

# 静态时序分析与逻辑综合

## STA & SYN

韩炳晋

2025/06/15

# Sec. 1 静态时序分析（Static Timing Analysis, STA）

STA是一种在数字IC设计中用于**验证电路时序正确性**的重要方法。

在YSYX的yosys-sta中，利用iSTA工具，你会得到这样的报告：

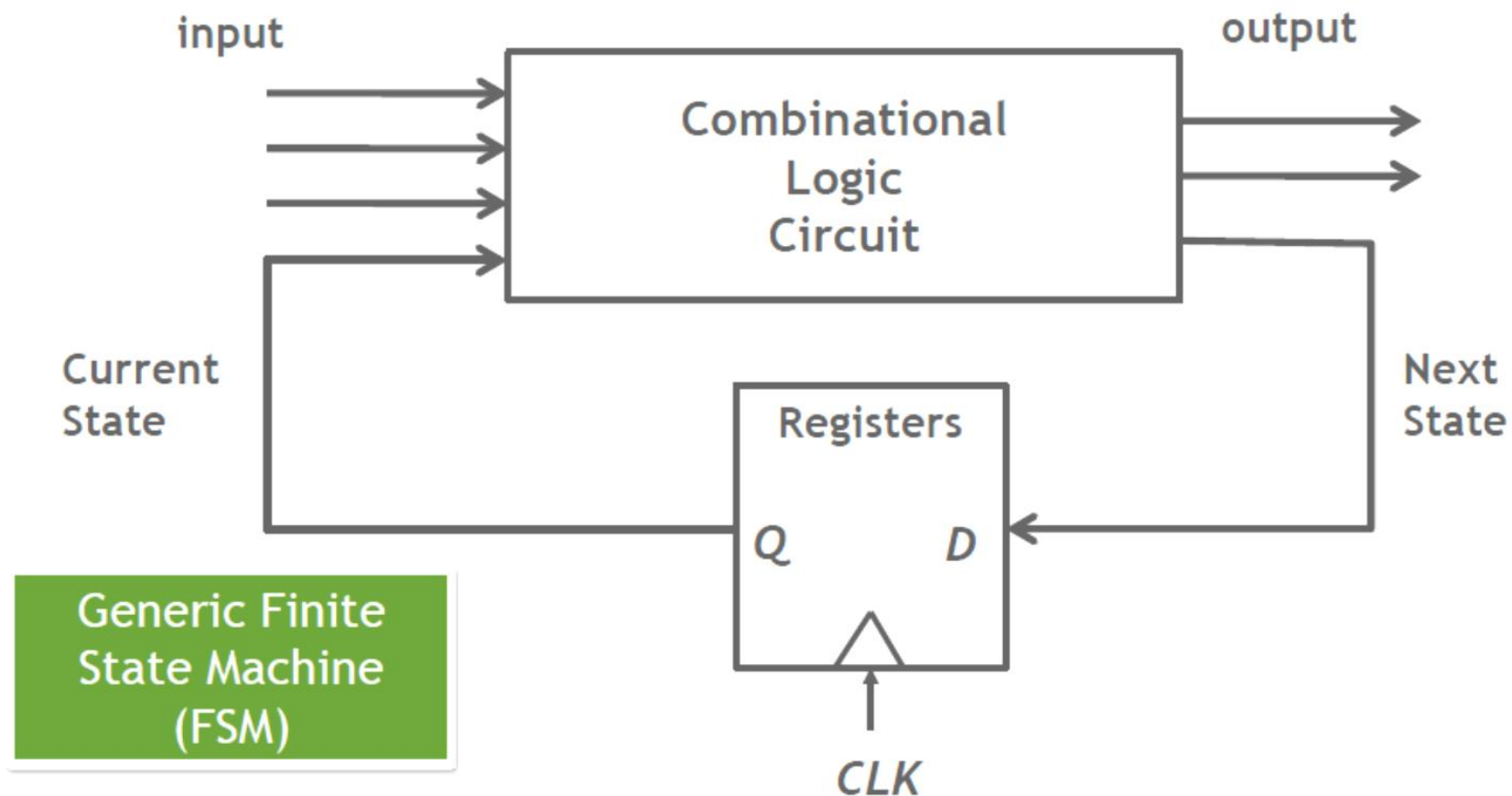
...

...							
_30813_:D (DFF_X1)		0.001	0.000	0.006		0.000	1.373f
clock (port)		1.941	0.000	0.000		0.000	0.000r
clock (clock net)	2044				NA		
_30813_:CK (DFF_X1)		0.001	0.000	0.000		0.000	0.000r
clock core_clock (rise edge)						1.123	1.123
clock network delay (ideal)						0.000	1.123
_30813_:CK (DFF_X1)							1.123r
library setup time						-0.039	1.084
clock reconvergence pessimism						0.000	1.084
path cell delay							1.373(100.000%)
path net delay							0.000(0.000%)
data require time							1.084
data arrival time							1.373
slack (VIOLATED)							-0.289

