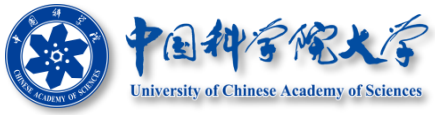
B0911007Y-01/02　2021-2022学年春季学期

计算机组成原理实验

用Verilog描述组合逻辑

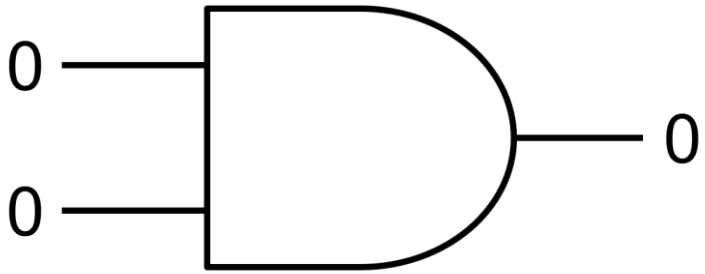
陈欲晓 刘士祺

2022年3月18日



复习：组合逻辑电路的定义

- 输出状态只取决于同一时刻各输入状态的组合



- 输入信号的变化瞬时影响输出信号
  - 不涉及同步，不需要时钟信号
- 电路内部不存储状态信息
  - 当前输出不依赖于先前的状态

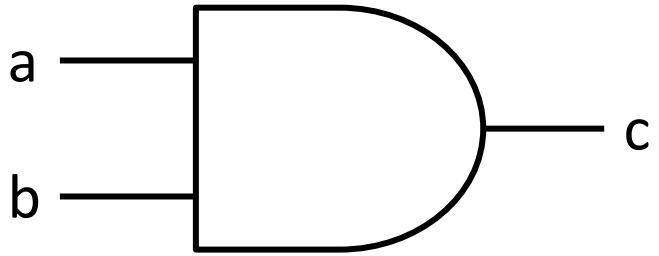
Verilog里的组合逻辑（1）

✓ - 推荐 ⚠ - 慎用 ✗ - 错误

- 使用assign语句（推荐）

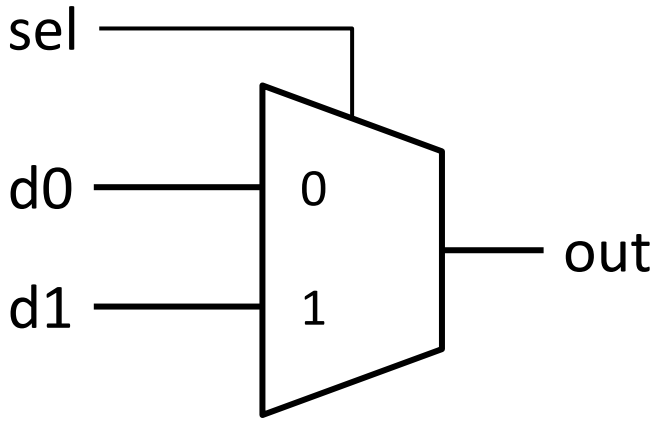
与运算

✓
```
wire [7:0] a, b, c;
assign c = a & b;
```



二路选择器

✓
```
wire sel;
wire [15:0] out, d0, d1;
assign out = sel ? d1 : d0;
```

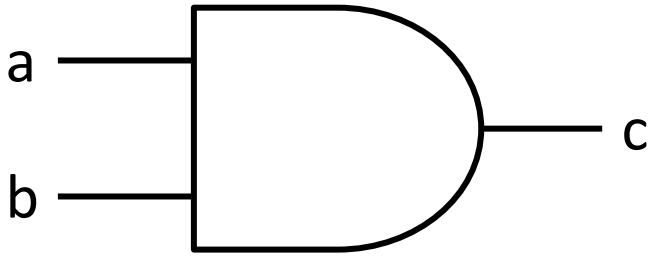


- 不易出错（相比于always语句）
- 电路结构清晰直观

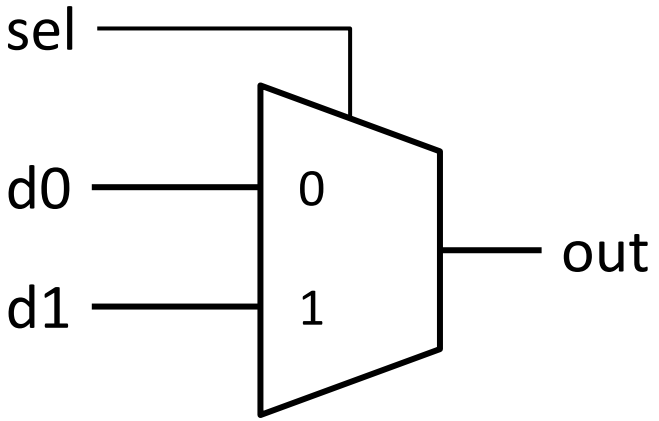
Verilog里的组合逻辑（2）

- 使用always语句（仅在特定情形下使用）
  - 注意always块内仅能对reg型变量赋值（即使是组合逻辑）

```
wire [7:0] a, b;
reg [7:0] c;
always @(*) begin
    c = a & b;
end
```




```
wire sel;
wire [15:0] d0, d1;
reg [15:0] out;
always @(*) begin
    if (sel)
        out = d1;
    else
        out = d0;
end
```



Verilog里的组合逻辑（3）

- 用always语句描述状态机次态
- 赋值务必覆盖所有可能情况
  - 任何条件下都有赋值语句
  - if语句要有else分支赋值
  - case语句要有default分支赋值
- 否则会生成锁存器（latch）
  - 变量在缺失分支下保持不变（Verilog语义）→生成锁存器
  - EDA工具难以正确约束时序
  - 易导致仿真和FPGA运行结果不一致

