实验一 运算器及其应用

实验目标

- 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

复习：Verilog描述注意事项

- **推荐使用简单规范的描述方式**
- **组合电路**
  - 使用assign 或者 always @* 描述, “ = ” 赋值
  - always描述时避免不完全赋值（否则出现锁存器)！
  - 避免出现反馈！例如，y = y + x
  - 无需复位，即组合函数的自变量中无复位信号
- **时序电路**
  - 使用always @(posedge clk, posedge rst]) 描述, “<=” 赋值
  - 边沿敏感变量表中避免出现除时钟和复位外的其他信号
  - 时钟信号避免出现在语句块内

变量类型问题

- 使用**always**语句描述的变量，务必声明为**reg**类型（声明为**reg**类型的变量，综合后不一定生成寄存器）！
- 使用**assign**语句描述的变量，应声明为**wire**类型
- **initial**或**always**语句中被赋值的变量，未定义直接使用的变量默认为**net**类型的标量，向量变量必须先定义后使用
- 同一进程中尽量在一个**if**或**case**语句块中对一个变量赋值，否则后边的赋值会覆盖前面的赋值，可能导致逻辑上的问题（特例：组合逻辑描述时，为避免形成锁存器而在开始给变量赋初值）

示例：变量类型

```
wire [7:0]  a, b;
reg [7:0]  r1, r2, r3;
```

```
always @(*)        // (en, a, b)
    if (en)  r1 = a;
    else  r1= b;
```

```
always @(en, a)   // @(*)
    if (en)  r2 = a;
```

```
always @(posedge  clk)
    if (en)  r3 = a;
```

组合电路：
- 敏感变量不要遗漏
- 所有条件分支均有赋值
- 不能含有反馈，如r1=r1+1

锁存器：尽量避免使用

寄存器：必有触发时钟

多驱动与多重时钟问题

- **多驱动问题**
  - 模块中所有的assign和always块都是并行执行的，不要在多个并行执行体中对同一变量赋值
- **多重时钟问题**
  - 不能采用行为描述方式来综合实现多个时钟或多个边沿驱动的触发器，例如

    always @(posedge clka, posedge clkb)

    always @(posedge clk, negedge clk)

参数化模块

module 模块名 #(parameter 参数声明) (端口声明);

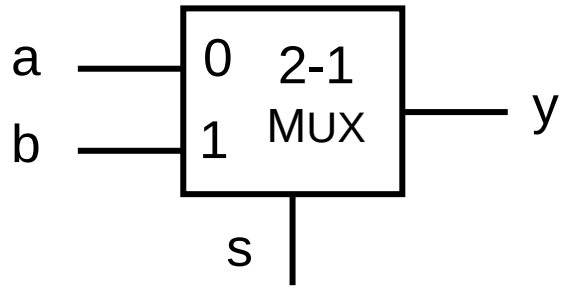
变量声明;

逻辑功能描述;

endmodule

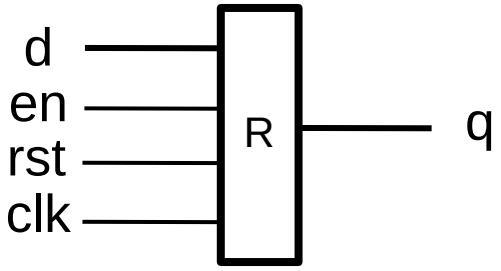
示例：MUX2

```
module mux2                          //模块名：mux2
    #(parameter WIDTH = 32)          //参数声明：数据宽度
    (output [WIDTH-1:0] y,           //端口声明：输出数据
    input [WIDTH-1:0] a, b,          //两路输入数据
    input s                          //数据选择控制
    );
assign y = s? b : a;                 //逻辑功能描述
endmodule
```



示例：寄存器

```
module  register
   #(parameter WIDTH = 32,
   RST_VALUE = 0)
   (input  clk, rst, en,
   input  [WIDTH-1 : 0]  d,
   output reg  [WIDTH-1 : 0] q);

always @(posedge clk, posedge rst)
   if (rst) q <= RST_VALUE;
   else  if (en)
      q <= d;

endmodule
```