-----

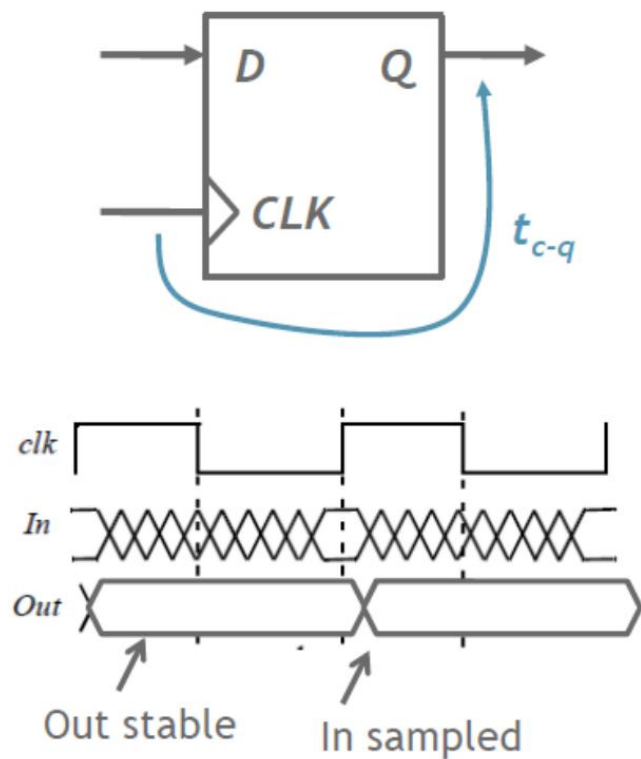
它的原理是什么，如何实现？

# 静态时序分析-时序逻辑



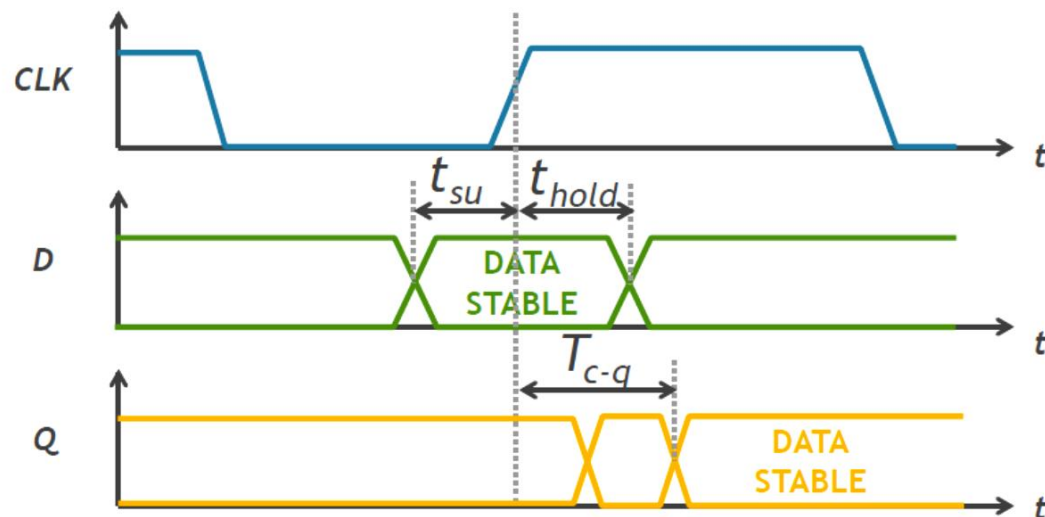
时序逻辑 = 组合逻辑 + 寄存器（触发器）  
目前不考虑带锁存器的时序逻辑

# 静态时序分析-触发器的Setup/Hold Time以及传输延迟



上升沿采样，其余时间保持稳定

术语	解释
建立(Setup)时间	时钟沿到达之前，数据信号 (D) 必须保持有效的时间。
保持(Hold)时间	时钟沿到达之后，数据信号 (D) 必须继续保持有效的时间。
传输延迟	D输入经过时钟沿后，传播到Q输出所需的时间。



# 静态时序分析-标准单元库中的Setup/Hold以及传输延迟（以Nangate45为例）

RTFM:

Description	Pos. edge Flip-Flop with drive strength X1
Strength	1
Cell Area	4.522 um <sup>2</sup>
Equation	Q = "(D)" QN = "(!D)"
Clock	CK
Type	Sequential
Input	D
Output	Q, QN
PG Pins	VDD (primary_power), VSS (primary_ground)



State Table					
CK	D	IQ <sub>(int)</sub>	IQN <sub>(int)</sub>	Q	QN
R	L	-	-	L	H
R	H	-	-	H	L
F	-	L	H	L	H
F	-	H	L	H	L

Propagation Delay [ns]					
Input Transition [ns]		0.0009		0.1462	
Load Capacitance [fF]		0.3656	60.73	0.3656	60.73
CK to Q	fall	0.05	0.11	0.06	0.12
	rise	0.05	0.13	0.06	0.14
CK to QN	fall	0.03	0.10	0.04	0.11
	rise	0.04	0.12	0.05	0.13

Constraints Time [ns]		
Setup CK to D	fall	0.06
	rise	0.04
Hold CK to D	fall	0.12
	rise	0.10

图片来源于：Nangate45的数据手册。

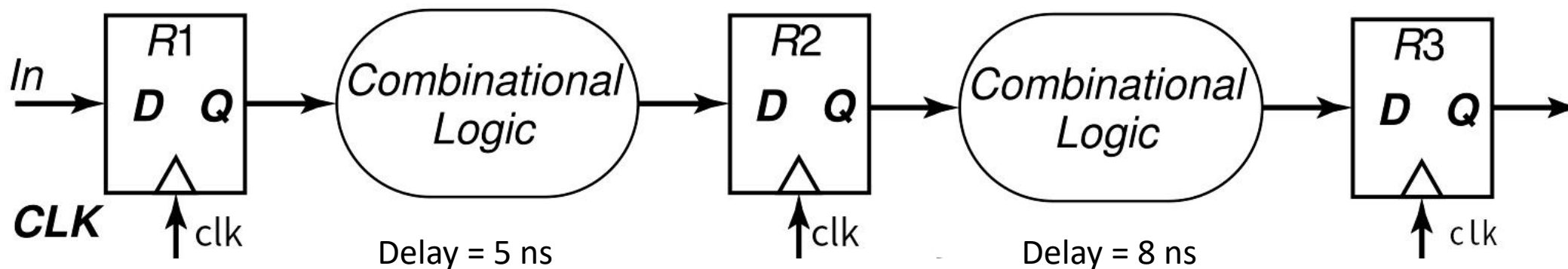
# 静态时序分析-标准单元库中的Setup/Hold以及传输延迟 (以Nangate45为例)

RTFSC: Liberty File: Nangate0penCellLibrary\_typical.lib

```
cell (DFF_X1) {
    ...
    pin (D) { // setup and hold time
        ...
        timing () {
            related_pin          : "CK";
            timing_type           : hold_rising;
            fall_constraint(Hold_3_3) {
                index_1 ("0.00117378,0.0449324,0.198535"); //input_net_transition
                index_2 ("0.00117378,0.0449324,0.198535"); //total_output_net_capacitance
                values ("0.001650,0.010970,0.010626", \
                    "0.004131,0.010748,0.006379", \
                    "0.143222,0.153005,0.144764");
            }
            rise_constraint(Hold_3_3) {
        ...
    pin (Q) { // propagation delay
        ...
        timing () {
        ...
            cell_fall(Timing_7_7) {
        ...
            cell_rise(Timing_7_7) {
```

