 32'b0000_0000_0000_0000_0000_0000_1100_1000; f = 3'b000;
35.      #10 a = 32'b0000_0000_0000_0000_0000_0000_1000_1000; b = 32'b1000_0000_0000_0000_0000_0000_1000_1000; f = 3'b000;
36.      #10 a = 32'b1000_0000_0000_0000_0000_0000_1000_1000; b = 32'b1000_0000_0000_0000_0000_0000_1000_1000; f = 3'b000;
37.      #10 a = 32'b0000_0000_0000_0000_0000_0000_1000_1000; b = 32'b0000_0000_0000_0000_0000_0000_1000_1000; f = 3'b000;
38.      #10 a = 32'b1000_0000_0000_0000_0000_0000_1000_1000; b = 32'b0000_0000_0000_0000_0000_0000_1100_1000; f = 3'b000;
39.      #20 f = 3'b001;
40.      #20 f = 3'b010;
41.      #20 f = 3'b011;
42.      #20 f = 3'b100;
43.      #20 f = 3'b101;
44.      #20 f = 3'b110;
45.      #20 f = 3'b111;
46.
47.  end
48.  endmodule
```

ALU 的仿真文件较为简单，因为自身的逻辑为简单的组合逻辑，设置数据时只需要遍历各种可能的情况即可，问题主要出在 Verilog 会默认一个数据的类型为无符号数，为了简单的实现有符号数的位移，所以要在输入时进行数据类型转换。以及在做减法中对于各类数据数据的遍历也需要花一些心思。

MAV 的仿真文件代码：

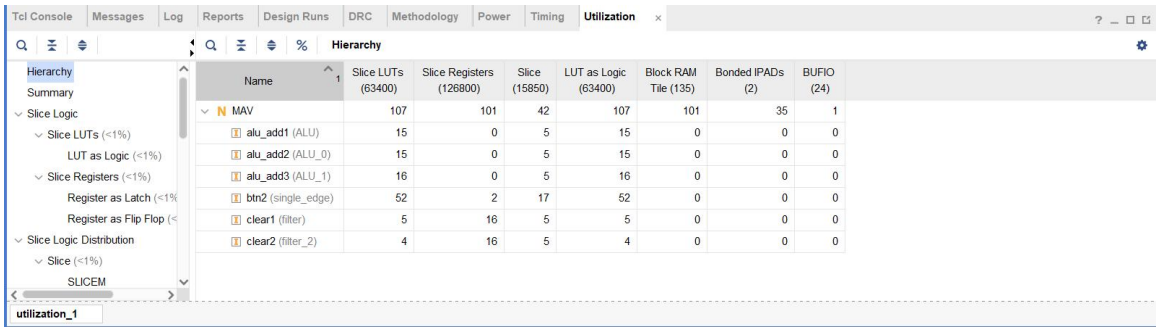
```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
/
3.  // Company:
4.  // Engineer:
5.  //
6.  // Create Date: 2023/04/01 15:13:27
7.  // Design Name:
8.  // Module Name: test_banch
9.  // Project Name:
```

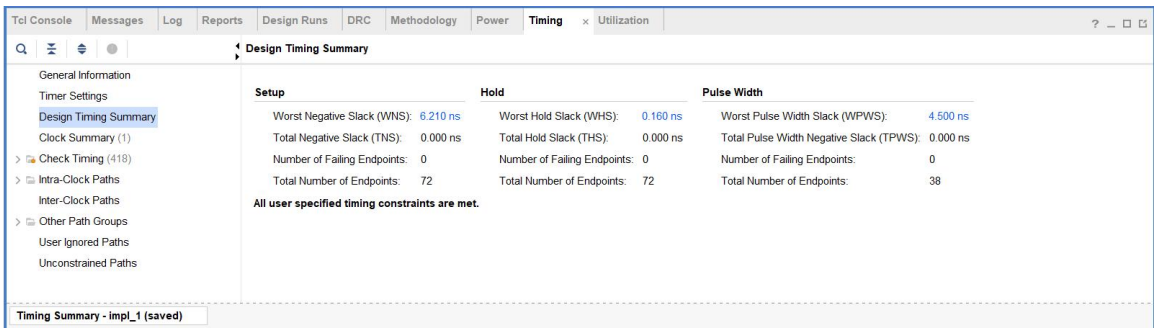
```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/


module test_banch();
reg clk,rstn,en;
reg [15:0] d;
wire [15:0] m;
MAV mav(.clk(clk),.rstn(rstn),.en(en),.d(d),.m(m));
initial begin
    d = 16'hffff;
    clk = 0;
    en = 0;
    rstn = 0;
end
always #1 clk = ~clk;
always #640000 en = ~en;