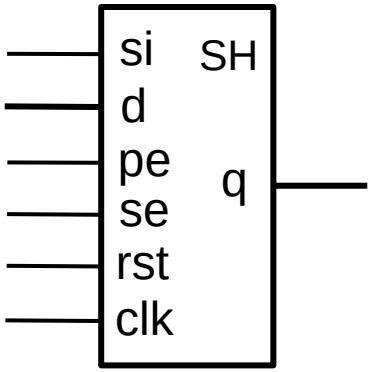
- d, q：输入、输出数据
- clk, rst, en：时钟、复位、使能

寄存器功能表

rst	clk	en	q	功能
1	x	x	0	复位
0	↑	1	d	置数
0	↑	0	q	保持

示例：移位寄存器

```
module  shifter
    #(parameter N = 8,
    RST_VALUE = {N{1'b0}})
    (input  clk, rst, pe, se,
    input  [N-1: 0]  d,
    output reg  [N-1: 0] q);

always @(posedge clk, posedge rst)
    if (rst) q <= RST_VALUE;
    else if (pe) q <= d;
    else if (se) q <= {si, q[N-1: 1]};

endmodule
```

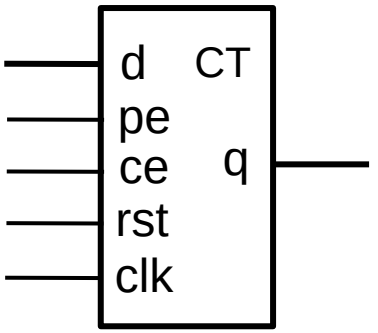


移位寄存器功能表

rst	clk	pe	se	功能
1	x	x	x	复位
0	↑	1	x	置数
0	↑	0	1	右移
0	↑	0	0	保持

示例：计数器

```
module  counter
    #(parameter N = 8,
    RST_VALUE = {N{1'b1}})
    (input  clk, rst, pe, ce,
    input  [N-1: 0]  d,
    output reg  [N-1: 0] q);

always @(posedge clk, posedge rst)
    if (rst) q <= RST_VALUE;
    else if (pe) q <= d;
    else if (ce) q <= q-1;

endmodule
```



递减计数器功能表

rst	clk	pe	ce	功能
1	x	x	x	复位
0	↑	1	x	置数
0	↑	0	1	计数
0	↑	0	0	保持

模块实例化

- **模块实例化语句格式：**

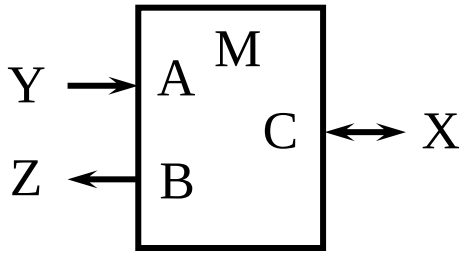
  module_name #(parameter_map) instance_name (port_map);

- **端口映射方式：基于位置或者基于名字，不可混合使用**
  - 位置映射：按模块中端口定义的顺序传递

    例如：模块定义为 moduel M(A, B, C);

    M M1(Y, Z, X);　　//顺序很重要
  - 名字映射：.PortName (value)