Liberty文件格式可参考: Alan Mishchenko. Liberty Reference Manual. 2007.  
[https://people.eecs.berkeley.edu/~alanmi/publications/other/liberty07\\_03.pdf](https://people.eecs.berkeley.edu/~alanmi/publications/other/liberty07_03.pdf)

## 静态时序分析-关键路径(Critical Path)



不考虑Setup和寄存器传输延迟:

最大的 $T_{clk} = 8\text{ns}$ , 最高频率为 125MHz

Delay = 8ns的这条路径被称为关键路径

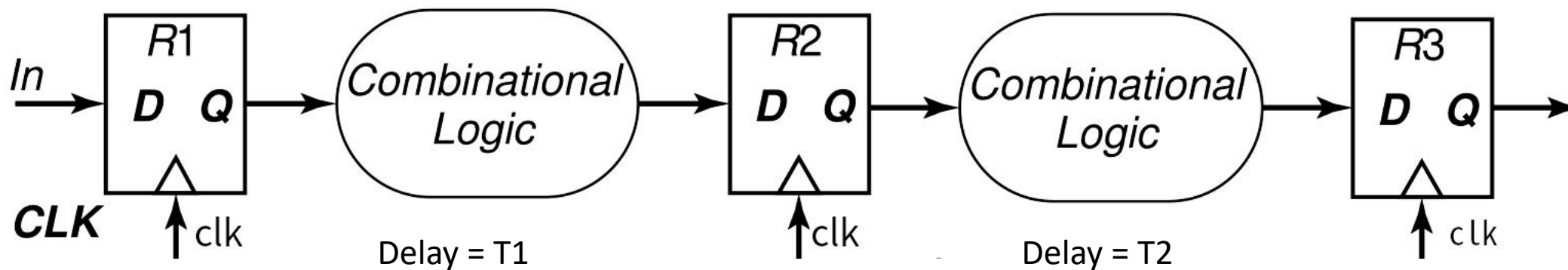
考虑Setup和寄存器传输延迟:

最大的 $T_{clk} = 8\text{ns} + T_{su} + T_{cq}$ , 最高频率变低

**Require Time =  $T_{clk} - T_{su}$**

**Arrival Time =  $T_{logic} + T_{cq}$**

## 静态时序分析-Hold Constraint



**Hold Time:** 时钟沿到达之后，数据信号（D）必须继续保持有效的时间。  
也就是说，Hold Time限制了组合逻辑的最小延迟。

$$T_{logic} > T_{hold} - T_{cq}$$



# 静态时序分析-读时序分析报告

Require Time = Tclk - Tsu


Arrival Time = Tlogic + Tcq

Point	Fanout	Capacitance	Resistance	Transition	Delta Delay	Incr	Path
..							
_31020_:CK (DFF_X1)		0.001	0.000	0.000		0.000	0.000r
_31020_:Q (DFF_X1)		0.006	0.000	0.017		0.094	0.094r
cpu_npc.exu.alu_GEN[3] (net)	6				0.000		
_32439_:A (BUF_X1)		0.001	0.000	0.017		0.000	0.094r
_32439_:Z (BUF_X1)		0.056	0.000	0.133		0.158	0.253r
_03454_ (net)	29				0.000		
..							
_30813_:D (DFF_X1)		0.001	0.000	0.006		0.000	1.373f
clock (port)		1.941	0.000	0.000		0.000	0.000r
clock (clock net)	2044				NA		
_30813_:CK (DFF_X1)		0.001	0.000	0.000		0.000	0.000r
clock core_clock (rise edge)						1.123	1.123
clock network delay (ideal)						0.000	1.123
_30813_:CK (DFF_X1)							1.123r
library setup time						-0.039	1.084
clock reconvergence pessimism						0.000	1.084
path cell delay							1.373(100.000%)
path net delay							0.000(0.000%)
data require time							1.084
data arrival time							1.373
slack (VIOLATED)							-0.289

## Sec. 2 逻辑综合

逻辑综合是指将电路的行为描述（如RTL级Verilog代码）转换为可实现的门级电路网表的过程。  
一生一芯中使用Yosys+ABC完成了综合。

```
assign c = a & b;  
assign y = ~c;
```



```
AND2_X1 _001_(.Y(c), .A(a), .B(b));  
INV_X1 _002_(.Y(y), .A(c), .B(b));
```

# 逻辑综合-预备知识-AIG

AIG 是仅由二输入与门及隐式反相边构成的有向无环图（DAG）

例如，一个异或运算 ( $z = x \oplus y$ ) 可以用如下的 AIG (图 2-4) 表示。其中，方形节点表示输入和输出，圆形节点表示与门，虚线箭头表示信号取反。由于 AIG 的结构简单，广泛应用于逻辑综合中。