initial
begin
    #5 rstn = 1;
    #2000 d = 16'b0000_0000_0000_0010;
   #630000 d = 16'b0000_0000_0000_0010;
    #630000 d = 16'b0000_0000_0000_0011;
    #630000 d = 16'b0000_0000_0000_0011;
    #630000 d = 16'b0000_0000_0000_0101;
    #630000 d = 16'b0000_0000_0000_0010;
    #630000 d = 16'b000_0000_0000_0001;
    #630000 d = 16'b0000_0000_0000_0100;
    #630000 d = 16'b0000_0000_0000_0100;
    #630000 d = 16'b0000_0000_0000_0101;
    #630000 d = 16'b0000_0000_0000_0010;
    #630000 d = 16'b0000_0000_0000_0010;
    #630000 d = 16'b0000_0000_0000_0011;
    #630000 d = 16'b0000_0000_0000_0011;
    #630000 d = 16'b0000_0000_0000_0101;
    #630000 d = 16'b0000_0000_0000_0010;
    #630000 d = 16'b000_0000_0000_0001;
```

```
57.       #630000 d = 16'b0000_0000_0000_0100;
58.       #630000 d = 16'b0000_0000_0000_0100;
59.       #630000 d = 16'b0000_0000_0000_0101;
60.       #630000 d = 16'b0000_0000_0000_0100;
61.       #630000 d = 16'b0000_0000_0000_0100;
62.       #630000 d = 16'b0000_0000_0000_0101;
63.       #630000 d = 16'b0000_0000_0000_0010;
64.       #630000 d = 16'b0000_0000_0000_0010;
65.       #630000 d = 16'b0000_0000_0000_0011;
66.       #630000 d = 16'b0000_0000_0000_0011;
67.       #630000 d = 16'b0000_0000_0000_0101;
68.       #630000 d = 16'b0000_0000_0000_0010;
69.       #630000 d = 16'b000_0000_0000_0001;
