# Lab1

## 实验目的与内容

1.ALU

- 掌握算术逻辑单元 (ALU) 的功能
- 掌握数据通路和有限状态机的设计方法
- 掌握组合电路和时序电路，以及参数化、结构化的Verilog
- 描述方法

2.FLS：ALU应用

## 逻辑设计

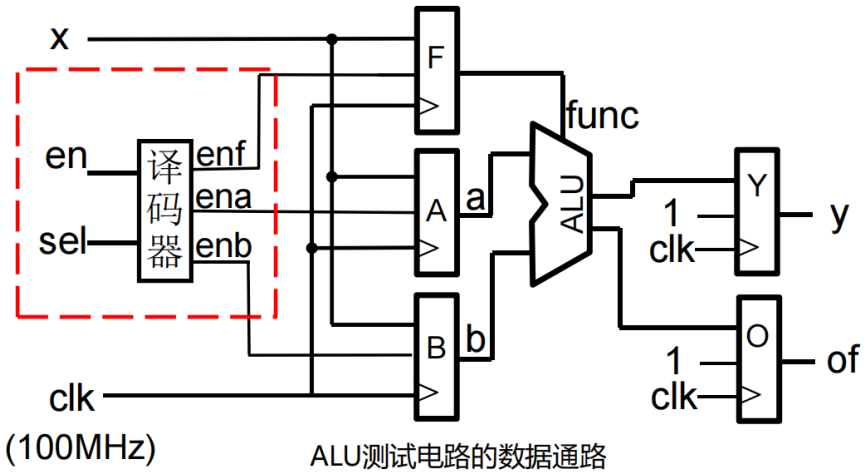
一、ALU设计与测试

### ◘ ALU测试电路设计

- ✓ **ALU测试电路的数据通路如下图所示，该电路主要由寄存器，译码器，ALU单元组成。采用结构化描述方式设计ALU测试电路。**
- ✓ **由于FPGAOL外设资源有限，因此端口需要分时复用：操作数a, b和功能f 复用开关输入x[5:0]。复用方法：通过sel和en ，译码生成寄存器使能信号ena，enb，enf（译码器真值表如下表所示），将开关输入x[5:0]分时存入寄存器 F(x[3:0])，A(x[5:0])，B(x[5:0])。**

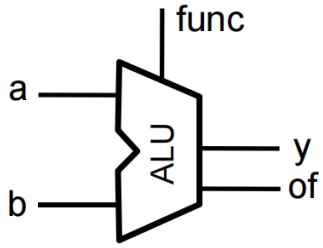
译码器真值表

| en | sel | ena | enb | enf |
|---|---|---|---|---|
| 1 | 00 | 1 | 0 | 0 |
| 1 | 01 | 0 | 1 | 0 |
| 1 | 10 | 0 | 0 | 1 |
| 1 | 11 | 0 | 0 | 0 |
| 0 | xx | 0 | 0 | 0 |




ALU测试电路的数据通路

## ◘ ALU模块功能介绍




**溢出判断只针对有符号数;**
**相等判断时，y输出为0或1（零扩展）；**
**比较判断时，y输出为0或1（零扩展）；**
**< 为带符号比较:**
采用$signed(a) < $signed(b)
**$<_u$ 为无符号比较:**
采用 a<b
*** 表示根据运算结果设置.**

ALU模块功能表

| func | y | of |
|---|---|---|
| 0000 | a + b | * |
| 0001 | a – b | * |
| 0010 | a == b | 0 |
| 0011 | $a <_u b$ | 0 |
| 0100 | a < b | 0 |
| 0101 | a & b | 0 |
| 0110 | a \| b | 0 |
| 0111 | a ^ b | 0 |
| 1000 | a >> b | 0 |
| 1001 | a << b | 0 |
| 其他 | 0 | 0 |