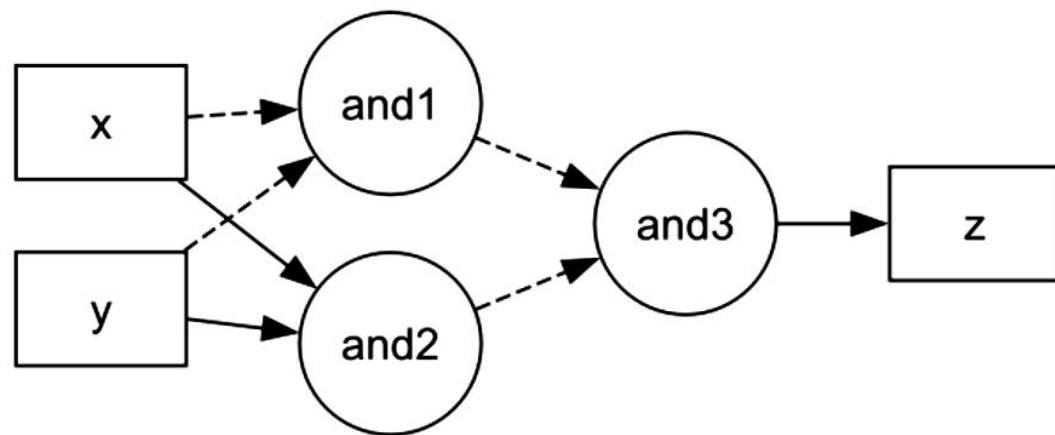


图 2-4 表示异或运算的与非图

# 逻辑综合-预备知识-Structural hashing (strashing)

结构哈希是一种消除结构冗余、提高图结构规范性的优化方法。

**它的基本思想是：**当向AIG中添加一个新的AND门节点时，如果该节点与现有节点在逻辑结构上是等价的（即有相同的输入，输入顺序可以交换），那么就复用现有节点，而不是新建一个功能完全一样的新节点。

不同的策略：

- One-level strashing
- Two-level strashing

```
Aig_Node * OperationAnd( Aig_Man * p, Aig_Node * n1, Aig_Node * n2 )
{
    Aig_Node * res, * cand, * temp; Aig_NodeArray * class;
    /** trivial cases **/
    if ( n1 == n2 )                return n1;
    if ( n1 == NOT(n2) )           return 0;
    if ( n1 == const )             return 0 or n2;
    if ( n2 == const )             return 0 or n1;
    if ( n1 < n2 ) { /** swap the arguments **/
        temp = n1; n1 = n2; n2 = temp;
    }
    /** one level structural hashing **/
    res = HashTableLookup( p->pTableStructure, n1, n2 );
    if ( res )                      return res;
    res = CreateNode( p, n1, n2 );
    HashTableAdd( p->pTableStructure, res );    return res;
}
```

一阶结构哈希的伪代码

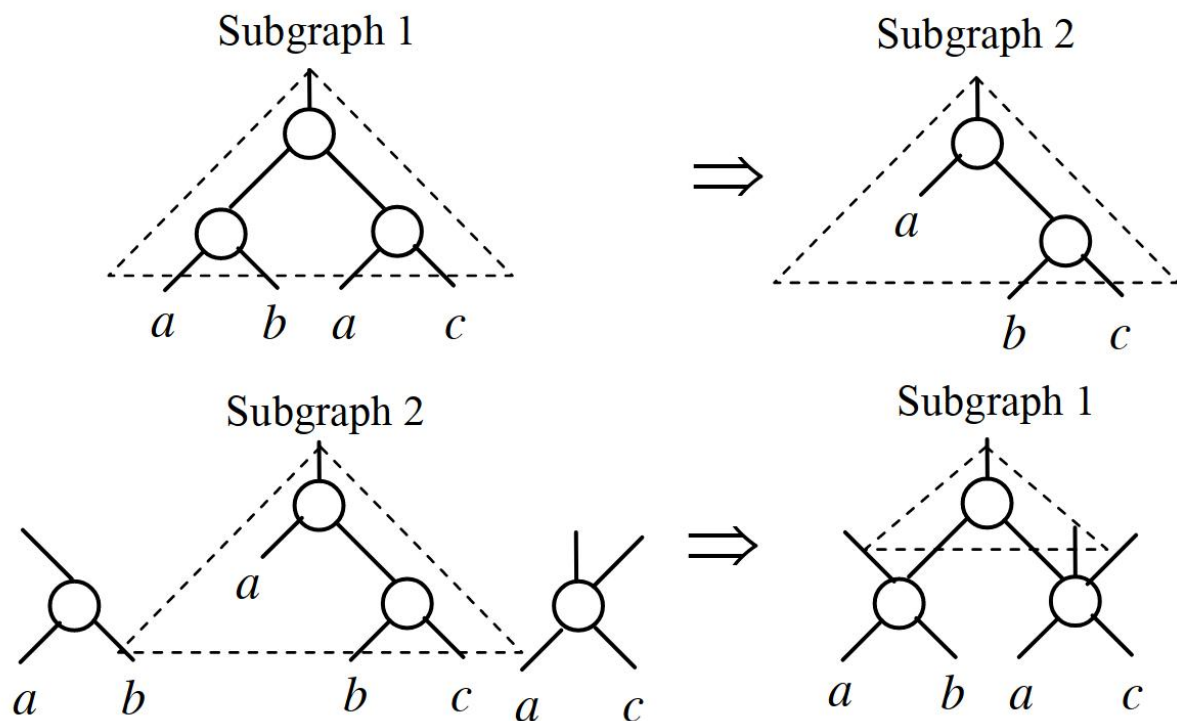
伪代码来源于：Mishchenko, A., Chatterjee, S., Jiang, R., Brayton, R., n.d. FRAIGs: A Unifying Representation for Logic Synthesis and Verification.