```
//当前状态state与次态ns(next state)
reg [2:0] state, ns;
//状态采用独热码（one-hot）编码
localparam S_IDLE = 3'b001;
localparam S_REQ  = 3'b010;
localparam S_WAIT = 3'b100;
//三段式状态机第二段
always @(*) begin
  case (state)
  S_IDLE:
    ns = req ? S_REQ : S_IDLE;
  S_REQ:
    ns = req_ok ? S_WAIT : S_REQ;
  S_WAIT:
    ns = resp ? S_IDLE : S_WAIT;
  default:
    ns = S_IDLE;
  endcase
end
```

✓

关于锁存器更多信息：https://www.runoob.com/w3cnote/verilog-latch.html

使用always语句的误区

```
always @(*) begin
  case (state)
  S_IDLE:
    if (start)
      ns = S_WORKING;
    else
      ns = S_IDLE;
  S_WORKING:
    if (complete)
      ns = S_IDLE;
    else
      ns = S_WORKING;
  endcase
end
```

✗

```
always @(*) begin
  case (state)
  S_IDLE:
    if (start)
      ns = S_WORKING;
    else
      ns = S_IDLE;
  S_WORKING:
    if (complete)
      ns = S_IDLE;
    else
      ns = S_WORKING;
  default:
    ns = ns;
  endcase
end
```

✗

```
always @(*) begin
  case (state)
  S_IDLE:
    if (start)
      ns = S_WORKING;
    else
      ns = S_IDLE;
  S_WORKING:
    if (complete)
      ns = S_IDLE;
    else
      ns = S_WORKING;
  default:
    // 无关紧要的值
    ns = S_IDLE;
  endcase
end
```