70.       #630000 d = 16'b0000_0000_0000_0100;
71.       #630000 d = 16'b0000_0000_0000_0100;
72.       #630000 d = 16'b0000_0000_0000_0101;
73.   end
74.
75.
76.   endmodule
```

MAV 的仿真文件中每一个信号所需的时间都很长，这是因为自己的封装不当，在 MAV 这个模块中试图解决所有问题，于是需要对输入的信号进行取毛刺取边沿，而去毛刺如果持续时间很短的话就会有一次跳过两个状态的情况出现，这不是我们想看到的情况，于是在反复揣摩实验之下选择了仿真代码中的数据。

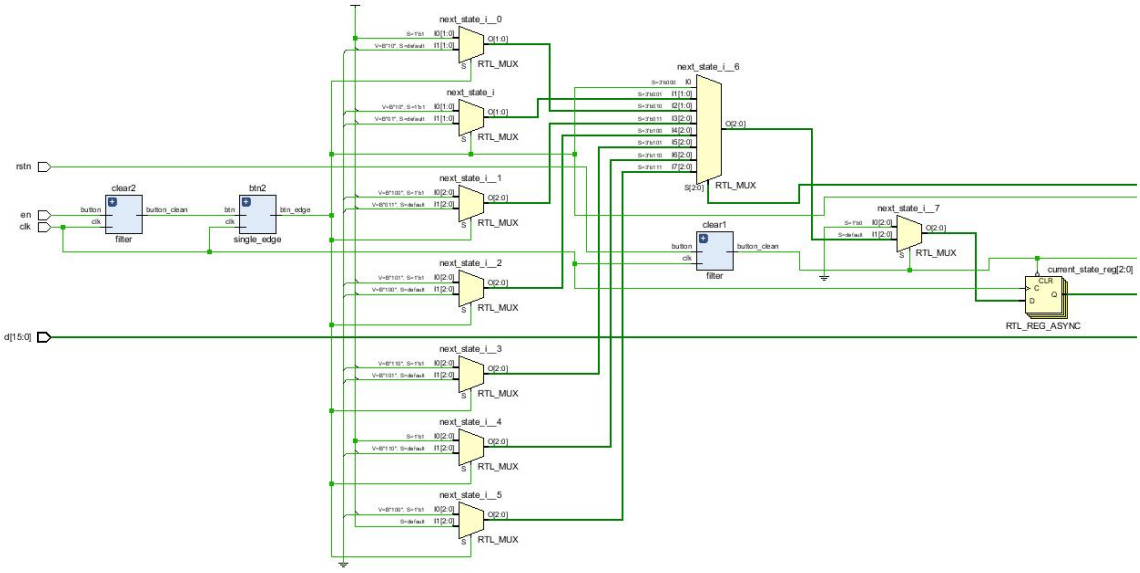
电路资源和时间性能

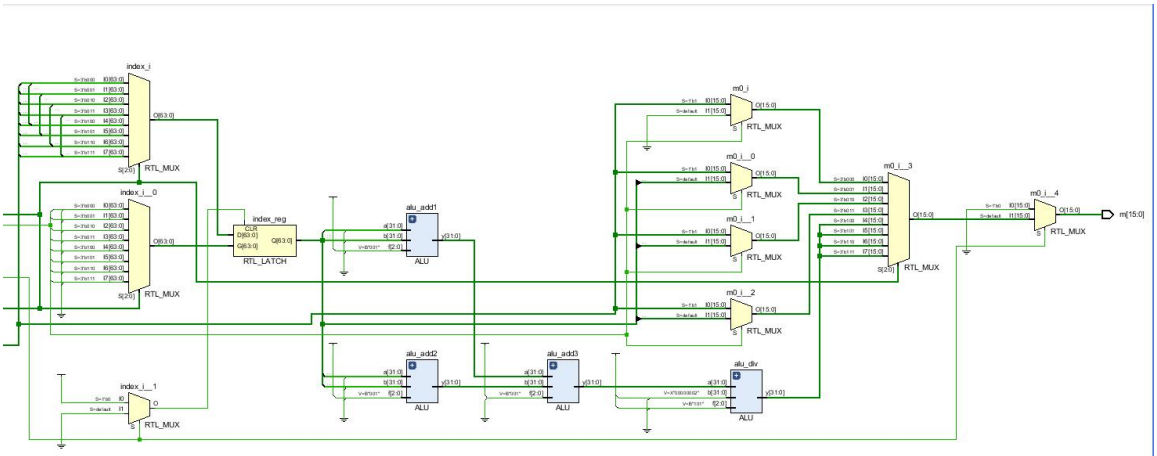




电路设计与分析

程序完整的 RTL 电路图如下：





其中模块的作用分别是：

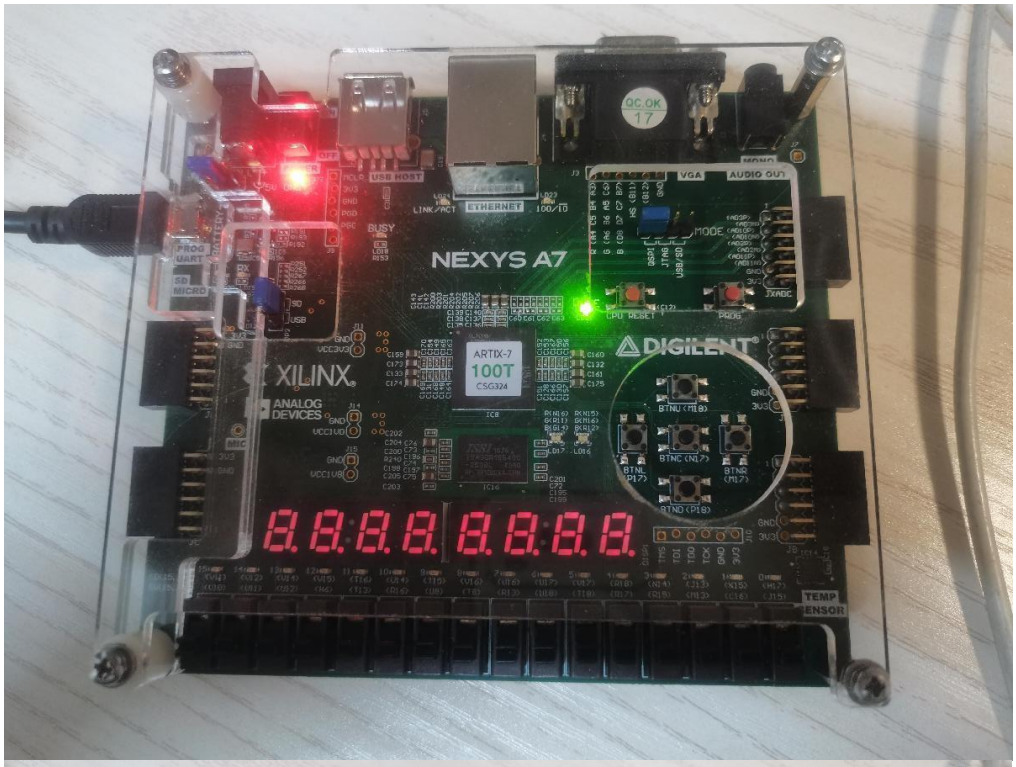
ALU：算术逻辑单元，用于进行 32 位数的计算

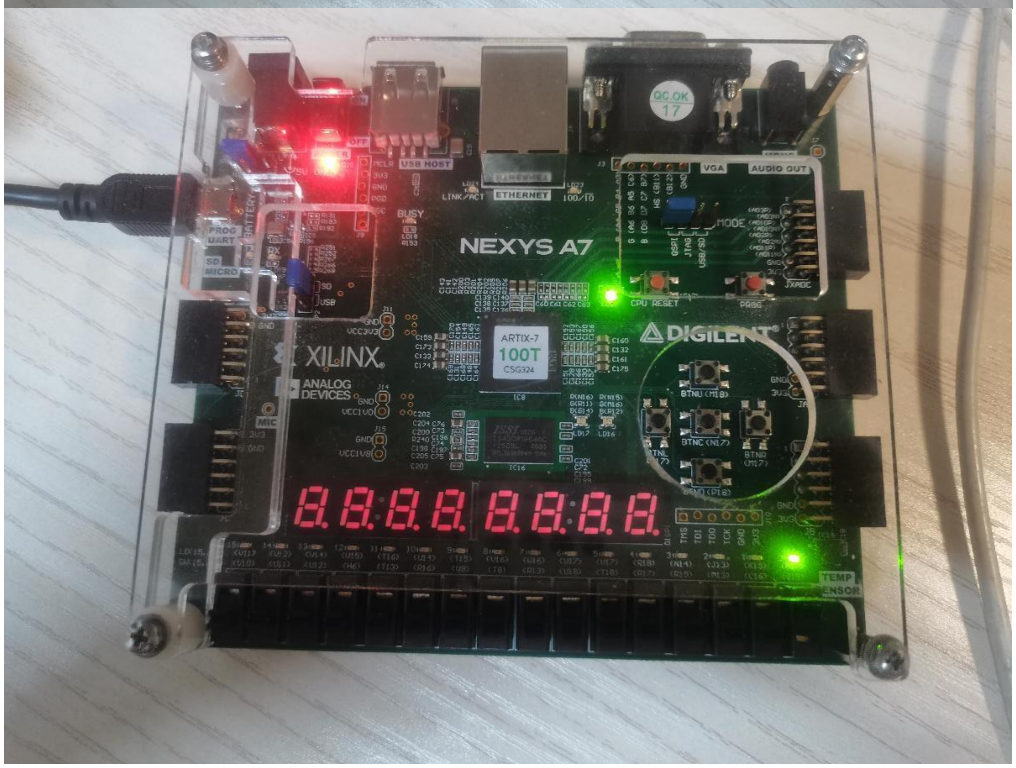
Filter：用于去除上板子时按按钮出现的毛刺

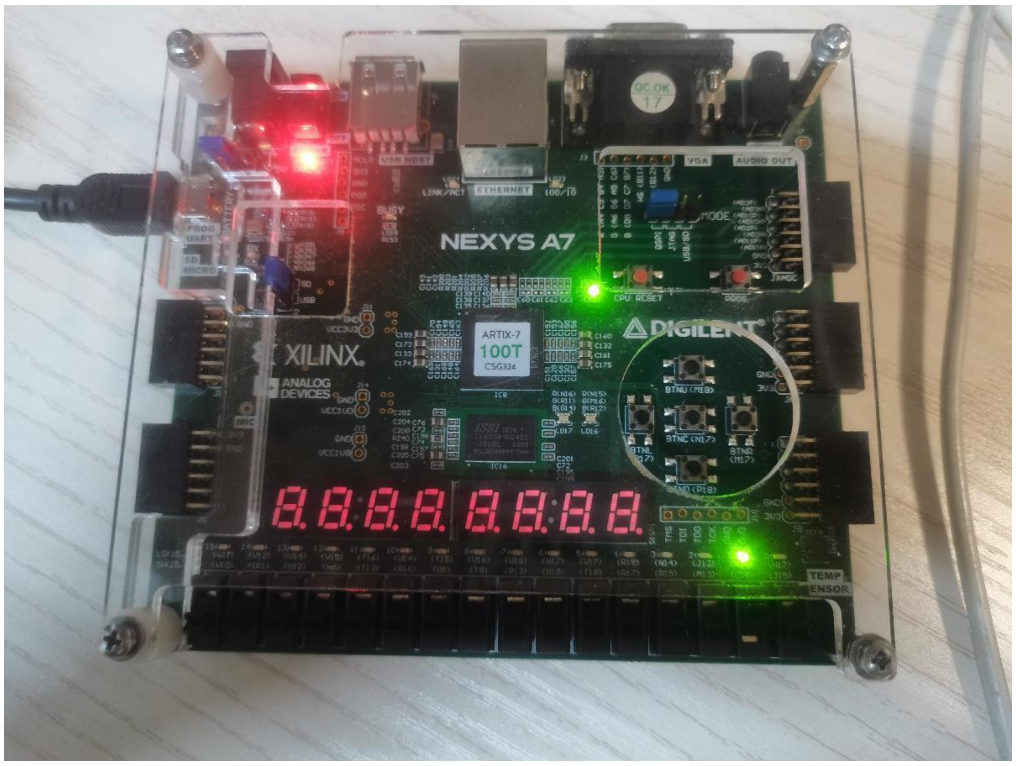
Single_edge：仅仅令触发信号触发一次。

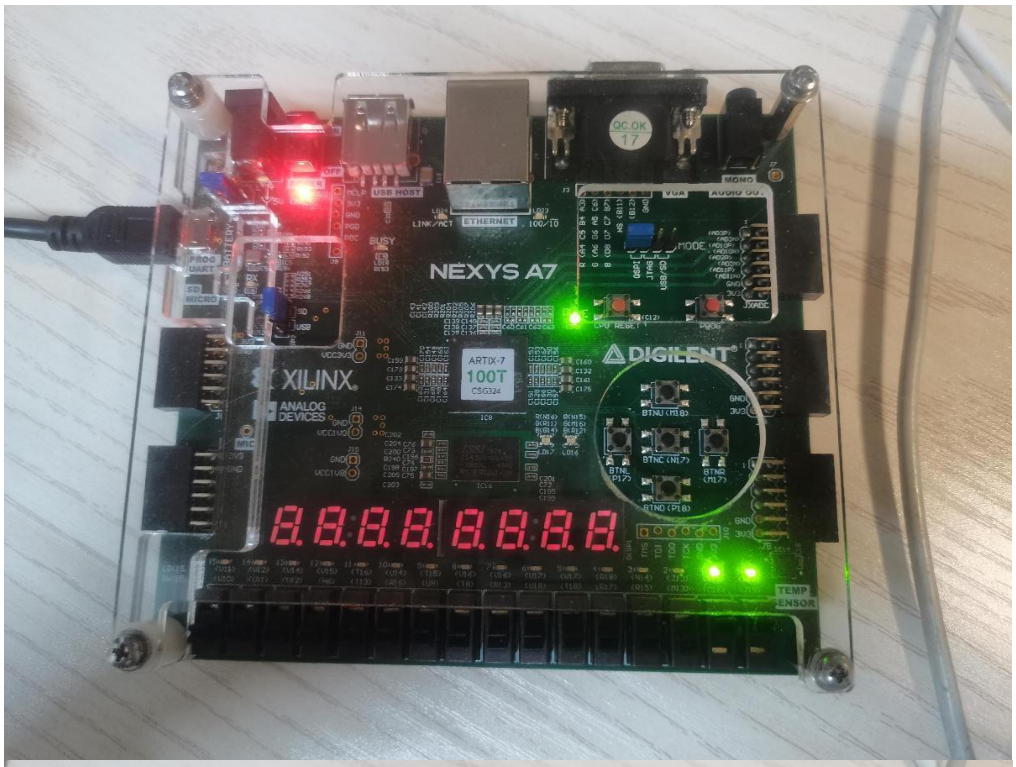