**绿色背景表项为必做内容，其他为选做内容**

- ALU文件结构

```
lab01
├── dec: decoder
└── ALU: alu
```

- ALU核心代码
  - alu单元

```
`timescale 1ns / 1ps

module alu #(parameter WIDTH = 6) (
  input [WIDTH-1:0] a,
  input [WIDTH-1:0] b,
  input [3:0] func,
  output [WIDTH-1:0] y,
  output of
);

  reg [WIDTH-1:0] y_reg;// 输出结果 y
  reg of_reg;// 溢出标志位寄存器

  always @(*) begin
    case (func)
```

```
16        // ADD
17        4'b0000: begin
18          y_reg = a + b;
19          of_reg = (a[WIDTH-1] == b[WIDTH-1]) && (y_reg[WIDTH-1] !=
    a[WIDTH-1]);
20        end
21        // SUB
22        4'b0001: begin
23          y_reg = a - b;
24          of_reg = (a[WIDTH-1] != b[WIDTH-1]) && (y_reg[WIDTH-1] !=
    a[WIDTH-1]);
25        end
26        // EQ
27        4'b0010: begin y_reg = (a == b) ? 1 : 0;  of_reg <= 0;    end
28        // LT_u
29        4'b0011: begin y_reg = (a < b) ? 1 : 0;
30        of_reg <= 0; end
31        // LT_s
32        4'b0100: begin y_reg = ($signed(a) < $signed(b)) ? 1 : 0;  of_reg
    <= 0;  end
33        // AND
34        4'b0101: begin y_reg = a & b;
35        of_reg <= 0;  end
36        // OR
37        4'b0110: begin y_reg = a | b;
38        of_reg <= 0;  end
39        // XOR
40        4'b0111: begin y_reg = a ^ b;
41        of_reg <= 0;  end
42        // SHR
43        4'b1000: begin y_reg = a >> b;
44        of_reg <= 0;  end
45        // SHL
46        4'b1001: begin y_reg = a << b;
47        of_reg <= 0;  end
48        // Default is 0
49        default: begin
50           y_reg = 0;
51           of_reg = 0;
52        end
53      endcase
54    end
55
56    assign y = y_reg;
57    assign of = of_reg;
58
59  endmodule
```

```
60
```

- decoder

```
module decoder(
    input en,
    input [1:0] sel,
    output reg ena, enb, enf // 引入寄存器类型的输出端口
);

always @(*) begin // 在一个总是块中定义逻辑表达式
    ena = (sel == 2'b00) ? en : 1'b0;
    enb = (sel == 2'b01) ? en : 1'b0;
    enf = (sel == 2'b10) ? en : 1'b0;
end

endmodule
```

- main

```
module lab01(
  input clk,
  input en,
  input [1:0]sel,
  input [5:0] x,
  output reg [5:0] y,
  output reg of
);

wire of1;
wire ena1, enb1, enf1;
reg [3:0] func;
reg [5:0] a;
reg [5:0] b;
wire [5:0] y1;

decoder dec(.en(en),.sel(sel),.ena(ena1),.enb(enb1),.enf(enf1));

alu ALU(.a(a),.b(b),.func(func),.y(y1),.of(of1));

always@(posedge clk)
begin
  if(ena1) a <= x;
  if(enb1) b <= x;
  if(enf1) func <= x[3:0];
```

```
    y <= y1;
    of <= of1;
end
endmodule
```

## 二、FLS

- FLS文件结构

```
my_fls
├── getedge: signal_edge
├── adder: alu
└── fsm1: fsm
```

- FLS核心代码
  - 取信号边沿

```
`timescale 1ns / 1ps
// 声明模块signal_edge，输入为时钟，置位信号，使能信号，输出为信号变化的边缘
module signal_edge(
    input clk,
    input rst,
    input button,
    output button_edge
);
    reg button_r1, button_r2;

    // always块，当时钟的上升沿到来时，更新button_r1
    always @(posedge clk)
        button_r1 <= ~rst & button;

    // always块，当时钟的上升沿到来时，更新button_r2
    always @(posedge clk)
        button_r2 <= button_r1;

    // 将button_r1和button_r2进行与操作，得到输出信号button_edge
    assign button_edge = button_r1 & ~button_r2;

endmodule