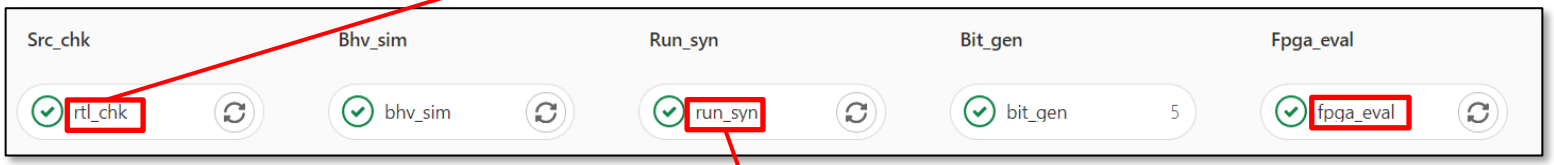
✓

- 变量在某些条件下保持不变是产生锁存器的根本原因！
- 次态ns在任何分支的赋值都不能依赖于其自身
  - 即使分支在运行过程中永远不会被用到

检查代码中的锁存器

①代码检查阶段排查不完整的if和case语句

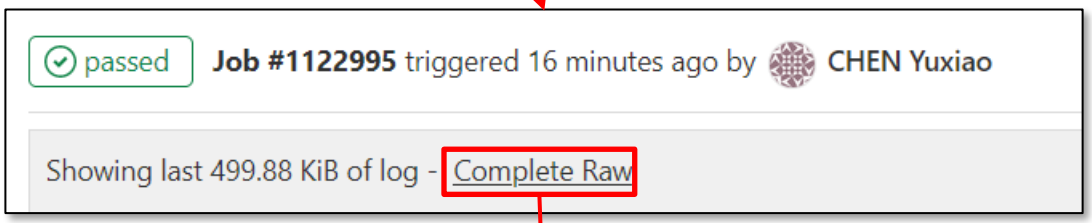
```
INFO: [Synth 8-155] case statement is not full and has no default
[/builds/imhcyx/cod-lab/fpga/design/ucas-cod/hardware/sources/alu/alu.v:18]
```



或者custom_cpu_chk（后续实验）

或者custom_run_syn（后续实验）

产生锁存器也可能通过FPGA测试




```
WARNING: [Synth 8-327] inferring latch for variable 'Result_reg'
[/builds/imhcyx/cod-lab/fpga/design/ucas-cod/hardware/sources/alu/alu.v:19]
```

②综合阶段排查锁存器（latch）

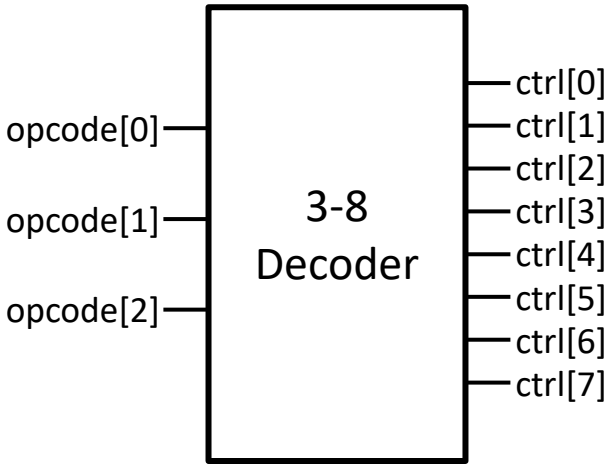
组合逻辑实例

- 译码器
- 选择器
- 加法器
- ALU

译码器

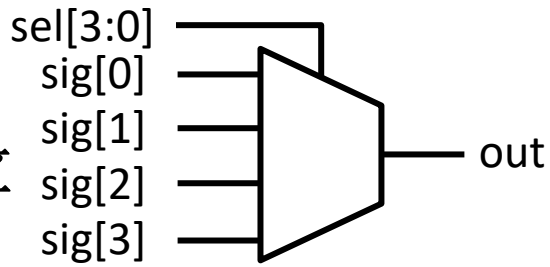
- 将操作码翻译为一系列控制信号（独热码）
  - ALUop，指令操作码，…
  - 先译码，再控制
- 例：3-8译码器

```
wire [2:0] opcode;
wire [7:0] ctrl;
assign ctrl[0] = opcode == 3'b000;
assign ctrl[1] = opcode == 3'b001;
assign ctrl[2] = opcode == 3'b010;
assign ctrl[3] = opcode == 3'b011;
assign ctrl[4] = opcode == 3'b100;
assign ctrl[5] = opcode == 3'b101;
assign ctrl[6] = opcode == 3'b110;
assign ctrl[7] = opcode == 3'b111;
```