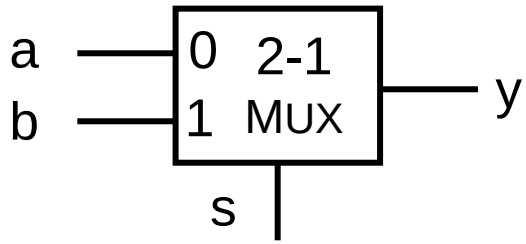
    M M1(.B(Z), .C(X), .A(Y));　// 顺序无关

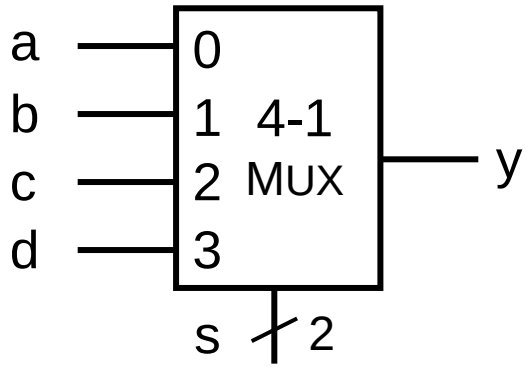


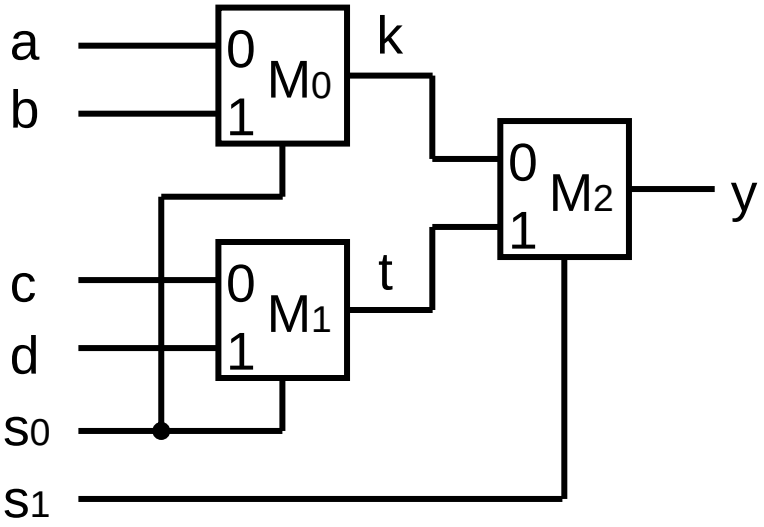
- **参数的映射方法类似**

示例：MUX4_8

```
module mux4_8
  (output [7:0] y,
  input [7:0] a, b, c, d,
  input [1:0] s
  );
wire [7:0] k, t;
//位置映射
mux2 #(7, 0) M0 (k, a, b, s[0]);
mux2 #(7) M1 (t, c, d, s[0]);
//名字映射
mux2 #(.MSB(7)) M2 (.s(s[1]), .a(k),
     .b(t), .y(y));
endmodule
```







仿真时钟

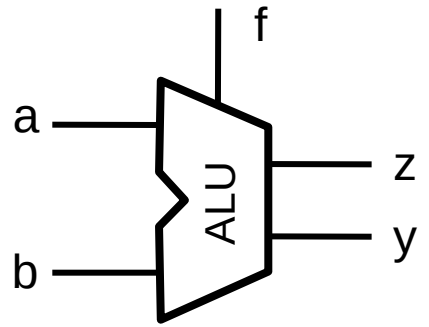
```
reg clk;
// 时钟周期和个数
parameter CYCLE = 10, Number = 20;

initial  begin
  clk = 0;
  repeat  (2* Number ）//或者 forever
     # CYCLE/2  clk = ~ clk;
end
```

- **initial和#仅用于仿真，不会产生实际硬件电路**

实验内容

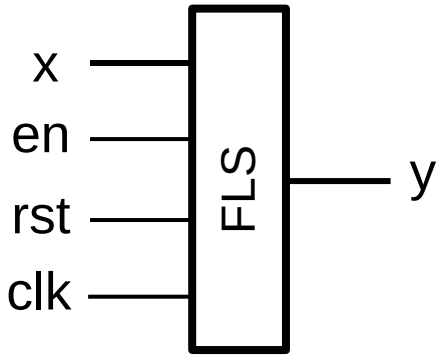
1. 算术逻辑单元（ALU）

- f：操作功能，加、减、与、或、异或等运算
- a, b：操作数，对于减运算，a是被减数
- y：运算结果，和、差 ……
- z：零标志，1--运算结果为零



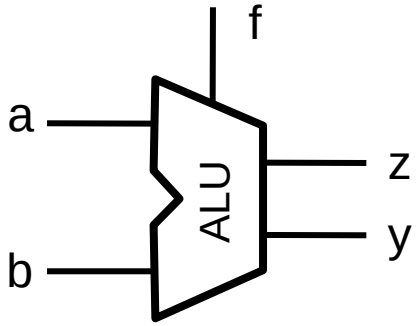
2. ALU应用：计算斐波那契—卢卡斯数列（Fibonacci Lucas Series）

- x：输入数列初始项
- en：输入和输出使能
- y：输出数列
- clk, rst：时钟，复位信号



ALU模块

```
module alu #(
    parameter WIDTH = 32          //数据宽度
)(
    input [WIDTH-1] a, b,         //两操作数
    input [2:0] f,                //操作功能
    output [WIDTH-1:0] y,         //运算结果
    output z                      //零标志
);
```