```

  - 状态机

```
```

```verilog
// 状态机模块：声明模块fsm，输入为时钟，重置信号，使能信号，输出为当前状态
module fsm(
    input clk,
    input rst,
    input en,
    output [1:0] state
);

    // 声明常量F0，F1和F2，表示三种状态
    parameter F0 = 2'b00;
    parameter F1 = 2'b01;
    parameter F2 = 2'b10;
    // 声明两个寄存器curr_state和next_state，存储当前状态和下一个状态
    reg [1:0] curr_state;
    reg [1:0] next_state;

    // always块，当当前状态改变时，根据状态转移表更新下一个状态
    // Moore的次态仅与现态有关
    always @(*) begin
        if(en) begin
        case (curr_state)
            F0:    next_state = F1;
            F1:    next_state = F2;
            F2:    next_state = F2;
            default:     next_state = F0;
        endcase
        end
        else begin
            next_state = curr_state;
        end
    end

    // always块，当时钟的上升沿到来时，根据重置信号或使能信号更新当前状态
    always @(posedge clk or posedge rst) begin
        if (rst)
            curr_state <= F0;
        else if (en)
            curr_state <= next_state;
    end

    // 将当前状态作为输出
    assign state = curr_state;

endmodule
```

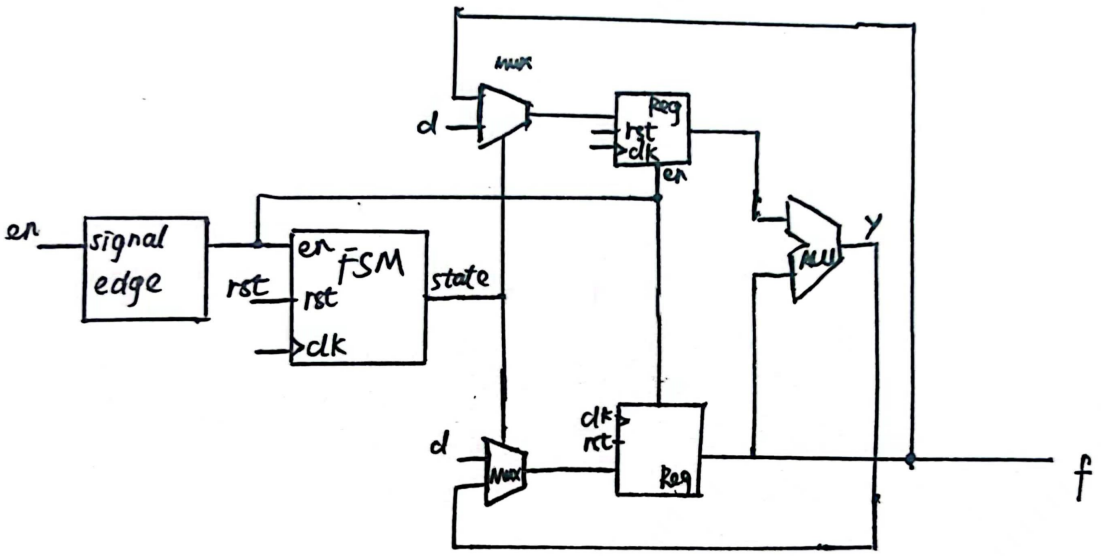
- alu模块与上面ALU部分的alu代码一致

- 顶层代码my_fls

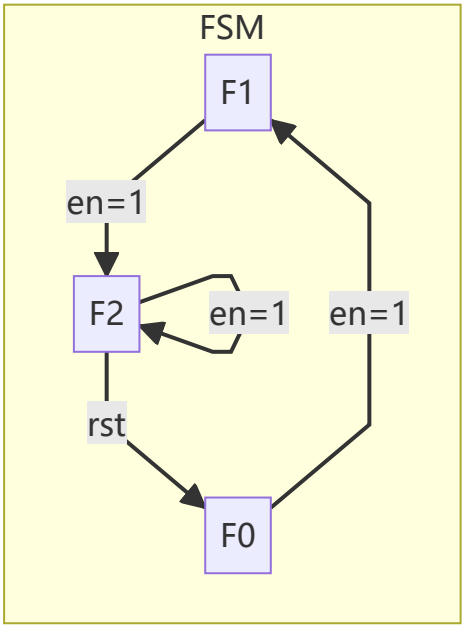
```
module my_fls(
    input clk,     // 时钟信号
    input rst,     // 复位信号
    input en,      // 使能信号
    input [6:0] d, // 数据输入
    output reg [6:0] f // 数据输出
    );

    // 获取边沿
    wire sedge;
    signal_edge getedge(
        .clk(clk),
        .rst(rst),
        .button(en),
        .button_edge(sedge)
    );

    // ALU
    reg [6:0] a;    // 寄存器a
    wire [6:0] sum; // ALU实现一个加法
    alu #(.WIDTH(7)) adder(
        .a(a),
        .b(f),
        .func(4'b0000),
        .y(sum)
    );

    // FSM
    wire [1:0] sel; // 状态选择信号
    fsm fsm1(
        .clk(clk),
        .rst(rst),
        .en(sedge),
        .state(sel)
    );

    // registers and MUXes
    always @(posedge clk) begin
        if (rst) // 复位时
            a <= 7'h00; // 将寄存器a清零
        else if (sedge) begin
            case (sel)