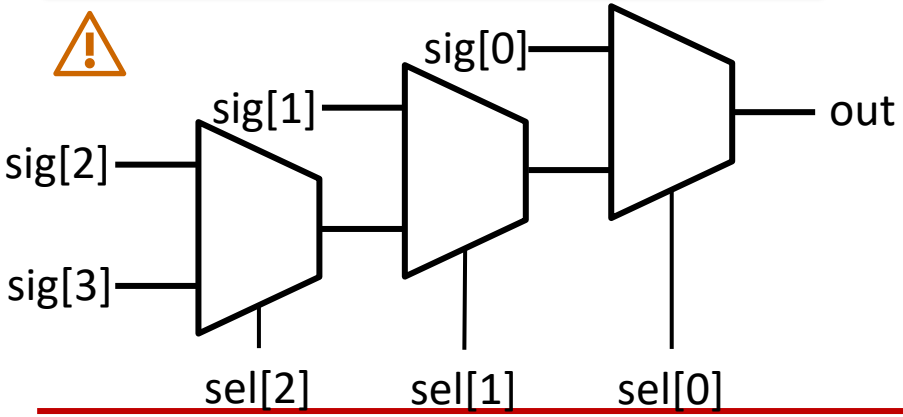
选择器

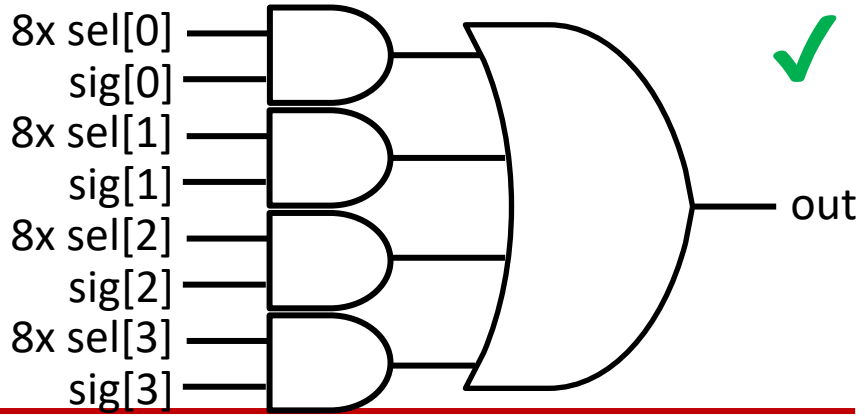
- 从多个信号中选择一个输出
- 如果选择信号是独热码，建议用与或运算
  - 逻辑门级数更少，延迟更低




```
wire [3:0] sel;
wire [7:0] sig [3:0];
wire [7:0] out;
assign out = sel[0] ? sig[0]
           : sel[1] ? sig[1]
           : sel[2] ? sig[2]
           : sig[3];
```

```
wire [3:0] sel;
wire [7:0] sig [3:0];
wire [7:0] out;
assign out = {8{sel[0]}} & sig[0]
           | {8{sel[1]}} & sig[1]
           | {8{sel[2]}} & sig[2]
           | {8{sel[3]}} & sig[3];
```