# 逻辑综合-Rewrite

Rewrite 是一种基于贪心算法的逻辑优化方法，可用于优化电路的面积和时序。

**核心思想：**

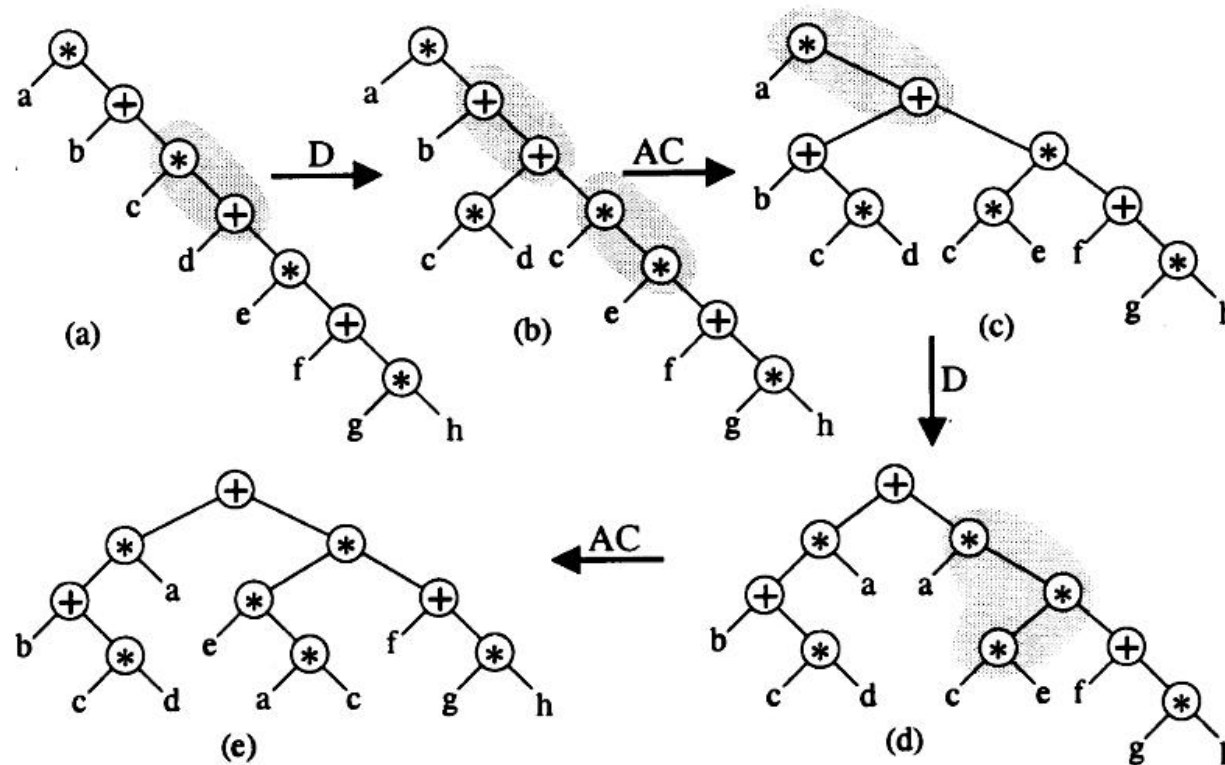
在电路网络中，识别小规模局部子图，查找等价但结构更优的替换模式，用以降低门数或减少电路深度。



**Figure 3.** Two cases of AIG rewriting of a node.

# 逻辑综合-Balance

Balance是在逻辑综合中通过应用结合律、交换律和分配律等代数变换，重新组织布尔表达式的树结构，以减少电路的逻辑层数。



ACD rules:

A: Associative

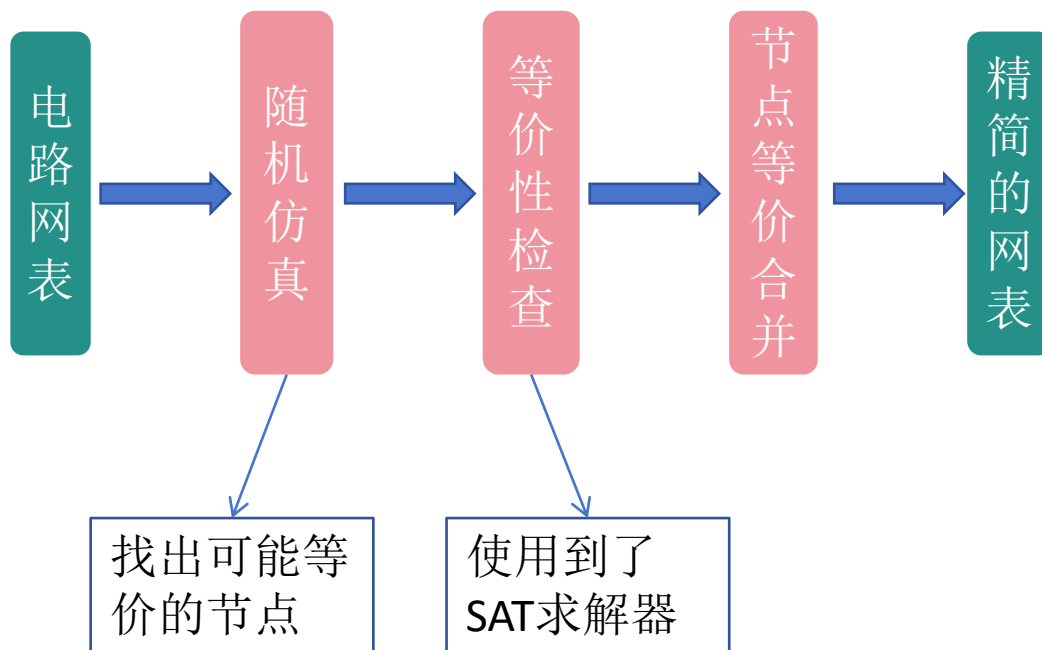
C: Commutative

D: Distributive

Fig. 9. Application of ACD rules to optimize performance.

# 逻辑综合-SAT-Sweeping

SAT-Sweeping旨在通过SAT（可满足性）判定方法，识别并合并电路中的功能等价节点，减少电路规模。

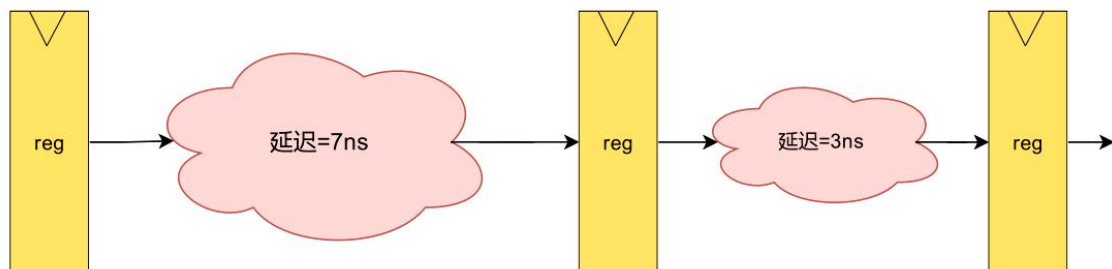


最早提出：Kuehlmann, A., 2004. Dynamic transition relation simplification for bounded property checking, in: IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.

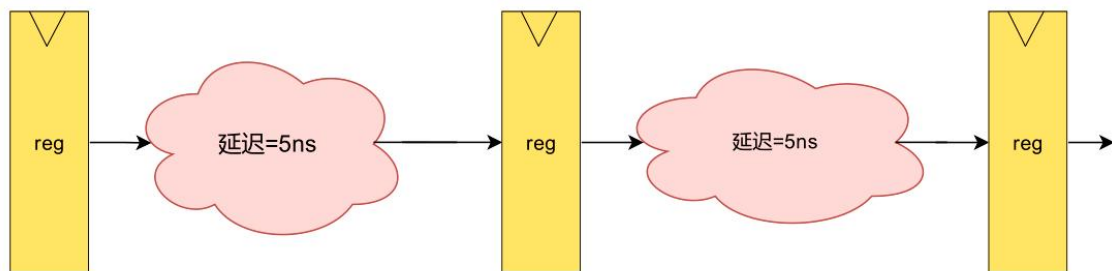
Mishchenko, A., Chatterjee, S., Jiang, R., Brayton, R., n.d. FRAIGs: A Unifying Representation for Logic Synthesis and Verification.