ALU 模块功能表

f	y	z
000	a + b	*
001	a - b	*
010	a & b	*
011	a | b	*
100	a ^ b	*
其他	0	1

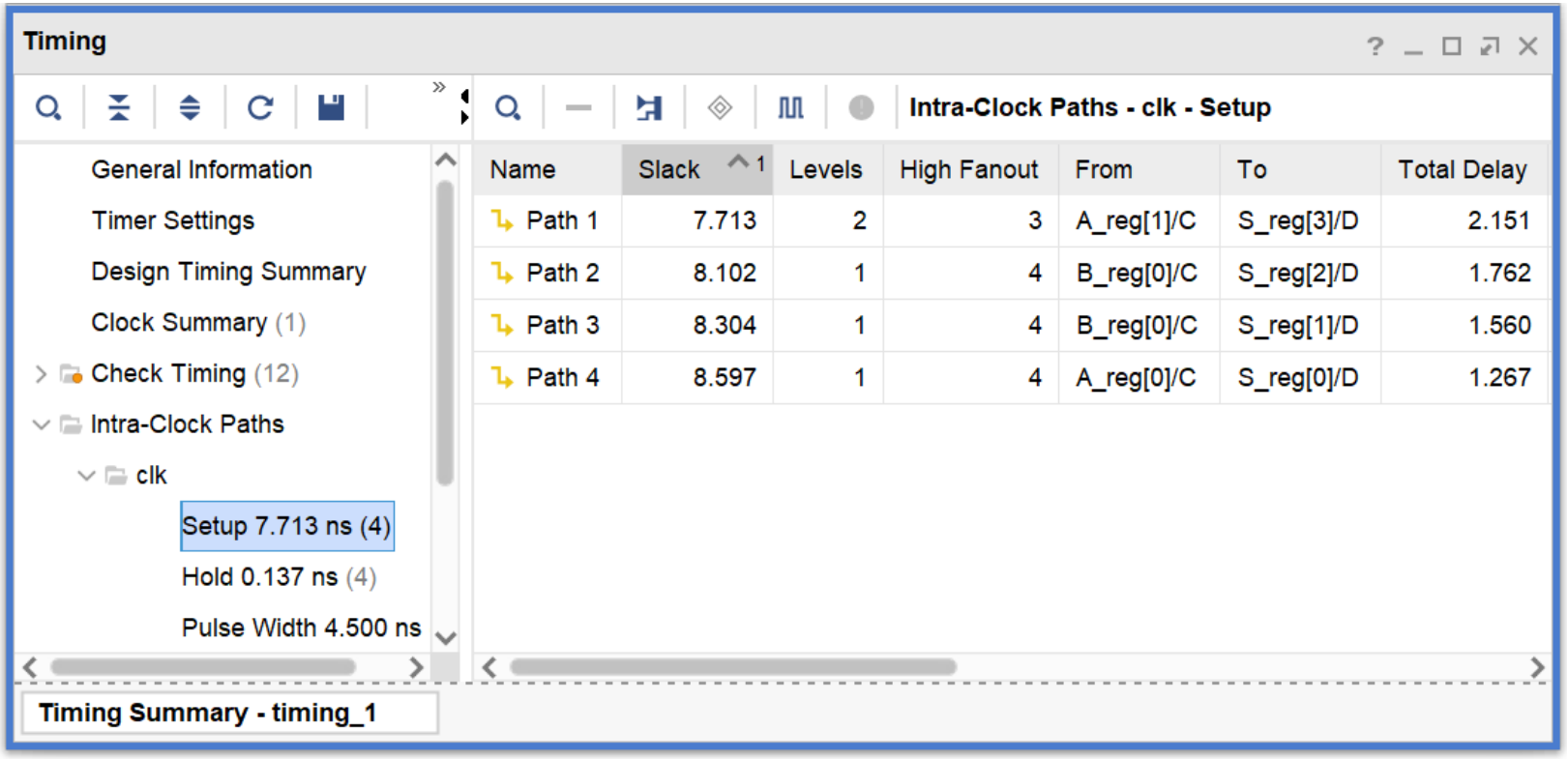
* 表示根据运算结果设置

ALU模块电路资源

- **查看Vivado生成电路**
  - RTL电路：Flow Navigator >> RTL Analysys >> Open Elaborated Design >> Schematic
  - 综合电路：Flow Navigator >> Synthesis >> Open Synthesized Design >> Schematic
- **查看电路资源使用情况**
  - 综合电路：Flow Navigator >> Synthesis >> Open Synthesized Design >> Report Utilization