                2'b00: a <= d; // 状态0，将输入数据d存入寄存器a
                2'b10: a <= f; // 状态2，将输出数据f存入寄存器a
                default: a <= a; // 其他状态，保持原值
```

```
46              endcase
47          end
48      end
49
50      always @(posedge clk) begin
51          if (rst) // 复位时
52              f <= 7'h00; // 将输出数据f清零
53          else if (sedge) begin
54              case (sel)
55                  2'b00: f <= d; // 状态0和1，将输入数据d存入输出数据f
56                  2'b01: f <= d;
57                  2'b10: f <= sum; // 状态2，将ALU计算结果存入输出数据f
58                  default: f <= f; // 其他状态，保持原值
59              endcase
60          end
61      end
62
63  endmodule
64
```

- FLS数据通路




- 有限状态机




# 仿真结果与分析

- ALU仿真

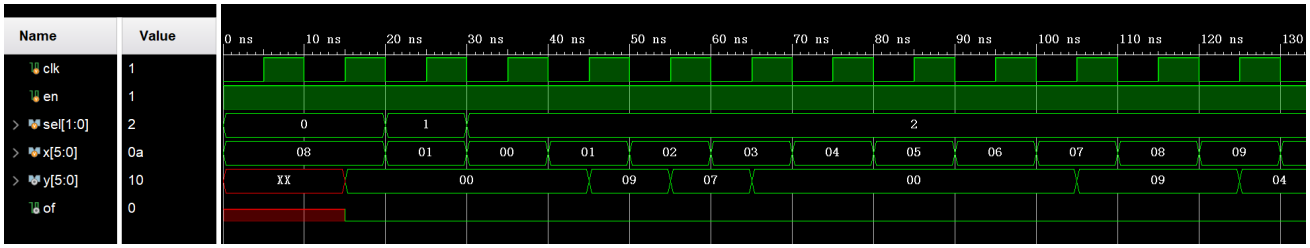
  仿真测试文件

```
1  `timescale 1ns / 1ps
2
3  module tb();
4      reg clk;
5      reg en;
6      reg [1:0] sel;
7      reg [5:0] x;
8
9      wire [5:0] y;
10
11     lab01 uut (
12         .clk(clk),
13         .en(en),
14         .sel(sel),
15         .x(x),
16         .y(y),
17         .of(of)
18     );
19
20     initial begin
21         // Initialize Inputs
22         clk = 0;
```

```
        en = 1;
        sel = 0;
        x = 6'b001000;
        #10;
        sel=2'b00;
        #10;
        x = 6'b000001;
        sel=2'b01;
        #10;
        x = 6'b000000;
        sel=2'b10;
        #10;
        x = 6'b000001;
        sel=2'b10;
        #10;
        x = 6'b000010;
        sel=2'b10;
        #10;
        x = 6'b000011;
        sel=2'b10;
        #10;
        x = 6'b000100;
        sel=2'b10;
        #10;
        x = 6'b000101;
        sel=2'b10;
        #10;
        x = 6'b000110;
        sel=2'b10;
        #10;
        x = 6'b000111;
        sel=2'b10;
        #10;
        x = 6'b001000;
        sel=2'b10;
        #10;
        x = 6'b001001;
        sel=2'b10;
        #10;
        x = 6'b001010;
        sel=2'b10;
        #10;
        $finish;

    end

    always #5 clk = ~clk;
```

```
endmodule