测试结果与分析

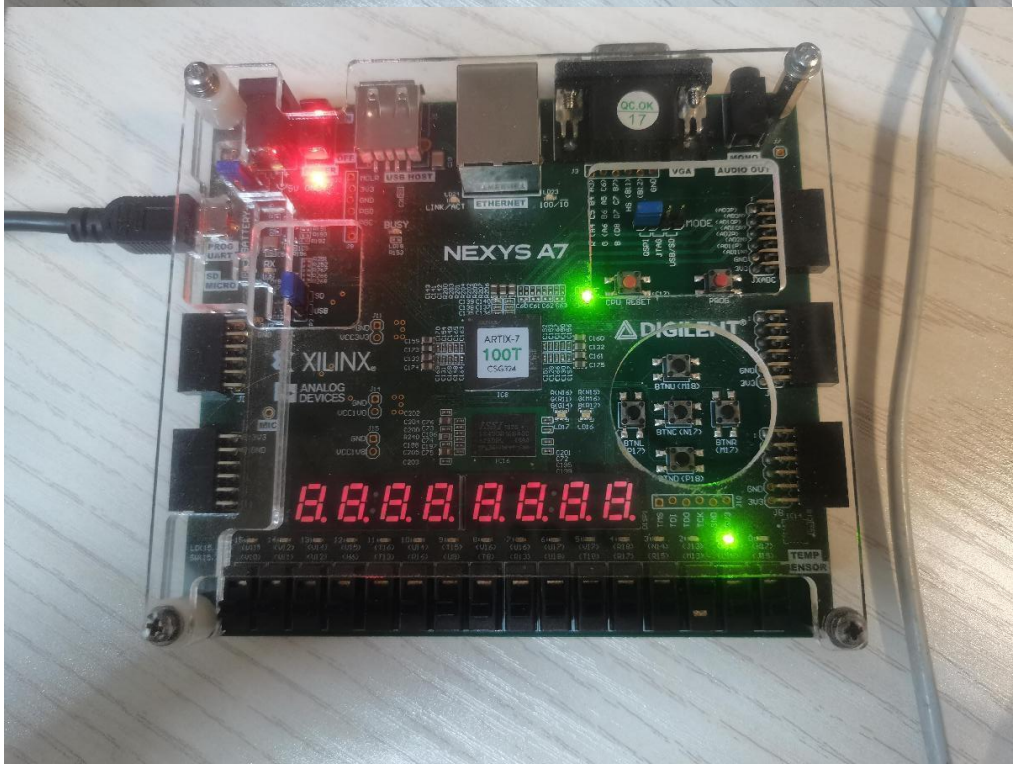
下面的图分别为依次输入 1、2、3、4 和 5 的结果，并利用 led 显示输出结果：

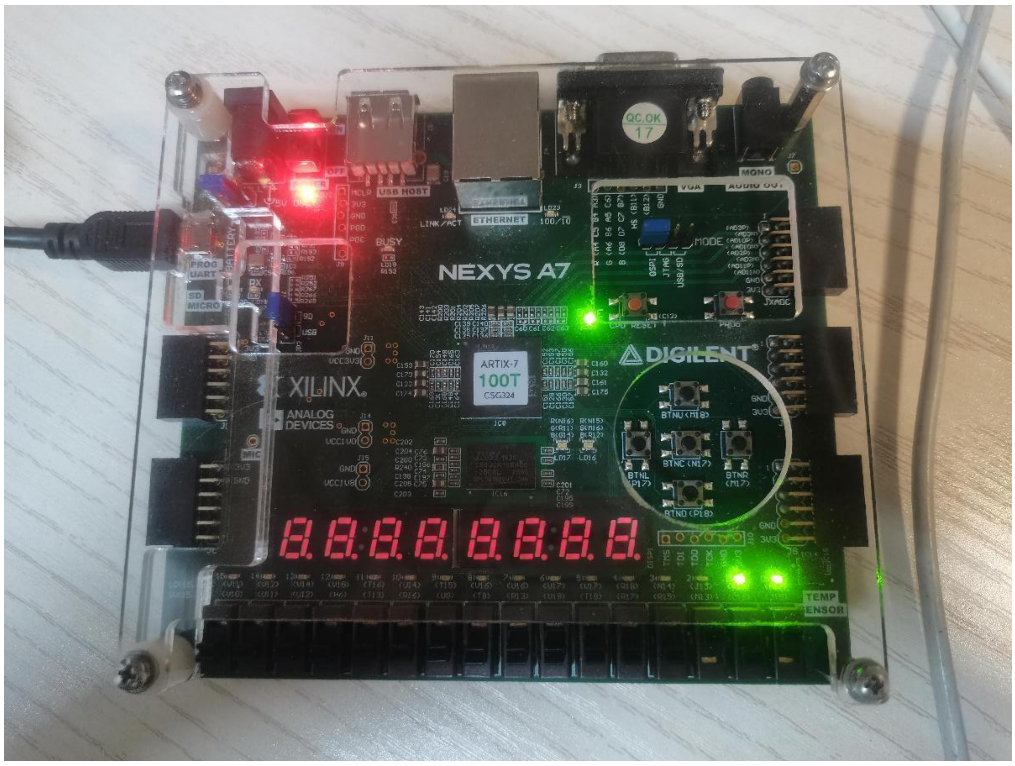





NEXYS A7
XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
CSG324
ETHERNET
USB HOST
VGA
AUDIO OUT
CPU RESET
TEMP SENSOR


NEXYS A7
XILINX
DIGILENT
ANALOG DEVICES
ARTIX-7
100T
ETHERNET
USB HOST
VGA
AUDIO OUT
MODE
TEMP SENSOR


NEXYS A7
XILINX
DIGILENT
ANALOG DEVICES
ARTIX-7
100T
ETHERNET
USB HOST
VGA
AUDIO OUT
MODE
TEMP SENSOR



最开始的图片是板子的初始状态，led 全部为熄灭状态。

第二张图是将 1 输入进去之后的情况，led[0]亮起。同理第三、四张图分别为 2、3 输入进去之后 led 的亮起情况。

第五张图代表了将 4 输入之后，MAV 检测到已经够 4 个数，于是进行求平均值的运算。在程序看来 1 + 2 + 3 + 4 = 10 = 4’b1001，在右移两位之后将变为 4’b0010，与图片中展示的相同。

