Lab 1 report

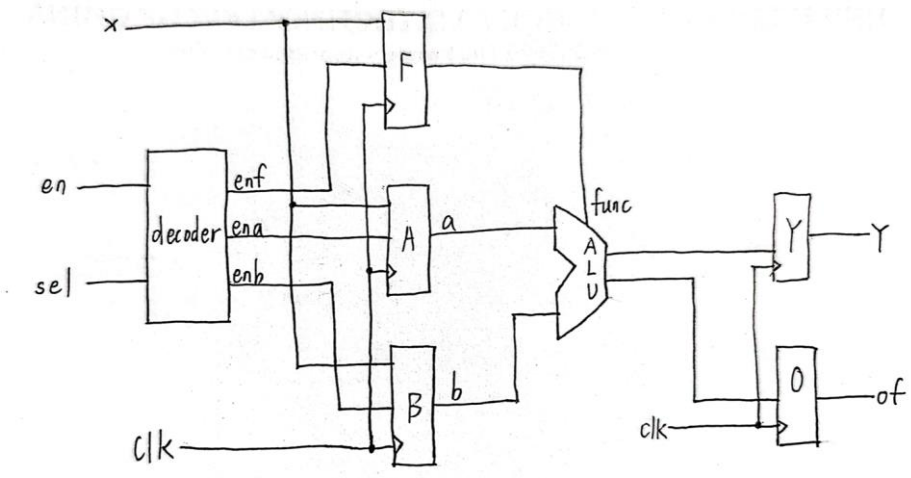
实验目的与内容

本次实验的主要内容是实现 ALU 的模块化结构和完成 FLS 的层次化设计，用 Verilog 语言进行描述，Vivado 实现综合与仿真，最后在 FPGAOL 平台完成烧写。主要目的在于对数字电路（组合与时序电路）和 Verilog 知识的复习，巩固数据通路和有限状态机的设计方法，并初步了解参数化，层次化的设计思想。

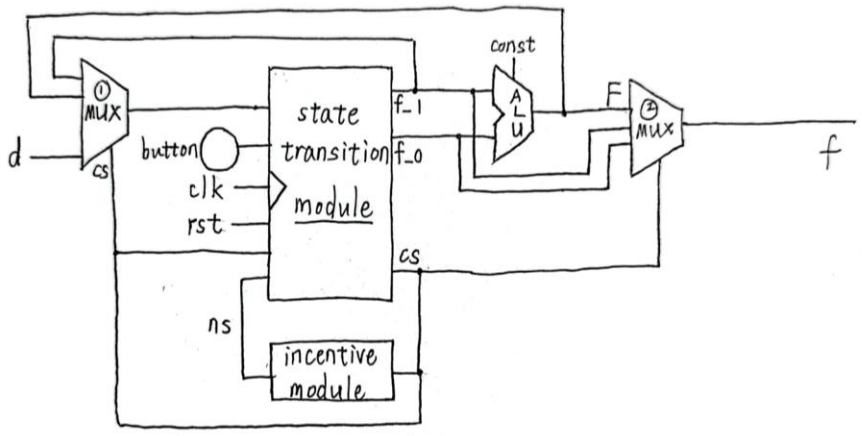
逻辑设计

数据通路设计

ALU 算术逻辑单元：



FLS 斐波那契卢卡斯数列计算器：



注：状态转换模块根据现态选通内部寄存器，从而决定覆写对象(f_0 或 f_1)；mux1 同样根据现态选择输入数据。

状态机设计：

此处仅列出 FLS 斐波那契卢卡斯数列计算器的有限状态机设计。

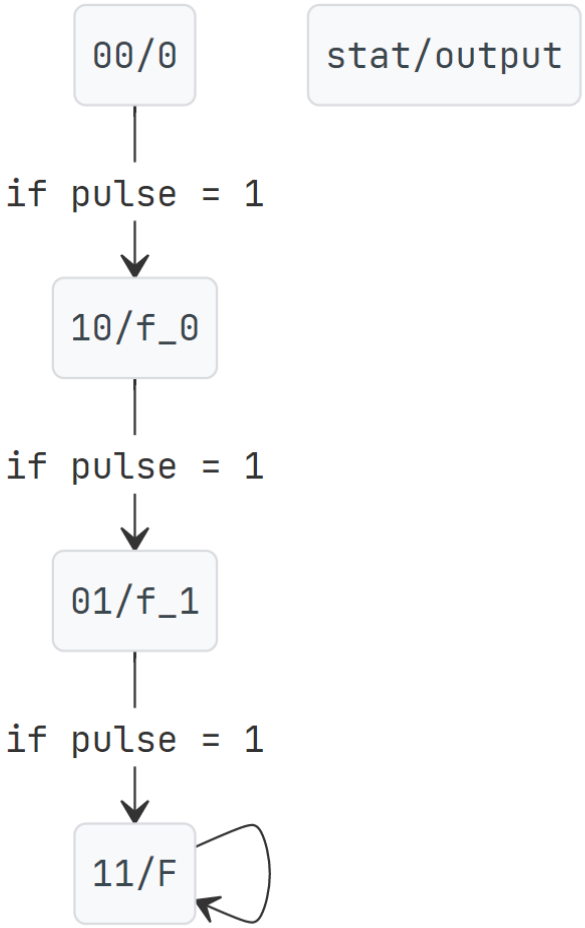
状态编码：

该 FSM 共四种状态，分别为 2’b00,2’b10,2’b01,2’b11。有三种中间数据，分别为 f(n-2),f(n-1),f(n-2)+f(n-1)。在设计文件中分别用 f_0,f_1,F 代表上述数据。