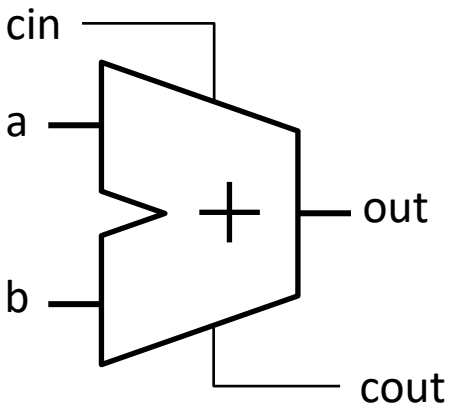




加法器

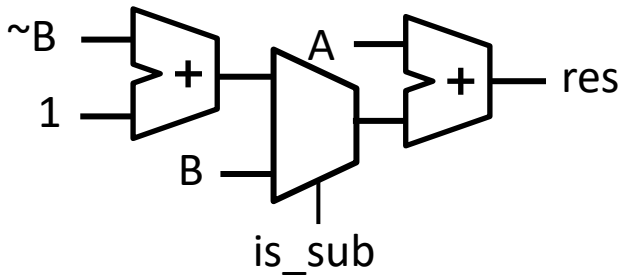
- 带进位输入和进位输出的加法器

```
wire [3:0] a, b, out;
wire cin, cout;
assign {cout, out} = a + b + cin;
```



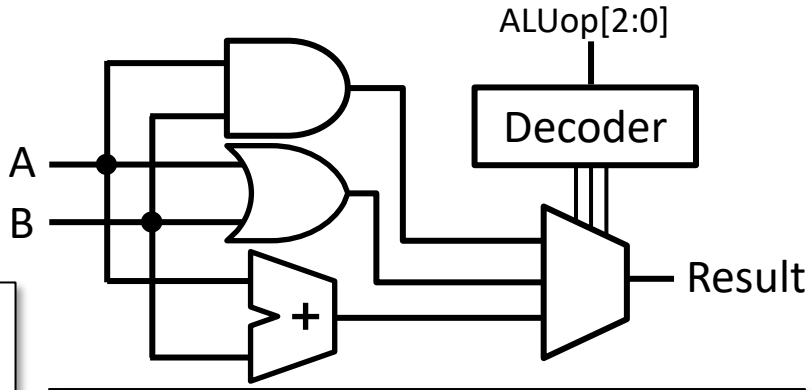
- 上述写法可认为是一个加法器
  - 多位+多位+1位进位输入（N+N+1）
- 其余写法每个“+”号对应一个加法器
  - 例：用“一套加法器”同时实现加减法

```
//input [31:0] A, B
wire is_sub;
wire [31:0] B2 = is_sub ? ~B+1 : B;
wire [31:0] res = A+B2;
```



简单ALU

- 基于实验项目1简化
  - 4-bit的与、或、加法运算
  - 只输出Result




```
`define ALUOP_AND 3'b000
`define ALUOP_OR  3'b001
`define ALUOP_ADD 3'b010

module alu(
  input [3:0] A,
  input [3:0] B,
  input [2:0] ALUop,
  output [3:0] Result
);
// 控制信号译码
wire op_and = ALUop == `ALUOP_AND;
wire op_or  = ALUop == `ALUOP_OR;
wire op_add = ALUop == `ALUOP_ADD;
```

```
// 计算
wire [3:0] and_res = A & B;
wire [3:0] or_res  = A | B;
wire [3:0] add_res = A + B;

// 选择结果
assign Result =
  {4{op_and}} & and_res |
  {4{op_or }} & or_res  |
  {4{op_add}} & add_res ;

endmodule
```

思考：如何只用一个加法器实现ADD、SUB、SLT三种运算？

总结

- **组合逻辑电路的定义**
- **Verilog组合逻辑写法**
  - assign语句：多数情况下使用
  - always语句：仅用于状态机次态
    - 分支要写全
    - 不依赖自身
- **组合逻辑实例**
  - 译码器：先译码，再控制
  - 选择器：多路选择用与或
  - 加法器：N+N+1
  - ALU：译码器+运算器+选择器

Q & A ?


中国科学院大学
University of Chinese Academy of Sciences
ICT
中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY,CAS