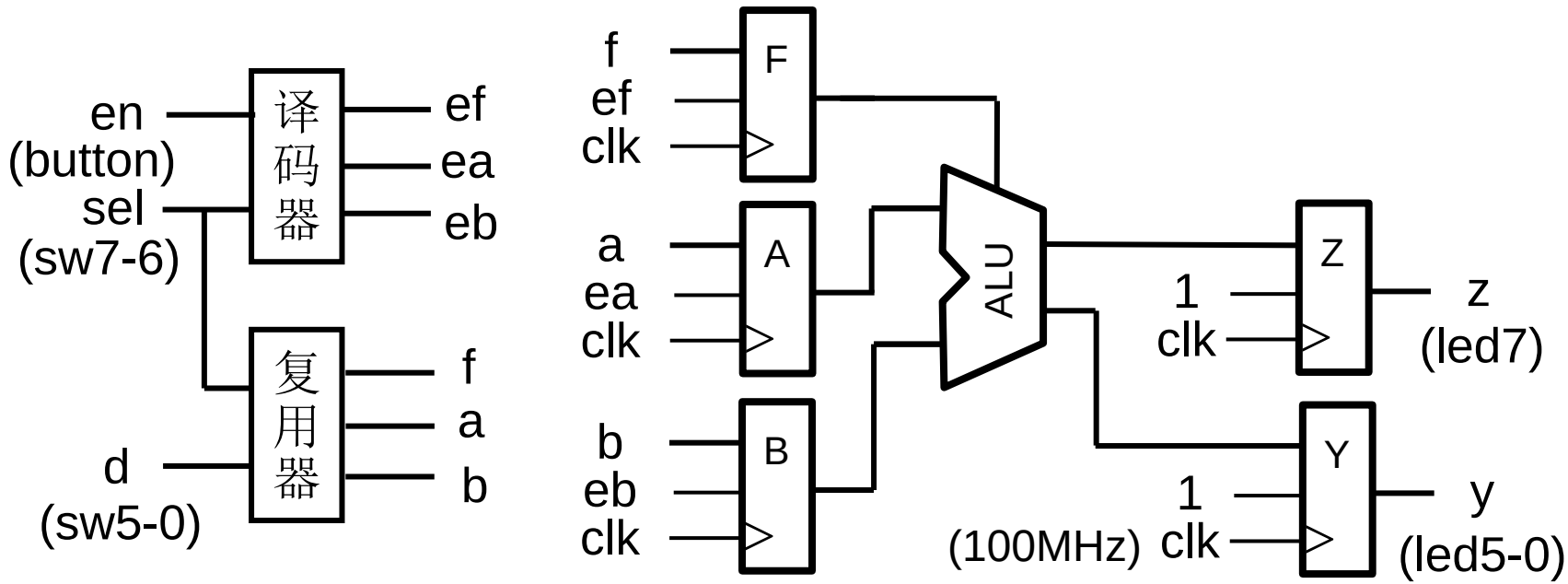
ALU模块电路性能

- 查看综合电路性能
  - Flow Navigator >> Synthesis >> Open Synthesized Design >> Report Timing Summary



ALU模块下载测试

- **ALU操作数(a, b, 6位)和功能(f, 3位)复用开关输入d (sw5~0)，通过选择sel (sw7~6)和使能en (buton)，分时存入寄存器 (F, A, B)**