```

仿真波形



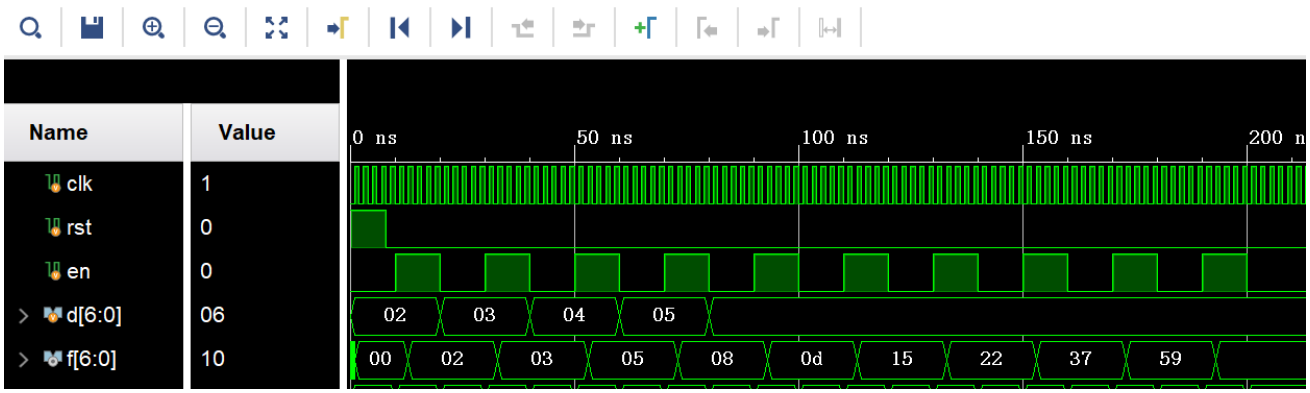

- FLS仿真

  仿真测试文件

```
module tb();
  reg clk;
  reg rst;
  reg en;
  reg [6:0] d;
  wire [6:0] f;

  my_fls test(
    .clk(clk),
    .rst(rst),
    .en(en),
    .d(d),
    .f(f)
  );

  parameter T = 1;

  // 声明时钟信号，每隔 T 个单位时间翻转一次
  always #(T) clk = ~clk;

  // 初始化时钟信号为 0，400个单位时间后结束仿真
  initial begin
    clk = 1'b0;
    #400 $finish;
  end

  // 初始化复位信号为 1，7个单位时间后置为 0
  initial begin
    rst = 1'b1;
    #8 rst = 1'b0;
  end

```

```
33    // 初始化 en 信号的时序，使用 for 循环修改 en 的值
34    integer i;
35    initial begin
36      en = 1'b0;
37      for (i = 0; i < 20; i = i + 1) begin
38        #10 en = ~en;
39      end
40    end
41
42    // 初始化 d 信号的时序，使用 for 循环修改 d 的值
43    integer j;
44    initial begin
45      d = 7'h02;
46      for (j = 0; j < 4; j = j + 1) begin
47        #20 d = d + 1;
48      end
49    end
50
51  endmodule
52
```