# 逻辑综合-Retiming

重定时是一项在保持电路功能等价的前提下，通过移动寄存器来改善电路性能的优化技术。其目标通常分为两类：最小化延迟与最小化面积

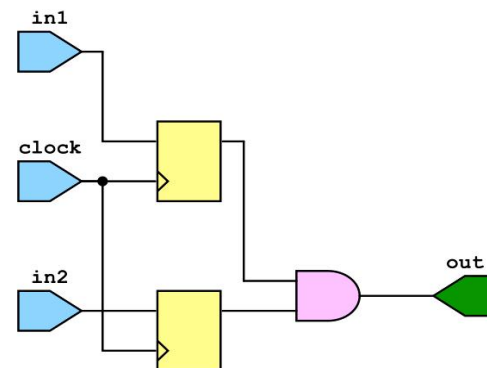


a) 重定时前，关键路径延时为 7ns，理想条件下最高频率为 143Mhz

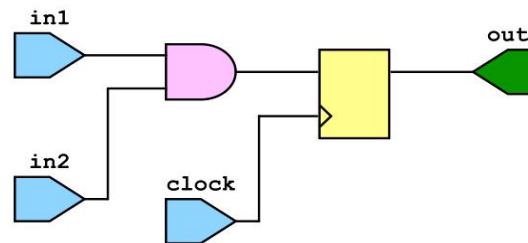


b) 重定时后，关键路径延时为 5ns，理想条件下最高频率为 200Mhz

最小化延时



a) 重定时前，电路有 2 个寄存器和 1 个与门

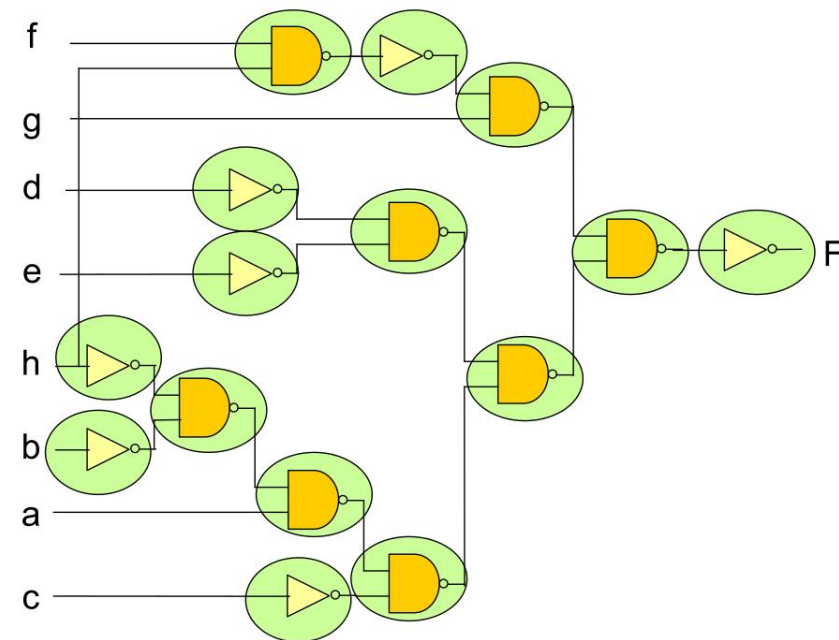
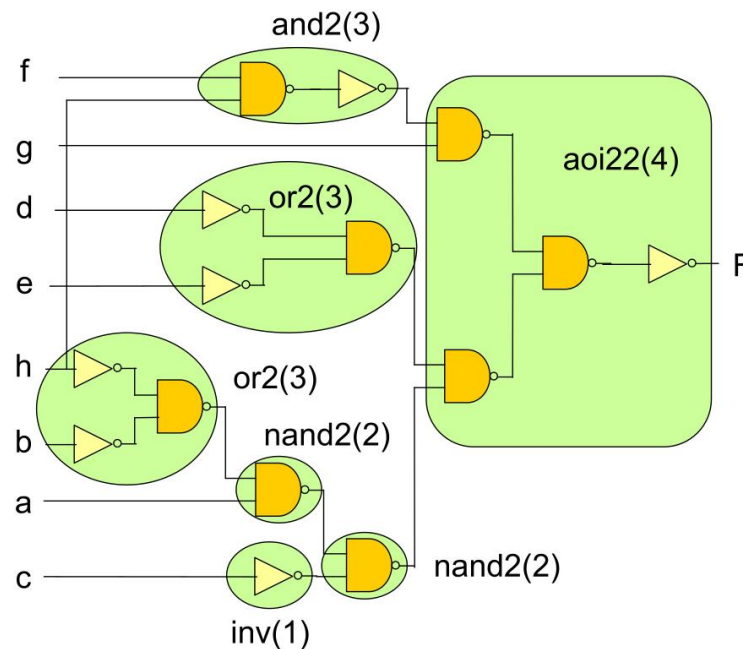
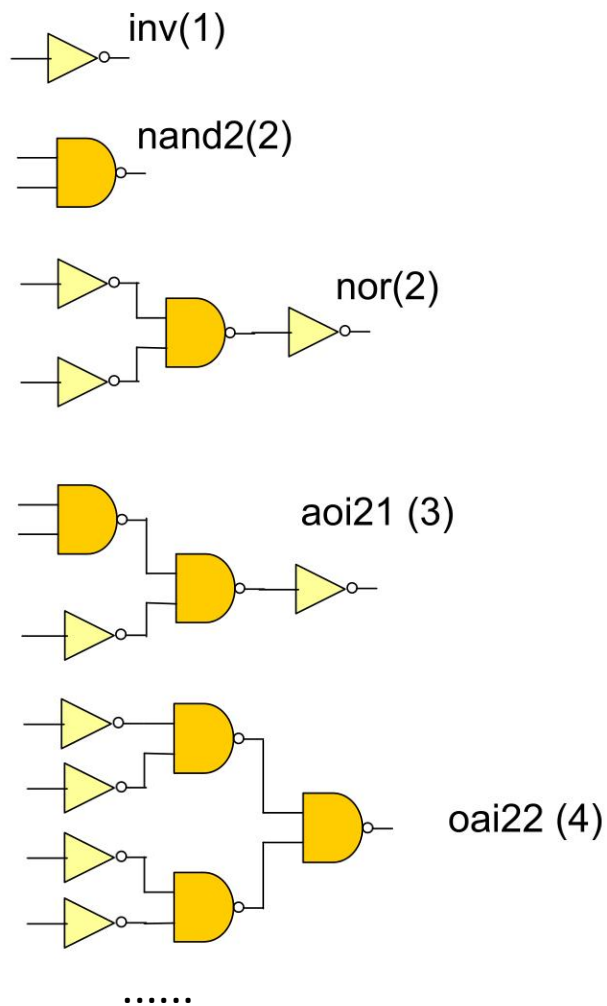