第六张图片又输入了一个 5.此时 2、3、4、5 的和为 14 = 4’b1110，右移两位之后为 4’b0011，也与图片效果一致。

总结

在本次实验中，我重新熟悉了 Verilog 的语法和编写技巧（以及排版格式），同时初步探索了有关数据通路和模块化之间的关系与妙用。同时，在助教和老师的帮助下，进一步认识了有关两段式有限状态机和时序逻辑组合逻辑的概念和 Verilog 实现。并作为一名初学者，第一次在一块实际的板子上跑了自己的程序。过程中出现了很多的问题，比如最开始我将寄存器的赋值放在了时序逻辑中，我开始没有发现，在检查实验时被助教指出之后，发现在原有基础逻辑上无法改正，于是开始从底层修改逻辑，而且最开始我的状态机只有 4 个状态，在第 4 个状态时使用位移操作，这样会陷入一种僵局，因为赋值是放在组合逻辑中的，于是位移操作便无休止的进行下去直到 4 个数完全相同。于是我修改了状态机的状态数，保证了一次赋值只能影响寄存器中的一个数。但实际上板子的问题还是没有解决，在很长

一段时间里，我的板子在算完 1、2、3、4 之后再算 2、3、4、5 会得到 4 这样的奇怪答案，于是检查自己的状态转移，发现按一次跳转了两个状态，原来是去毛刺没有取干净，于是修改了信号有效所需周期数，发现终于一切正常。

建议：老师的实验文档能否再详细一些，本次实验的 PPT 总给我一种上下文接不上的感觉。以及老师能否考虑到一下有些同学上学期没接触过板子的情况（。以及 ppt 中部分代码有误，还望可以订正一下。

总代码如下：

MAV.v:

```
`timescale 1ns / 1ps

module MAV(
    input  clk,     //时钟
    input  rstn,    //复位信号
    input  en,      //有效信号
    input  [15:0]  d,   //输入数据
    output [15:0] m    //输出数据

);
parameter S0 = 3'b000;
parameter S1 = 3'b001;
parameter S2 = 3'b010;
parameter S3 = 3'b011;
parameter S4 = 3'b100;
parameter S5 = 3'b101;
parameter S6 = 3'b110;
parameter S7 = 3'b111;


reg [2:0] current_state = S0;
reg [2:0] next_state = S0;
reg [63:0] index = 64'b0;
wire [31:0] out,out1,out2,out3;
wire [31:0] two = 32'b0000_0000_0000_0000_0000_0000_0000_0010;
reg [15:0] m0;
wire [2:0] t1,t2,t3,t4;
wire rstn_clear,en1,en_clear;

ALU #(32) alu_add1(.a({16'b0,index[63:48]}),.b({16'b0,index[47:32]}),.f(3'b001),.y(out1),.t(t1));
ALU #(32) alu_add2(.a({16'b0,index[31:16]}),.b({16'b0,index[15:0]}),.f(3'b001),.y(out2),.t(t2));
ALU #(32) alu_add3(.a(out1),.b(out2),.f(3'b001),.y(out3),.t(t3));
```