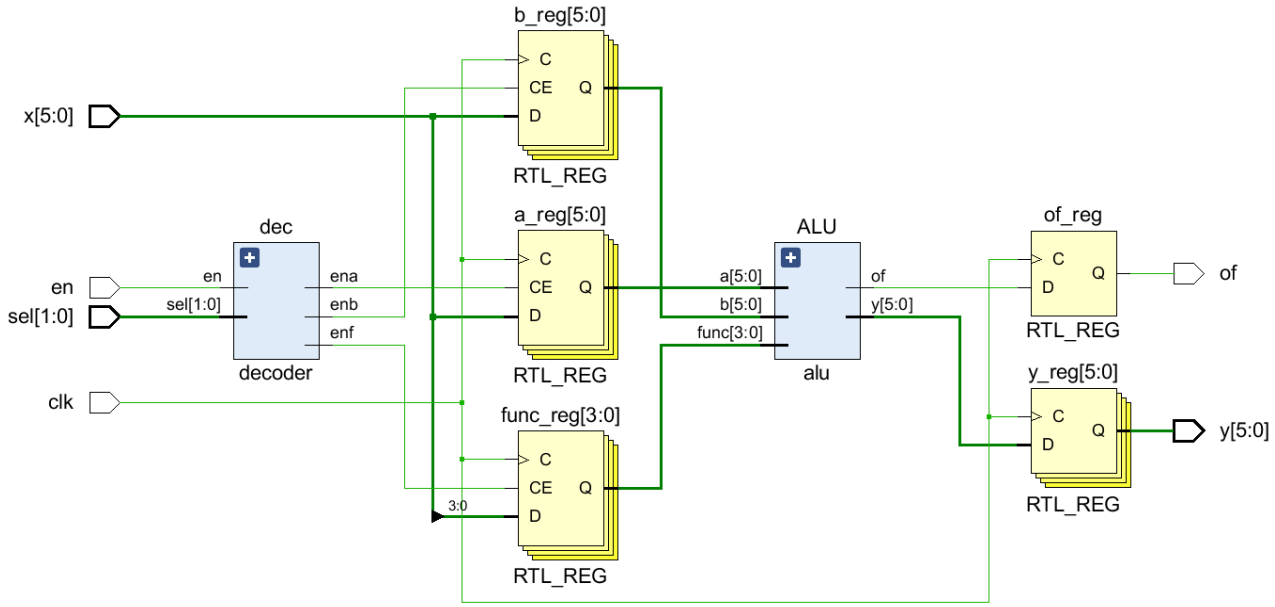
仿真波形



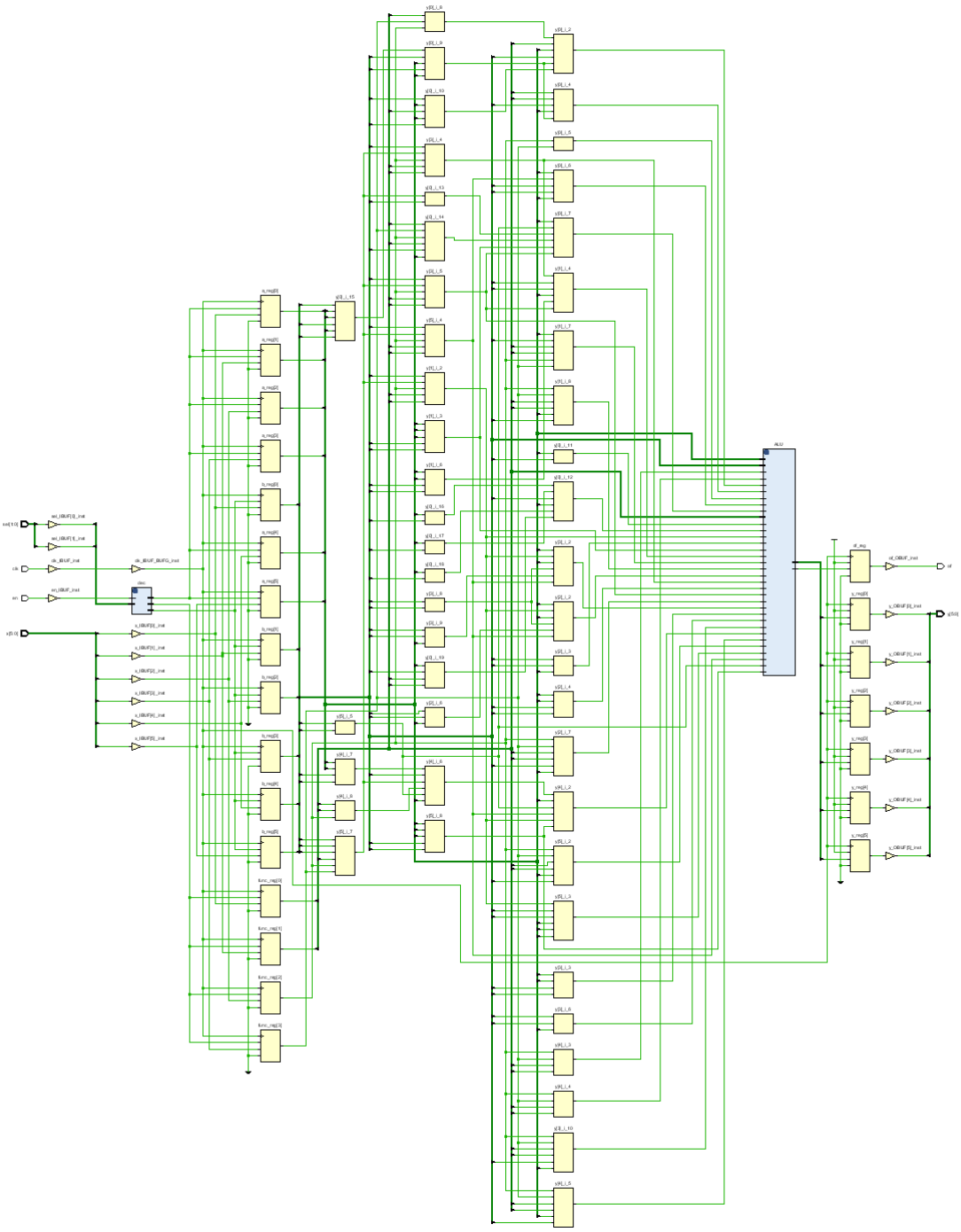

# 电路设计与分析

一、ALU

- RTL分析电路


b_reg[5:0]
C
CE
D
Q
RTL_REG
x[5:0]
dec
en
sel[1:0]
clk
ena
enb
enf
decoder
a_reg[5:0]
RTL_REG
ALU
a[5:0]
b[5:0]
func[3:0]
of
y[5:0]
alu
of_reg
RTL_REG
of
y_reg[5:0]
RTL_REG
y[5:0]
func_reg[3:0]
3:0
RTL_REG

- 综合后电路



- 二者异同

  相同点：

  - 在RTL代码分析电路原理图和综合后电路原理图中，模块的输入和输出端口以及模块的结构都是相同的。
  - 在RTL代码分析电路原理图和综合后电路原理图中，都需要使用逻辑门和触发器等基本电路元件。

  不同点：

  - 电路原理图的层次结构不同。在RTL代码分析电路原理图中，更容易看到代码的层次结构和模块之间的连接方式，而在综合后电路原理图中，这种结构并不容易看出来。

- 综合器可以优化电路以减少面积和功耗，并生成更复杂的电路结构，从而使电路的性能和可靠性更好。因此，综合后电路原理图中可能存在的逻辑门和触发器的数量和种类，可能与 RTL 代码分析电路原理图中的数量和种类不同。
- 综合后电路原理图中，可能会出现与输入和输出端口无关的中间信号线，这是综合器根据优化算法生成的。

综合后电路原理图的电路结构更加优化和复杂，而且综合器可以根据性能和面积等方面的要求进行优化，因此综合后的电路可能与 RTL 代码分析电路原理图中的电路结构存在较大的差异。

# 总结收获

收获：

- 重新对上学期的数电实验相关知识进行了温习，包括状态机、verilog的一些语法等等。
- 对RTL电路和综合后电路有了进一了解。

建议：

- 实验手册可以更清楚点的！刚开始我甚至没弄明白要干啥（雾。