北京大学
PEKING UNIVERSITY

人工智能的硬件基石

从物理器件到计算架构

第三讲：数字逻辑与计算单元设计

主讲：陶耀宇

2026年春季

注意事项

北京大学 PEKING UNIVERSITY

- 课程作业情况

- **作业将在3月底-4月中旬、4月中旬-5月初、5月中旬-6月初**

  2-3周完成时间

- **第1次lab时间：4月10日-5月10日**
- **第2次lab时间：5月10日-6月15日**
- **本周开始，助教安排硬件Verilog/SystemVerilog编写及设计、验证全流程入门介绍，有兴趣的同学可参与！**

注意事项

北京大学 PEKING UNIVERSITY

- **课程作业情况**

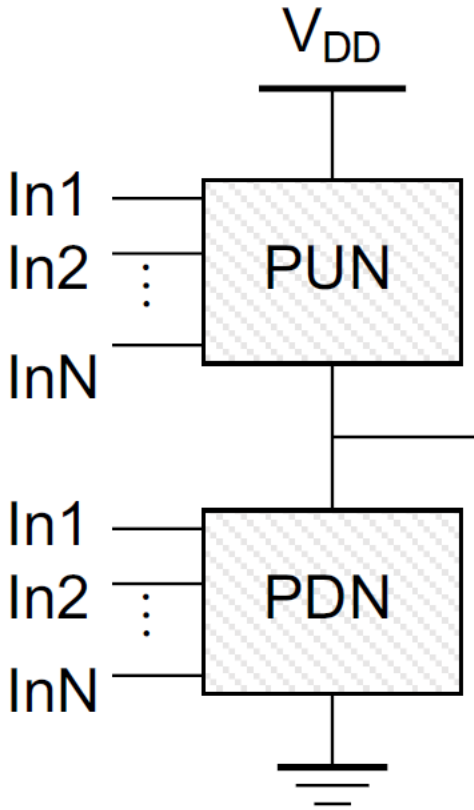
- CLAB平台：请各位选课同学在第2-3周，**使用学号登陆CLab平台**，并同意用户协议以激活账号
- 激活账号是后续lab能够进行的必要条件，请务必完成！
- Clab网址：clab.pku.edu.cn
- **Clab问题请联系助教李中源、罗子翔同学**

目 录

CONTENTS

经典静态逻辑电路

- NMOS、PMOS可以组成PDN和PUN



PMOS only

上拉逻辑设计：

Pull-up network: make a connection from V_{dd} to F when F(In1,In2…)=1

F(In1,In2,…InN)

NMOS only

Pull-down network: make a connection from Ground to F when F(In1,In2…)=0

下拉逻辑设计：

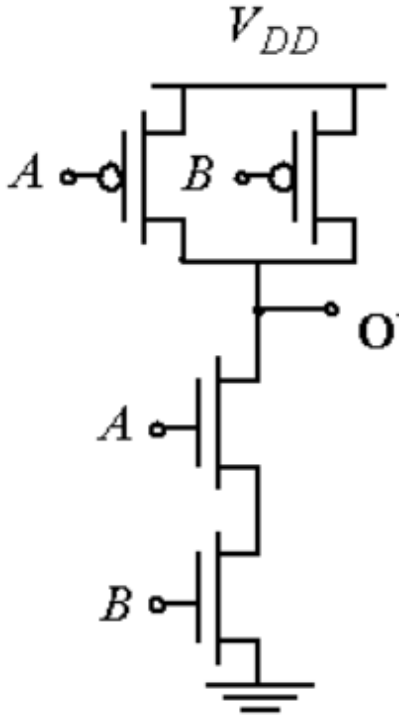
PUN and PDN are *dual* networks

经典静态逻辑电路

- 由PDN和PUN组成的NAND门电路

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table of a 2 input NAND gate