b) 重定时后，电路有 1 个寄存器和 1 个与门

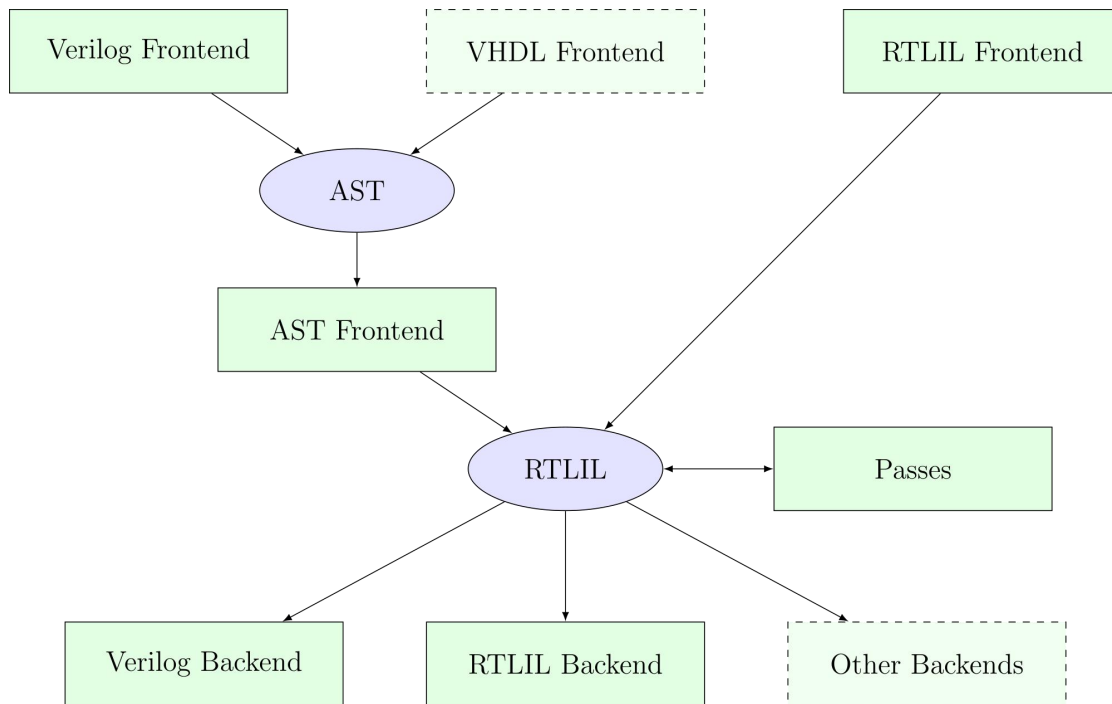
最小化面积



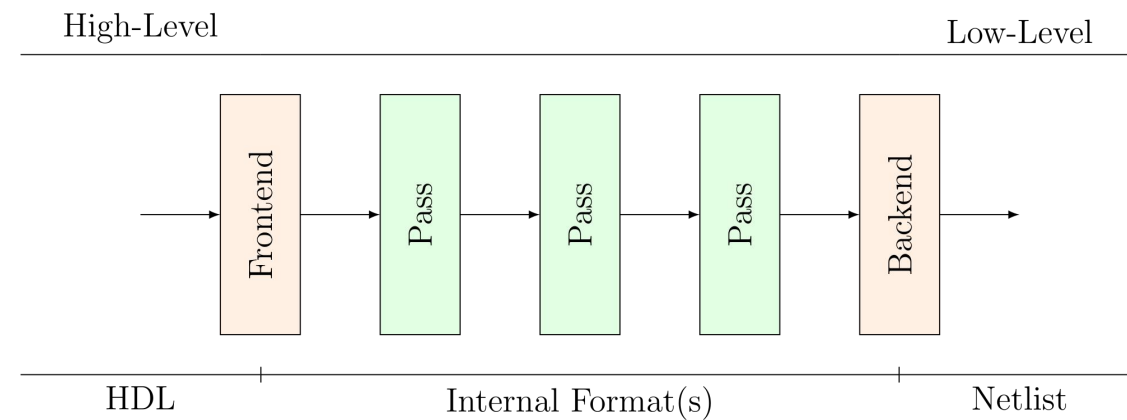
# 逻辑综合-Mapping



# 逻辑综合-Yosys综合流程



Yosys simplified data flow



General data- and control-flow of a synthesis tool

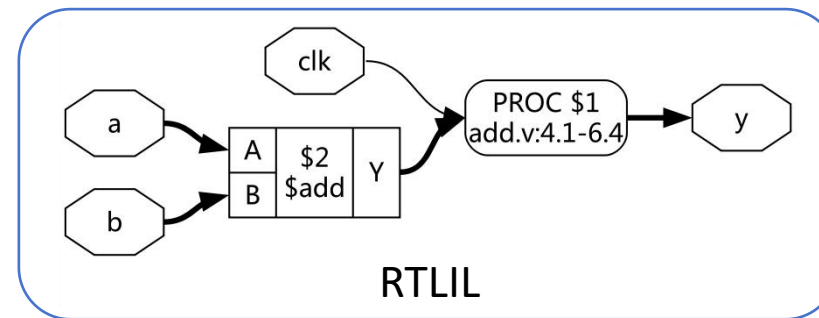
图片来源于: <https://yosyshq.readthedocs.io>

# 逻辑综合-Yosys综合流程示例

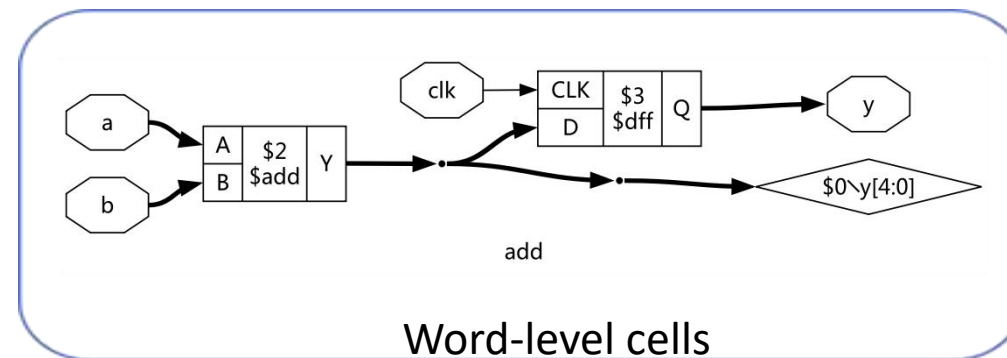
```
module add(  
    input wire clk,  
    input wire[3:0] a,  
    input wire[3:0] b,  
    output reg[4:0] y);  
always@(posedge clk) begin  
    y <= a+b;  
end
```

RTL Code

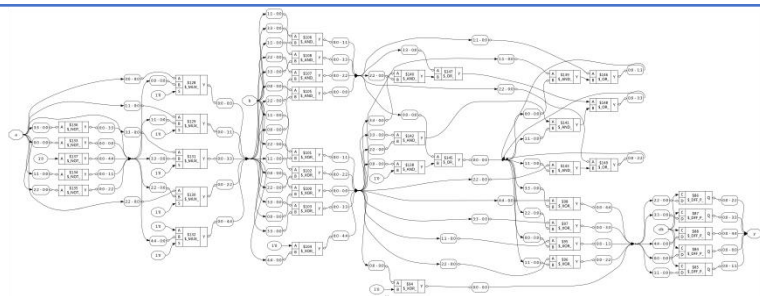
read\_verilog



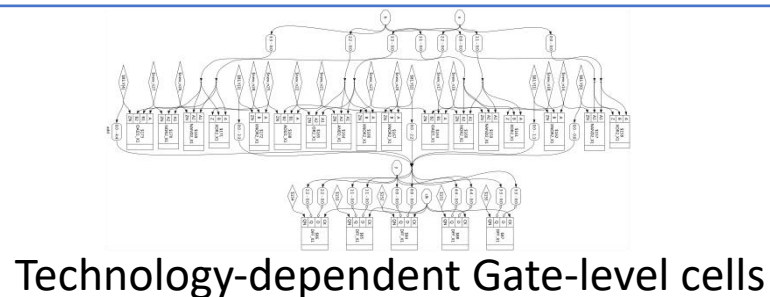
proc



techmap



dfflibmap  
abc -liberty



# 逻辑综合-Yosys调用ABC进行综合优化

```
module alu(  
    input  [7:0] A,          // 输入操作数A  
    input  [7:0] B,          // 输入操作数B  
    input  [2:0] opcode,     // 操作码  
    output reg [7:0] Y,      // 结果  
    output reg carry  
);  
  
always @(*) begin  
    case(opcode)  
        3'b000: {carry, Y} = A + B;    // 加法  
        3'b001: {carry, Y} = A - B;    // 减法  
        3'b010: Y = A & B;             // 按位与  
        3'b011: Y = A | B;             // 按位或  
        3'b100: Y = A ^ B;             // 按位异或  
        3'b101: Y = ~A;                // A取反  
        default: Y = 8'b0;  
    endcase  
end  
  
endmodule
```

## Baseline:

```
read_verilog alu.v  
synth  
abc -liberty merged.lib -script "+map;stime;"