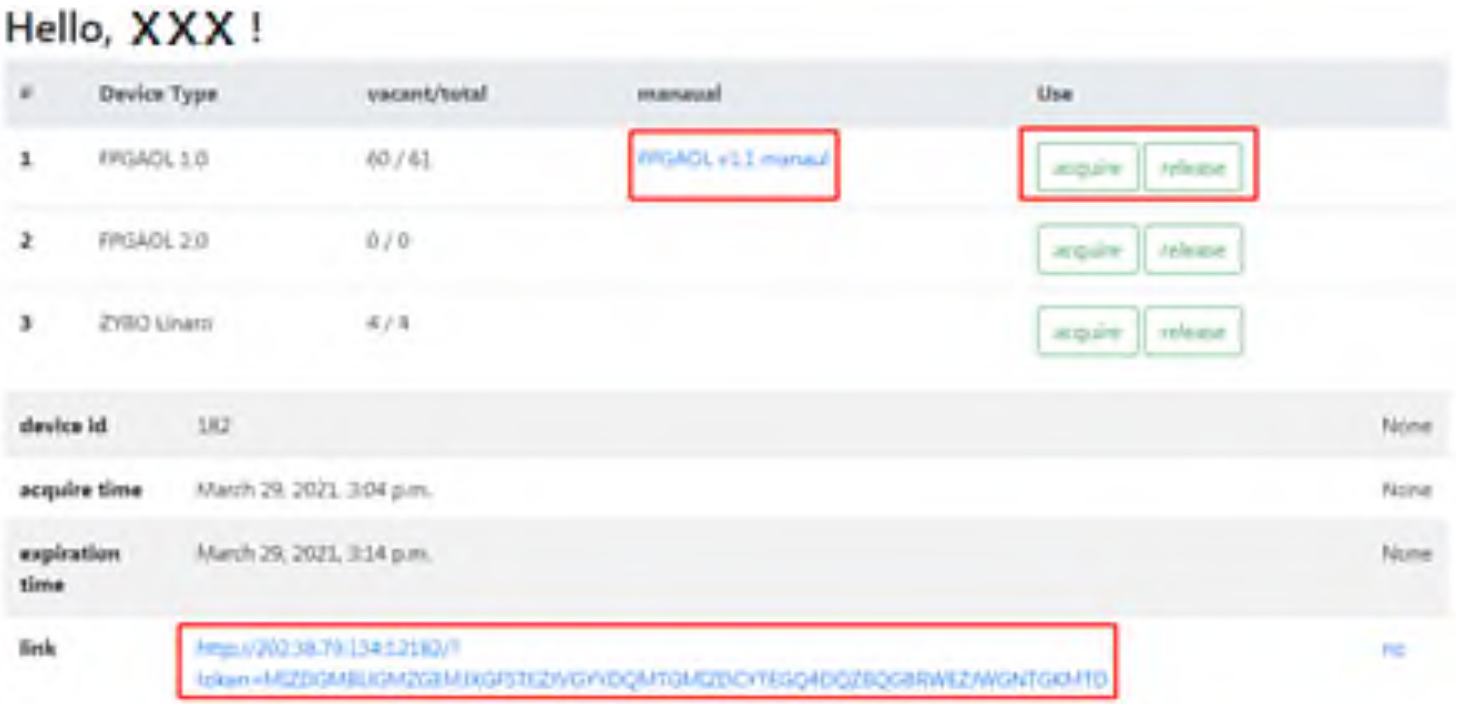


FPGAOL测试

- 登录平台网站：**fpgaol.ustc.edu.cn**，使用统一身份认证登录，或者直接以游客身份登录

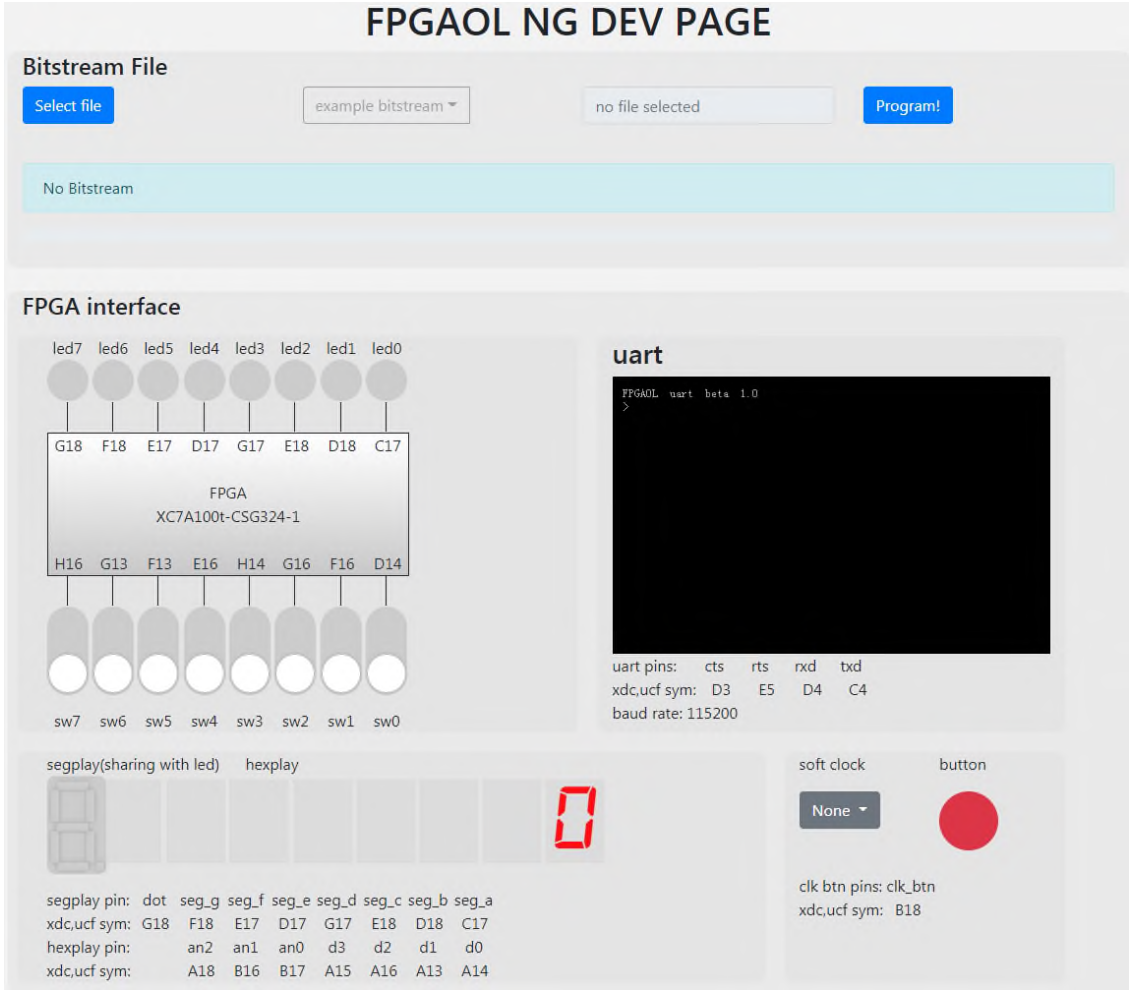
FPGAOL测试 (续1)

- **设备获取：点击“acquire”按钮获取一个FPGA结点**
  - 成功后在下方link栏将显示链接，通过链接进入设备操作界面
  - 默认使用时长10分钟(自动释放结点，点击“release”按钮手动释放)



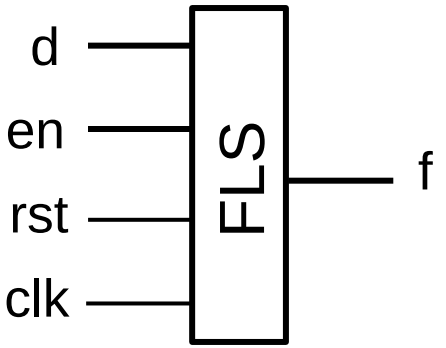
FPGAOL测试 (续2)