```
ALU #(32) alu_div(.a(out3),.b(two),.f(3'b101),.y(out),.t(t4));

filter clear1(.clk(clk),.button(rstn),.button_clean(rstn_clear));
// single_edge btn1(.clk(clk),.btn(rstn1),.btn_edge(rstn_edge));     //控制信号只进行激活一次
filter clear2(.clk(clk),.button(en),.button_clean(en1));    //去除按钮毛刺
single_edge btn2(.clk(clk),.btn(en1),.btn_edge(en_clear));

always @(posedge clk,posedge rstn_clear) begin
   if(!rstn_clear) begin
       current_state <= S0;
   end
   else begin
       current_state <= next_state;
   end
end

always@(*) begin
     next_state = current_state;
     if(!rstn_clear) begin
        index = 64'b0;
        m0 = 16'b0;
        next_state = S0;
    end
    else
    case (current_state)
        S0:begin    //复位设置数值
            if(en_clear) begin
                m0 = d;
                index[63:48] = d;
                next_state = S1;
           end
           else begin
                m0 = 16'b0;
                next_state = S0;
            end
        end
        S1:begin    //第一次输入成功
            if(en_clear)begin
                m0 = d;
                index[47:32] = d;
                next_state = S2;
            end
            else begin
                m0 = index[63:48];
                index = index;
                next_state = S1;
            end
```

```
        end
        S2:begin    //第二次输入成功
            // next_state = current_state;
            // index = {index[63:32],32'b0};
            if(en_clear)begin
                m0 = d;
               index[31:16] = d;
               next_state = S3;
            end
           else begin
                m0 = index[47:32];
                index = index;
                next_state = S2;
            end
        end
        S3:begin
            // next_state = current_state;
            // index = {index[63:16],16'b0};
            if(en_clear) begin
                m0 = d;
                index[15:0] = d;
                next_state = S4;
            end
            else begin
                m0 = index[31:16];
                index = index;
                next_state = S3;
            end
        end
        S4:begin
            if(en_clear) begin
                index[63:48] = d;
                m0 = out[15:0];
                next_state = S5;
            end
            else begin
                m0 = out[15:0];
                index = index;
                next_state = S4;
            end
        end
        S5: begin
            if(en_clear) begin
                index[47:32] = d;
                m0 = out[15:0];
                next_state = S6;
            end
            else begin
```

```
                m0 = out[15:0];
                index = index;
                next_state = S5;
            end

        end
        S6: begin
            if(en_clear) begin
                index[31:16] = d;
                m0 = out[15:0];
                next_state = S7;
            end
             else begin
                next_state = S6;
                index = index;
                m0 = out[15:0];
            end
        end
        S7: begin
            if(en_clear) begin
                index[15:0] = d;
                m0 = out[15:0];
                next_state = S4;
            end
             else begin
                m0 = out[15:0];
                index = index;
                next_state = S7;
            end
        end
   endcase
end
assign m = {m0[15:0]};
endmodule
```

ALU.v:

```
`timescale 1ns / 1ps

module ALU #(
parameter WIDTH = 32     //数据宽度
)(
input signed [WIDTH-1:0] a, b,       //两操作数
input [2:0] f,                       //功能选择
output reg [WIDTH-1:0] y,     //运算结果
output reg [2:0] t                     //比较标志
```

```
);

always@(*)begin
    case (f)
        3'b000: begin
            y = a - b;
            if(a[WIDTH-1] == b[WIDTH - 1])begin
                if(a == b)
                    t = 3'b001;
                else
                    t = (a>b)?3'b000:3'b110;
            end
            else
                t = (a[WIDTH-1] > b[WIDTH-1])?3'b010:3'b100;
        end
        3'b001: begin
            y = a + b;
            t = 3'b000;
        end
        3'b010: begin
            y = a & b;
            t = 3'b000;
        end
        3'b011: begin
            y = a | b;
            t = 3'b000;
        end
        3'b100: begin
            y = a ^ b;
            t = 3'b000;
        end
        3'b101: begin
            y = a >> b[4:0];
            t = 3'b000;
        end
        3'b110: begin
            y = a << b[4:0];
            t = 3'b000;
        end
        3'b111: begin
            y = a >>> b[4:0];
            t = 3'b000;
        end
    endcase

end

endmodule
```

Filter.v:

```
`timescale 1ns / 1ps


module filter(
    input clk,
    input button,
    output button_clean
);
reg [15:0] cnt;
always@(posedge clk) begin
    if(button==1'b0)
        cnt <= 16'h0;
    else if(cnt<16'b1111_0000_0000_0000)
        cnt <= cnt + 1'b1;
end
assign button_clean = cnt[15]&cnt[14]&cnt[13]&cnt[12];
endmodule
```

Single_edge.v:

```
`timescale 1ns / 1ps

module single_edge(
    input clk,
    input btn,
    output btn_edge
);
reg btn1 = 1'b0;
reg btn2 = 1'b0;
always@(posedge clk)
begin
    btn1 <= btn;
end
always@(posedge clk)
begin
    btn2 <= btn1;
end
assign btn_edge = btn1 & (~btn2);
endmodule
```