下面列出每种状态的含义：

- 2’b00：尚未存储 f(n-2)和 f(n-1)，此时输出 f 为 0。
- 2’b10：已存储 f(n-2)但未存储 f(n-1)，此时输出 f 为 f(n-2)。
- 2’b01：已存储 f(n-2)和 f(n-1)且尚未进入数列循环累加阶段，此时输出 f 为 f(n-1)。
- 2’b11：已存储 f(n-2)和 f(n-1)且已进入数列循环累加阶段，此时输出 f 为 f(n-2)+f(n-1)。

状态转换图：



注：pulse 不为 1 时，所有状态均保持原有状态。

核心代码设计：

各个模块的核心代码部分及必要注释如下：

Module--Alu_unit:

```
    always @(*)
    begin
        case(func)
            4'b0000: begin
                y=a+b;
                OF=(~a[WIDTH-1]&~b[WIDTH-1]&y[WIDTH-1] | a[WIDTH-
1]&b[WIDTH-1]&~y[WIDTH-1]);
            end
```

```
            4'b0001: begin
                y=a-b;
                OF=(~a[WIDTH-1]&b[WIDTH-1]&y[WIDTH-1] | a[WIDTH-
1]&~b[WIDTH-1]&~y[WIDTH-1]);
            end
           //正数加正数得负数，负数加负数得正数为溢出，故列出最高位最
小项表达式如上。减法同理。
            4'b0010: begin
                y=(a==b);
                OF=0;
            end
            ……//后续功能分支描述同理
        endcase
    end
```

Module--decoder:

```
    always @(*)
    begin
        if(en)
        begin
            case(sel)
                2'b00:{ena,enb,enf}=3'b100;
                2'b01:{ena,enb,enf}=3'b010;
                2'b10:{ena,enb,enf}=3'b001;
                2'b11:{ena,enb,enf}=3'b000;
            endcase
        end
        else begin
            {ena,enb,enf}=3'b000;
        end
    end//使能有效时正常译码，无效时输出位清零。
```

Module--reg:

```
    always @(posedge clk)
    begin
        if(en)o<=x;
    end
```

Top Module--ALU:

```
    wire [5:0] a,b,FUNC,Y;
    wire enf,ena,enb;
    wire OF;
    decoder dec(.sel(sel),.en(en),.enf(enf),.ena(ena),.enb(enb));
    FF Freg(.clk(clk),.en(enf),.x(x),.o(FUNC));
    FF Areg(.clk(clk),.en(ena),.x(x),.o(a));
    FF Breg(.clk(clk),.en(enb),.x(x),.o(b));
    alu_unit alu(.a(a),.b(b),.func(FUNC[3:0]),.y(Y),.OF(OF));
    FF Yreg(.clk(clk),.en(1),.x(Y),.o(y));
    FF Oreg(.clk(clk),.en(1),.x(OF),.o(of));
//按照数据通路连接各级模块即可
//输入输出溢出(of)信息时只保留对应寄存器的[0]位
```

Top Module—FLS:

采用了标准的状态机三段式描述方法：

```
    //初始化
    reg [1:0] cs=2'b00,ns;
    reg [6:0] f_0,f_1;
    wire [6:0] F;
//中间参数,分别代表 f(n-2),f(n-1),f(n-2)+f(n-1)
    alu_unit #(7) alu(.a(f_0),.b(f_1),.func(4'b0000),.y(F));
//给定参数 7，实例化一个 alu unit
```

```
//双寄存器构造单周期长度脉冲
    reg r1=0,r2=0;
    wire pulse;
    always@(posedge clk)
    r1<=en;
    always@(posedge clk)
    r2<=r1;
    assign pulse=r1&(~r2);
//这一步使得每次按下 button(en)时，无论 en 信号持续时间长短，都只产生一
次有效脉冲(pulse)用于后续操作，防止按压时间过长造成多次状态转换。
```

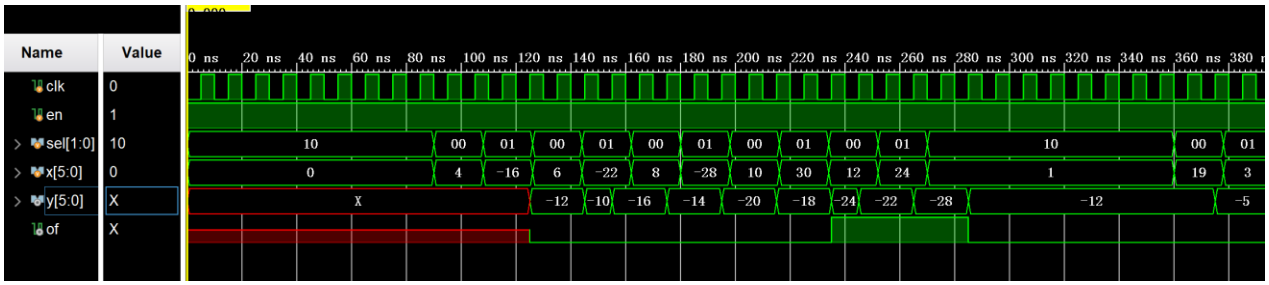
```
// ----------------------------------------------
//状态转换模块
always@(posedge clk)
begin
    if(rst)
    begin
        f_0=0;
        f_1=0;
        cs=2'b00;
    end
       //复位时存储值与状态均重置
    else if(pulse)//在有效脉冲时
    begin
        cs<=ns;
       //状态转换
        case(cs)
        2'b00:f_0<=d;
        2'b10:f_1<=d;
        2'b11:
        begin
            f_0<=f_1;
            f_1<=F;
        end
        endcase
       //该块主要描述由现态进入次态时的行为
       //三种中间参量在状态转换时进行维护：
       //2’b00:若现态为 00，则次态时应存储 f_0
       //2’b10:若现态为 10，则次态时应存储 f_1
       //2’b01:若现态为 01，则执行运算，不存储任何值(保持原态)
       //2’b11:若现态为 11，则次态按数列迭代方法修改 f_0,f_1
    end
end
// ----------------------------------------------
```

```
    // ----------------------------------------------
    //状态激励模块
    always@(*)
    begin
        case(cs)
        2'b00:ns=2'b10;
        2'b10:ns=2'b01;
        2'b01:ns=2'b11;
        2'b11:ns=2'b11;
        endcase
      //按照状态转换图进行描述即可
    end
    // ----------------------------------------------
```