```

## 结果:

**Area = 168.91 Delay = 283.67 ps**

Yosys 0.44 (git sha1 80ba43d26)

# 逻辑综合-Yosys调用ABC进行综合优化-面积/延迟

Baseline: **Area = 168.91 Delay = 283.67 ps**

面积优化:

```
read_verilog alu.v
synth -top alu
abc -liberty merged.lib -script "+strash; fraig; balance -l; rewrite -l; refactor -l;
balance -l; rewrite -l; rewrite -zl; balance -l; refactor -zl; rewrite -zl; balance
-l;map;stime;"
```

**Area = 148.43 Delay = 316.03 ps**

面积减小12.12%，延迟增加11.41%

延迟优化:

```
read_verilog alu.v
synth -top alu
abc -liberty merged.lib -script "+strash; balance; rewrite; refactor; balance;
rewrite; rewrite -z; balance; refactor -z; rewrite -z; balance; map; upsize; stime"
```

**Area = 228.23 Delay = 208.86 ps**

面积增加35.12%，延迟减小26.37%

# 逻辑综合-Yosys调用ABC进行Retiming

```
module mac(  
    input clk,  
    input [7:0]A,  
    input [7:0]B,  
    input [15:0]C,  
    output reg[16:0]Y  
);  
  
reg [15:0] multi_result;  
  
always@(posedge clk) begin  
    multi_result <= A*B;  
    Y = multi_result + C;  
end  
  
endmodule
```

## Baseline:

```
read_verilog mac.v  
synth -top mac  
abc -clk clk -liberty merged.lib -script "+strash;map;stime;"
```

**Area = 650.37 Delay = 612.45 ps**

## 使用retiming:

```
read_verilog mac.v  
synth -top mac  
abc -clk clk -liberty merged.lib -script "+strash; retime -M 4;  
map;stime;"
```

**Area = 651.17 Delay = 337.03 ps**

面积增加0.12%，延迟减小44.97%