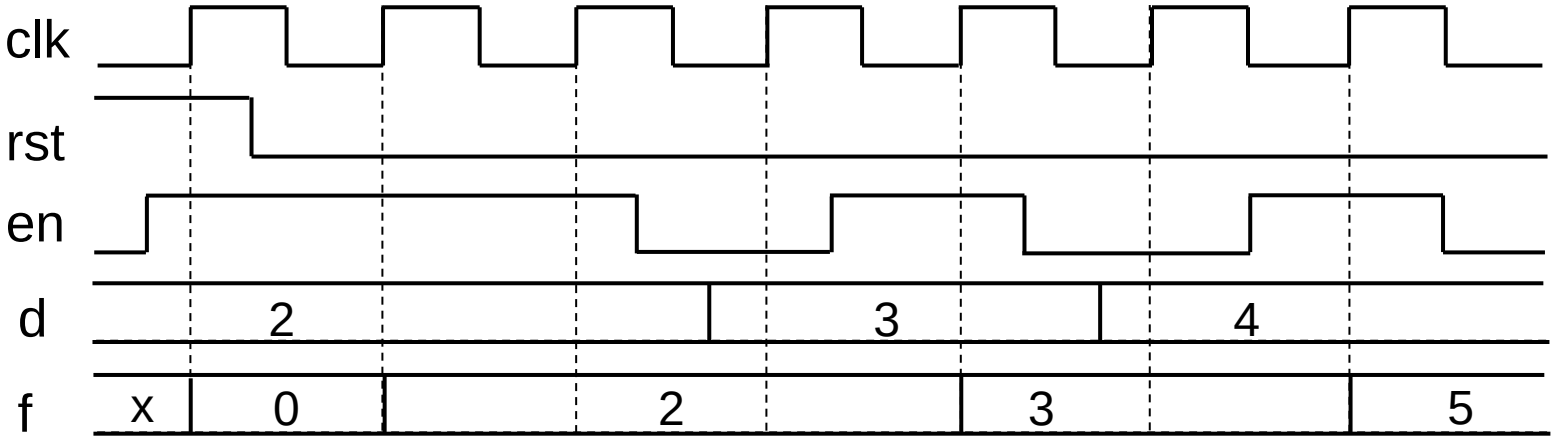
- 烧写FPGA：点击“Select file”按钮，选择需要烧写的bit文件，点击“Program！”



FLS模块

- **fn = fn-2 + fn-1　(n > 1)**
- **复位有效时，f = 0**
- **正常工作时，fn = f0、f1、f2、f3 ……（f0 = d0, f1 = d1）**

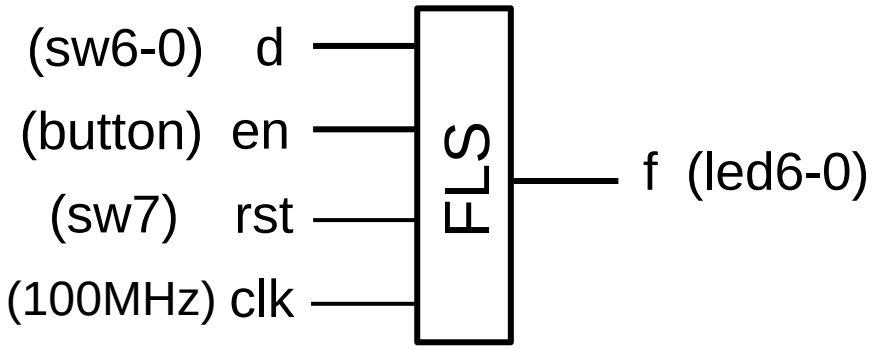




FLS模块 (续)

- **FLS模块端口定义如下：**

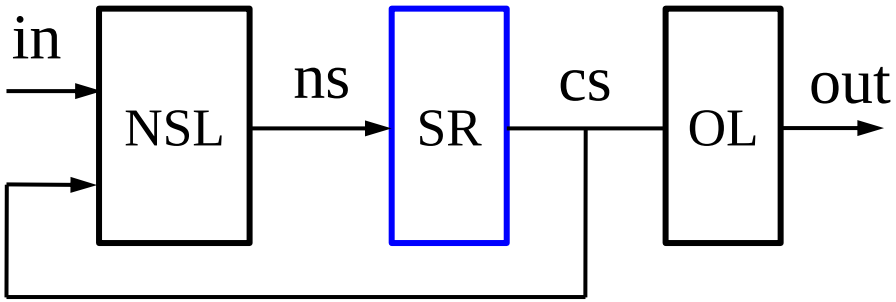
```
module  fls (
  input  clk, rst, en,
  input  [6:0]  d,
  output [6:0]  f
);
```



- **逻辑设计要求**
  - 数据通路：结构化
  - 控制器：Moore型FSM

FSM描述模式

```
// 描述CS
    always  @(posedge clk)
        if (rst)   cs <= S0;  //同步复位
        else   cs <= ns;

// 描述NS
    always  @* begin
        ns = cs;      //默认赋值
        case (cs)
            S0: begin
                ……
            end
            ……
        endcase
    end
```



实验步骤

1. 完成ALU模块的逻辑设计和仿真
2. 查看32位ALU的RTL和综合电路图，以及综合电路资源和时间性能报告
3. 完成6位ALU的下载测试，并查看RTL电路图，以及实现电路资源和时间性能报告
4. 完成FLS的逻辑设计、仿真和下载测试

The End