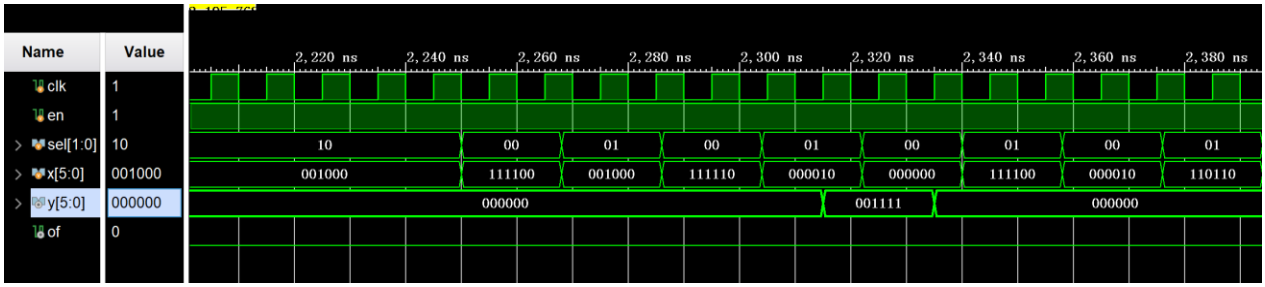
```
    //组合输出模块
    assign
f=(cs==2'b11)?F:((cs==2'b10)?f_0:((cs==2'b01)?f_1:7'b0));
//多分支组合输出模块
//根据现态决定输出，与状态图描述一致
    // ----------------------------------------------
```

仿真结果与分析

Top Module--ALU:

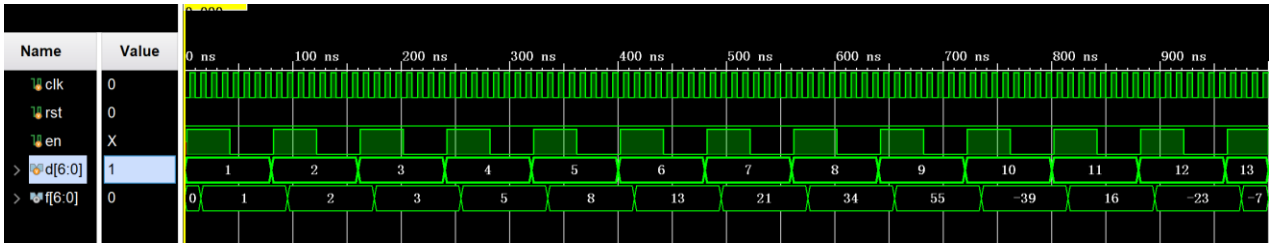


每个周期的前 90ns 选通 func 寄存器，后 180ns 每 36ns 为一个输入周期，分别写入 a(00),b(01)，然后在下一个时钟周期执行相应运算。图中波形为 func=4’b0000 时的加法运算，注意从到 a=10,b=30 时开始发生溢出，其余位置均正确产生了结果。



上图展示的是 a>>b 运算的结果。

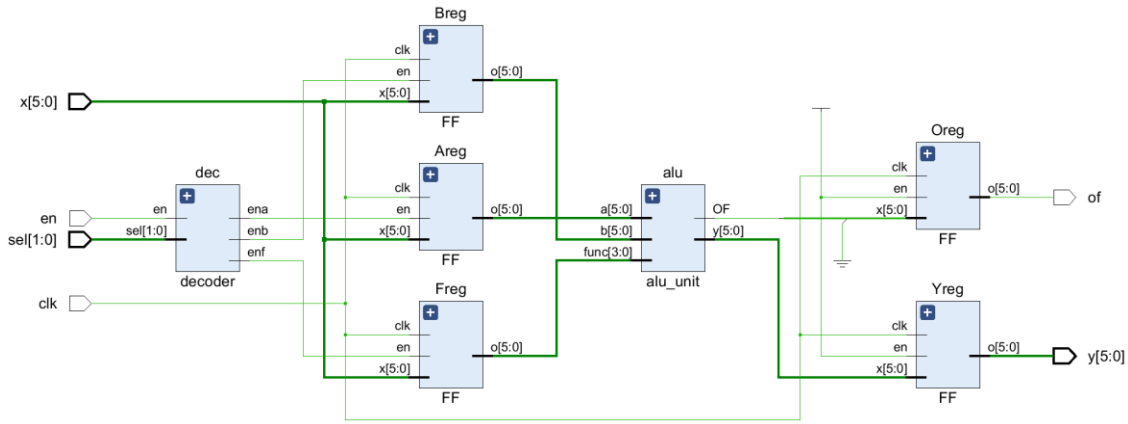
Top Module--FLS:



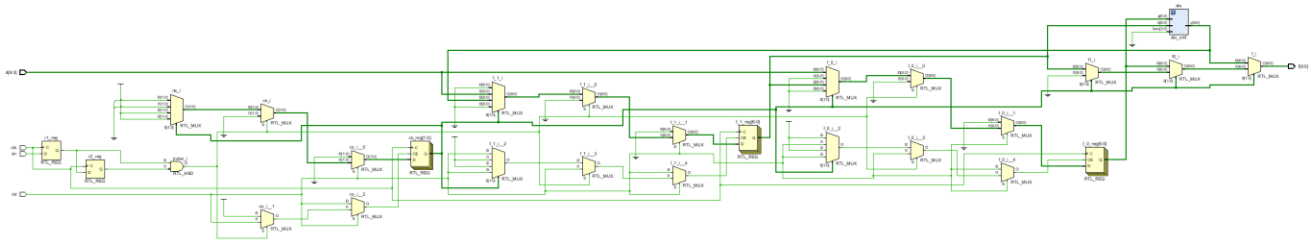
1. 前两次按压 button 时分别录入 f_0,f_1 的值，后续按压 button 则按照斐波那契数列迭代方法计算数列元素。这里采用的 f_0 和 f_1 的值分别为 1 和 2，可以看出前两次按压 button 后输出了 f_0,f_1 的值，后续的运算也正确输出了每个斐波那契数列元素。
2. 请贴出有特点的仿真测试文件，并说明你在编写仿真测试文件时，对各类情况的考虑（选做）。

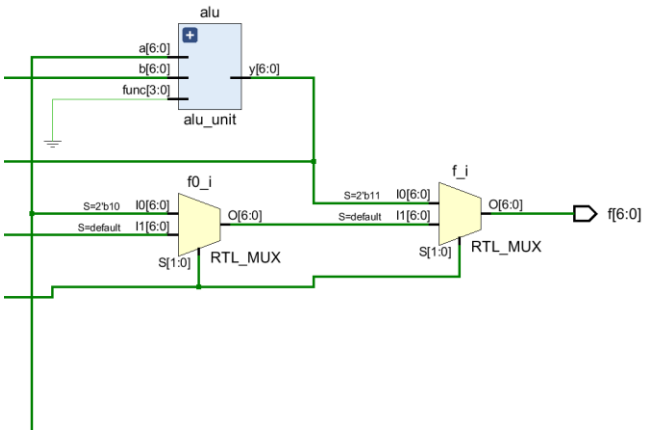
电路设计与分析

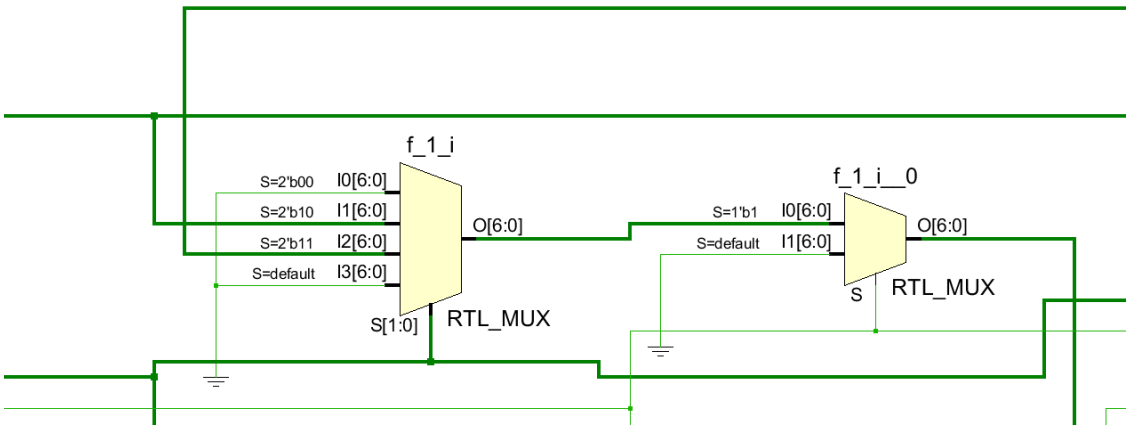
Top Module--ALU:



Top Module--FLS:



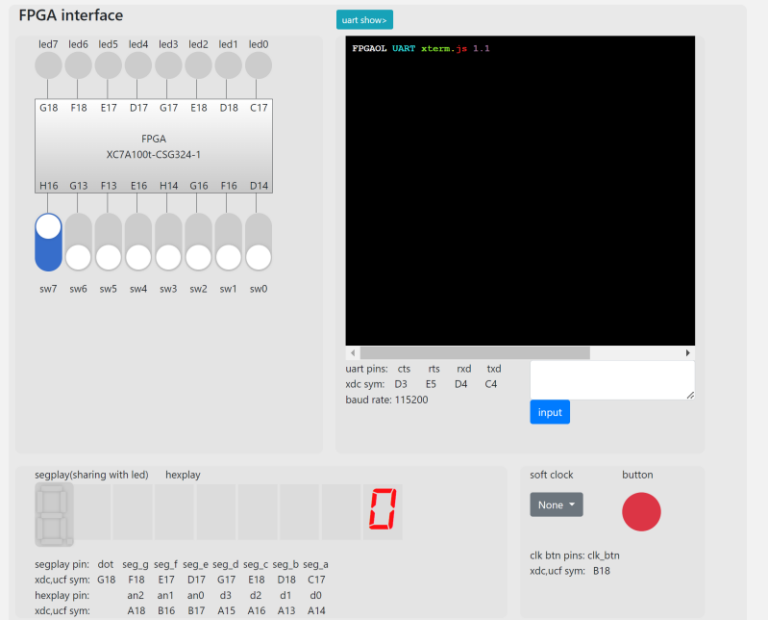




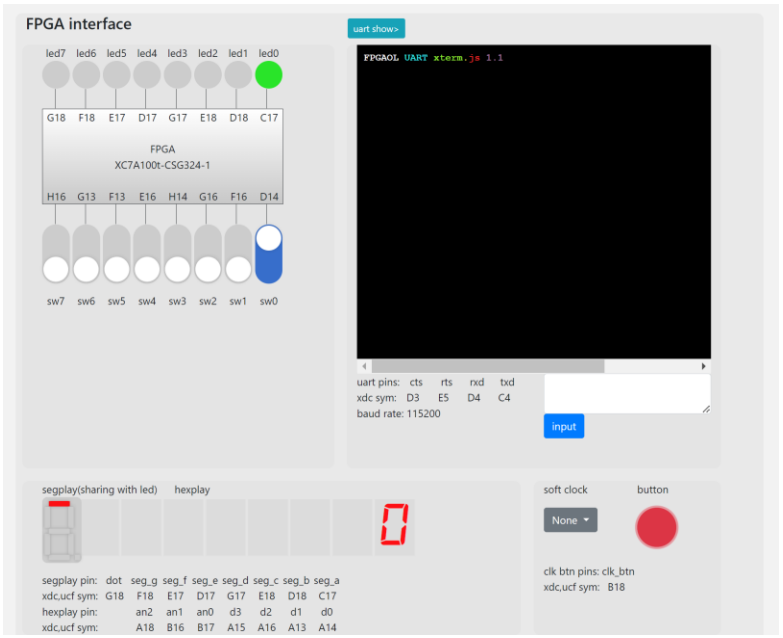
状态转换模块与状态激励模块较为复杂，但总体仍符合数据通路设计。

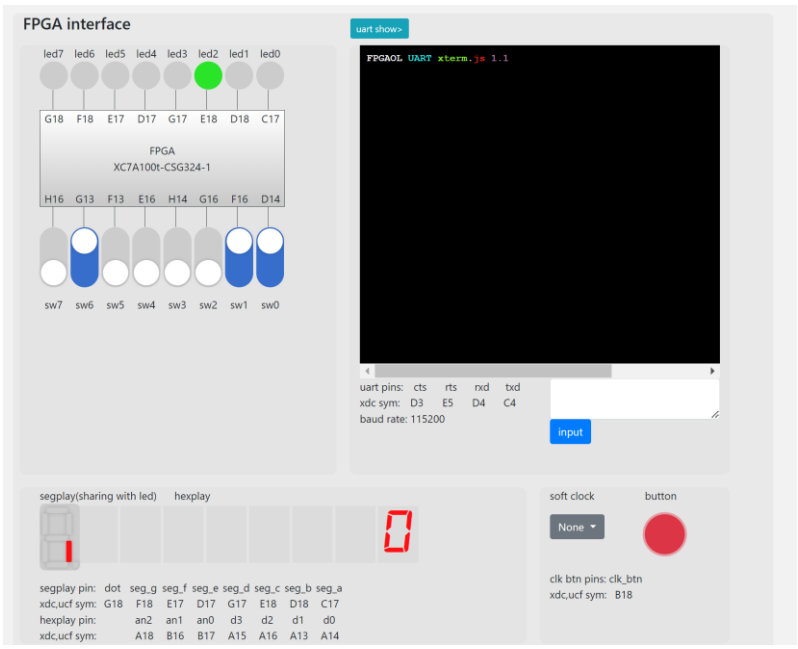
测试结果与分析

Top Module--ALU:

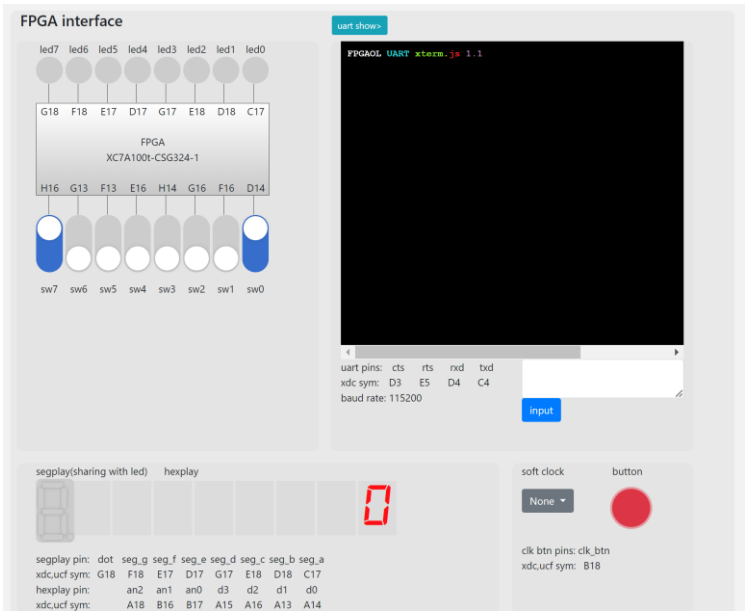


选通 func，输入 0000，执行加法。

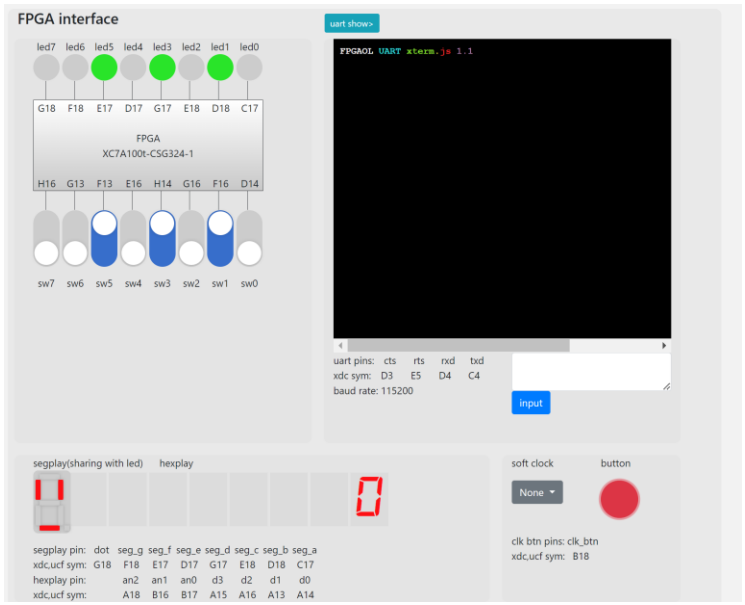


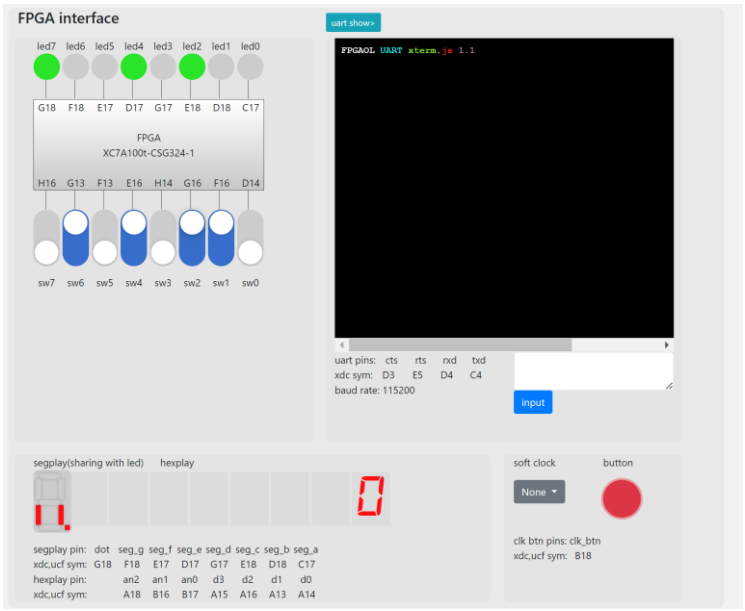


输入 a=1 与 b=3，第二次按下 button 后产生结果 4。



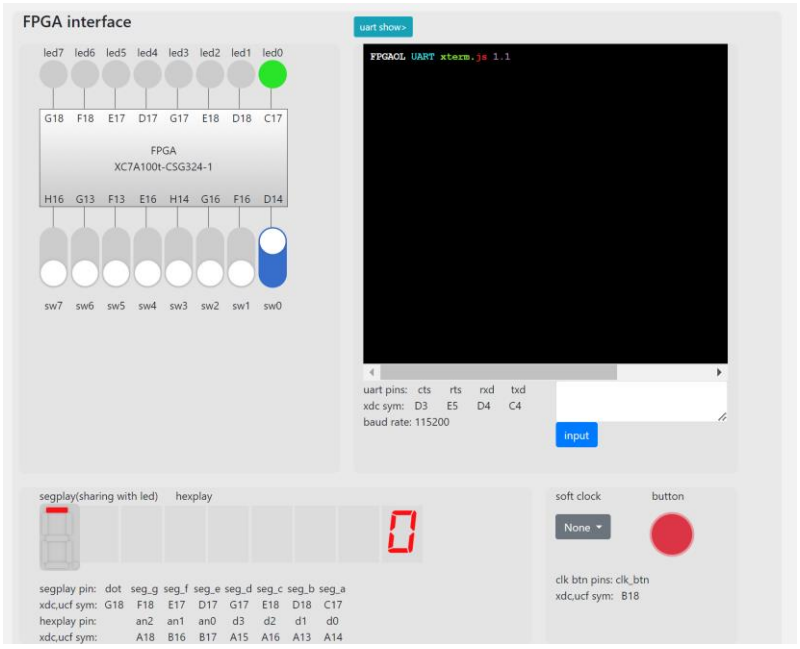
Func 输入 0001，执行减法



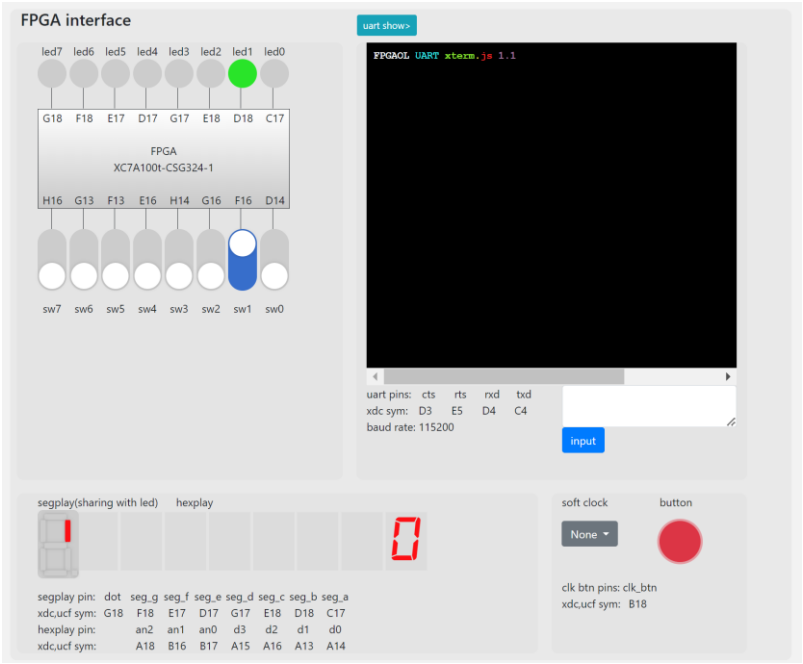


输入 a=-22,b=22,相减后发生溢出，OF 信号亮起，溢出后的正确结果为 20。

Top Module--FLS:

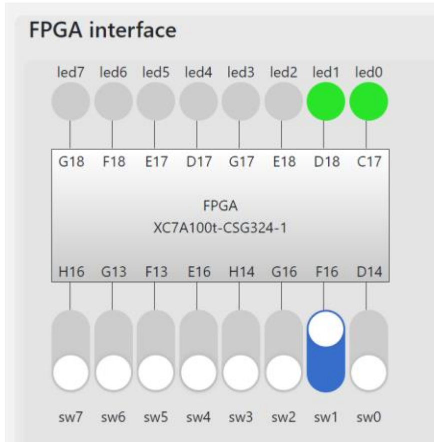


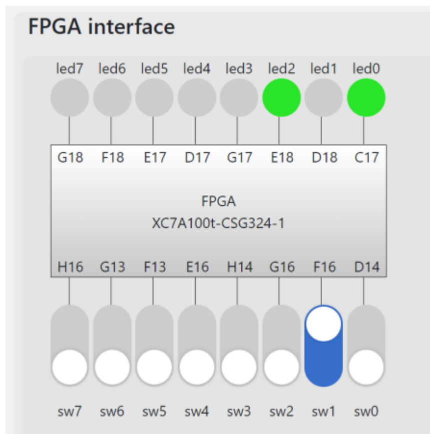
第一个状态(2’b00)至第二个状态(2’b10)，输入 f_0=1，led 显示如图。

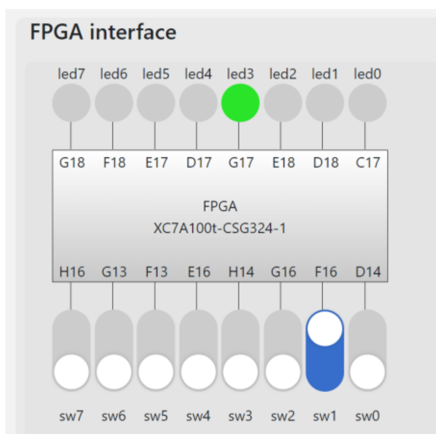


状态 2’b10，输入 f_1=2，进入状态 2’b01。Led 显示如图。

第三次按下 button 后，进入无限循环的 2’b11 状态。第三，四，五次按下 button 的 led 显示如图。结果符合迭代公式。







第三次按下 button 后，进入无限循环的 2’b11 状态。第三，四，五次按下 button 的 led 显示如图。结果符合迭代公式。

总结

经过本次实验，完成了对数字电路和 Verilog 有关知识的复习，巩固了数据通路和有限状态机的设计方法，并初步了解了参数化，层次化的设计思想。

本次实验个人认为难度适中，涵盖内容较为全面，ALU 算术逻辑单元的选题也有实际意义，有利于后续更深层次的学习。

附件

设计文件：

```
`timescale 1ns / 1ps
module alu_unit
    #(parameter WIDTH = 6)
    (
        input [WIDTH-1:0] a, b,
        input [3:0] func,
        output reg [WIDTH-1:0] y,
        output reg OF
    );
    always @(*)
    begin
        case(func)
            4'b0000: begin
                y=a+b;
                OF=(~a[WIDTH-1]&~b[WIDTH-1]&y[WIDTH-1] | a[WIDTH-
1]&b[WIDTH-1]&~y[WIDTH-1]);
            end
            4'b0001: begin
                y=a-b;
                OF=(~a[WIDTH-1]&b[WIDTH-1]&y[WIDTH-1] | a[WIDTH-
1]&~b[WIDTH-1]&~y[WIDTH-1]);
            end
            4'b0010: begin
                y=(a==b);
                OF=0;
            end
            4'b0011: begin
                y=(a<b);
                OF=0;
            end
            4'b0100: begin
                y=($signed(a)<$signed(b));
                OF=0;
            end
            4'b0101: begin
```

```
                y=a&b;
                OF=0;
            end
            4'b0110: begin
                y=a|b;
                OF=0;
            end
            4'b0111: begin
                y=a^b;
                OF=0;
            end
            4'b1000: begin
                y=a>>b;
                OF=0;
            end
            4'b1001: begin
                y=a<<b;
                OF=0;
            end
            default: begin
                y=0;
                OF=0;
            end
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
module decoder
    (
        input [1:0] sel,
        input en,
        output reg ena,enb,enf
    );
    always @(*)
    begin
        if(en)
```

```
        begin
            case(sel)
                2'b00:{ena,enb,enf}=3'b100;
                2'b01:{ena,enb,enf}=3'b010;
                2'b10:{ena,enb,enf}=3'b001;
                2'b11:{ena,enb,enf}=3'b000;
            endcase
        end
        else begin
            {ena,enb,enf}=3'b000;
        end
    end
endmodule
```

```
`timescale 1ns / 1ps
module FF
    (
        input clk,en,
        input [5:0] x,
        output reg [5:0] o
    );
    always @(posedge clk)
    begin
        if(en)o<=x;
    end
endmodule
```

```
`timescale 1ns / 1ps
module ALU
    (
        input clk,en,
        input [1:0] sel,
        input [5:0] x,
        output [5:0] y,
        output of
```

```
    );
    wire [5:0] a,b,FUNC,Y;
    wire enf,ena,enb;
    wire OF;
    decoder dec(.sel(sel),.en(en),.enf(enf),.ena(ena),.enb(enb));
    FF Freg(.clk(clk),.en(enf),.x(x),.o(FUNC));
    FF Areg(.clk(clk),.en(ena),.x(x),.o(a));
    FF Breg(.clk(clk),.en(enb),.x(x),.o(b));
    alu_unit alu(.a(a),.b(b),.func(FUNC[3:0]),.y(Y),.OF(OF));
    FF Yreg(.clk(clk),.en(1),.x(Y),.o(y));
    FF Oreg(.clk(clk),.en(1),.x(OF),.o(of));
endmodule
```

```
module FLS
    (
        input clk,rst,
        input en,
        input [6:0] d,
        output [6:0] f
    );
    //初始化
    reg [1:0] cs=2'b00,ns;
    reg [6:0] f_0,f_1;
    wire [6:0] F;
    alu_unit #(7) alu(.a(f_0),.b(f_1),.func(4'b0000),.y(F));
    //双寄存器构造单周期脉冲
    reg r1=0,r2=0;
    wire pulse;
    always@(posedge clk)
    r1<=en;
    always@(posedge clk)
    r2<=r1;
    assign pulse=r1&(~r2);
    // ---------------------------------------------
    //状态转换模块
    always@(posedge clk)
    begin
        if(rst)
```

```
        begin
            f_0=0;
            f_1=0;
            cs=2'b00;
        end
        else if(pulse)
        begin
            cs<=ns;
            case(cs)
            2'b00:f_0<=d;
            2'b10:f_1<=d;
            2'b11:
            begin
                f_0<=f_1;
                f_1<=F;
            end
            endcase
        end
    end
    // -----------------------------------------------
    //状态激励模块
    always@(*)
    begin
        case(cs)
        2'b00:ns=2'b10;
        2'b10:ns=2'b01;
        2'b01:ns=2'b11;
        2'b11:ns=2'b11;
        endcase
    end
    // -----------------------------------------------
    //组合输出模块
    assign
f=(cs==2'b11)?F:((cs==2'b10)?f_0:((cs==2'b01)?f_1:7'b0));
    // -----------------------------------------------
endmodule
```

仿真文件：

```
`timescale 1ns / 1ps
module ALU_testbench();
reg clk,en;
reg [1:0] sel;
reg [5:0] x;
wire [5:0] y;
wire of;
reg [5:0] a,b,f;
ALU ALU(.clk(clk),.en(en),.sel(sel),.x(x),.y(y),.of(of));
integer k=0;
initial
begin
    clk=0;
    en=1;
    sel=2'b10;
    x=6'b0;
    f=6'b0;
    a=6'b0;
    b=6'b111111;
end
always #5 clk=~clk;
always #270 f=f+1;
always #18 a=a+1;
always #18 b=b-3;
always
begin
    if(k<10)
    begin
        case(sel)
        2'b10:
        begin
            x=f;
            #90 sel=sel-2;
        end
        2'b00:
        begin
            x=a;
            #18 sel=sel+1;
            k=k+1;
```

```
        end
        2'b01:
        begin
            x=b;
            #18 sel=sel-1;
            k=k+1;
        end
        default:k=k+1;
        endcase
    end
    else
    begin
        k=0;
        sel=2'b10;
    end
end
endmodule
```

```
`timescale 1ns / 1ps
module FLS_testbench();
reg clk=0,rst=0;
reg en;
reg [6:0] d=7'b0000001;
wire [6:0] f;
FLS FLS(.clk(clk),.rst(rst),.en(en),.d(d),.f(f));
always #5 clk=~clk;
always
begin
    #2 en=1;
    #40 en=0;
    #38;
end
always #80 d=d+1;
initial
begin
    #3000 rst=1;
end
```

```
endmodule
```

约束文件：

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk }];
set_property -dict { PACKAGE_PIN B18  IOSTANDARD LVCMOS33}
[get_ports { en }];
## FPGAOL SWITCH
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 }
[get_ports { x[0] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 }
[get_ports { x[1] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 }
[get_ports { x[2] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 }
[get_ports { x[3] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 }
[get_ports { x[4] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 }
[get_ports { x[5] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 }
[get_ports { sel[0] }];
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 }
[get_ports { sel[1] }];
## FPGAOL LED
set_property -dict { PACKAGE_PIN C17  IOSTANDARD LVCMOS33}
[get_ports { y[0] }];
set_property -dict { PACKAGE_PIN D18  IOSTANDARD LVCMOS33}
[get_ports { y[1] }];
set_property -dict { PACKAGE_PIN E18  IOSTANDARD LVCMOS33}
[get_ports { y[2] }];
set_property -dict { PACKAGE_PIN G17  IOSTANDARD LVCMOS33}
[get_ports { y[3] }];
set_property -dict { PACKAGE_PIN D17  IOSTANDARD LVCMOS33}
[get_ports { y[4] }];
set_property -dict { PACKAGE_PIN E17  IOSTANDARD LVCMOS33}
[get_ports { y[5] }];
```

```
set_property -dict { PACKAGE_PIN G18  IOSTANDARD LVCMOS33}
[get_ports { of }];
```

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk }];
set_property -dict { PACKAGE_PIN B18  IOSTANDARD LVCMOS33}
[get_ports { en }];
## FPGAOL SWITCH
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 }
[get_ports { d[0] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 }
[get_ports { d[1] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 }
[get_ports { d[2] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 }
[get_ports { d[3] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 }
[get_ports { d[4] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 }
[get_ports { d[5] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 }
[get_ports { d[6] }];
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 }
[get_ports { rst }];
## FPGAOL LED
set_property -dict { PACKAGE_PIN C17  IOSTANDARD LVCMOS33}
[get_ports { f[0] }];
set_property -dict { PACKAGE_PIN D18  IOSTANDARD LVCMOS33}
[get_ports { f[1] }];
set_property -dict { PACKAGE_PIN E18  IOSTANDARD LVCMOS33}
[get_ports { f[2] }];
set_property -dict { PACKAGE_PIN G17  IOSTANDARD LVCMOS33}
[get_ports { f[3] }];
set_property -dict { PACKAGE_PIN D17  IOSTANDARD LVCMOS33}
[get_ports { f[4] }];
set_property -dict { PACKAGE_PIN E17  IOSTANDARD LVCMOS33}
[get_ports { f[5] }];
```

```
set_property -dict { PACKAGE_PIN F18  IOSTANDARD LVCMOS33}
[get_ports { f[6] }];
```