$OUT = \overline{A \bullet B}$ NAND门

PDN: Connects OUT to ground when A●B=1

PUN: Connects OUT to V_{dd} when $\overline{A} + \overline{B} = 1$

So OUT = Complement of PDN function

Also OUT = PUN function with each input inverted

当A和B同为1时，输出为0

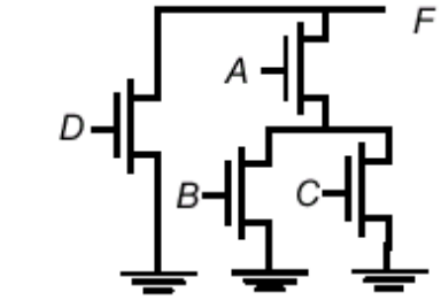
当A和B有0时，输出为1

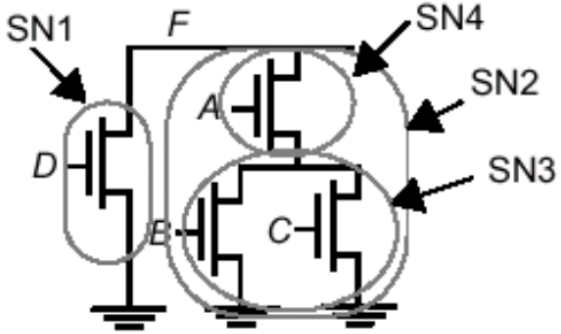
经典静态逻辑电路

- 由PDN和PUN组成的复杂静态逻辑电路

$$F = \overline{D + (A\,(B + C))}$$

什么情况下F为0 -> D+A(B+C) = 1





忽略取非操作，直接构建PDN

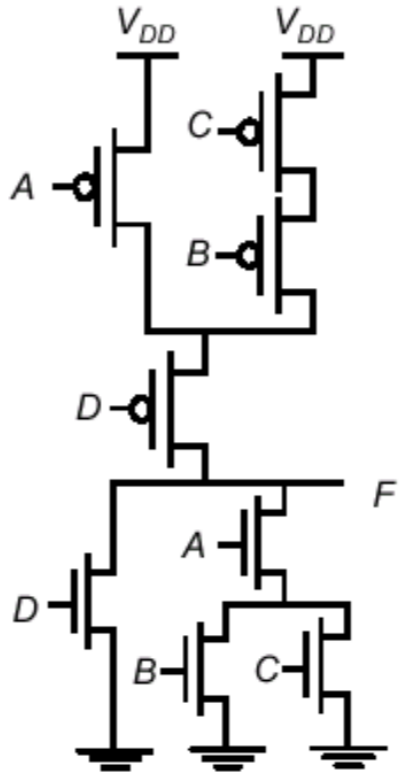
Ex：D+X意味着D与X并联

构建PDN的亚网络

在SN3内，B和C是并联的，

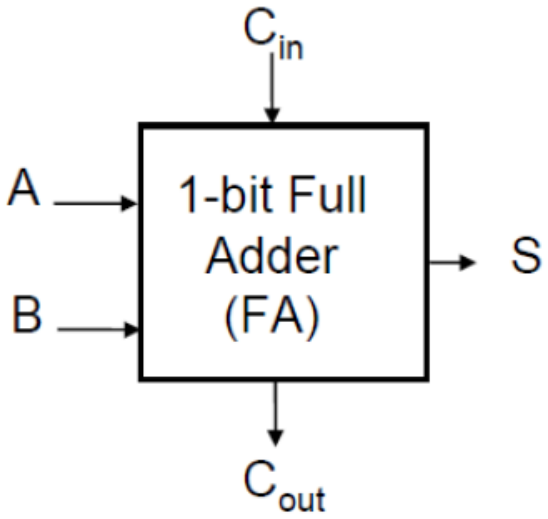
则在PUN中，他们串联

自下向上构建



经典静态逻辑电路

- 简单1bit加法器电路
  - A[n-1:0] + B[n-1:0] = S[n-1:0]



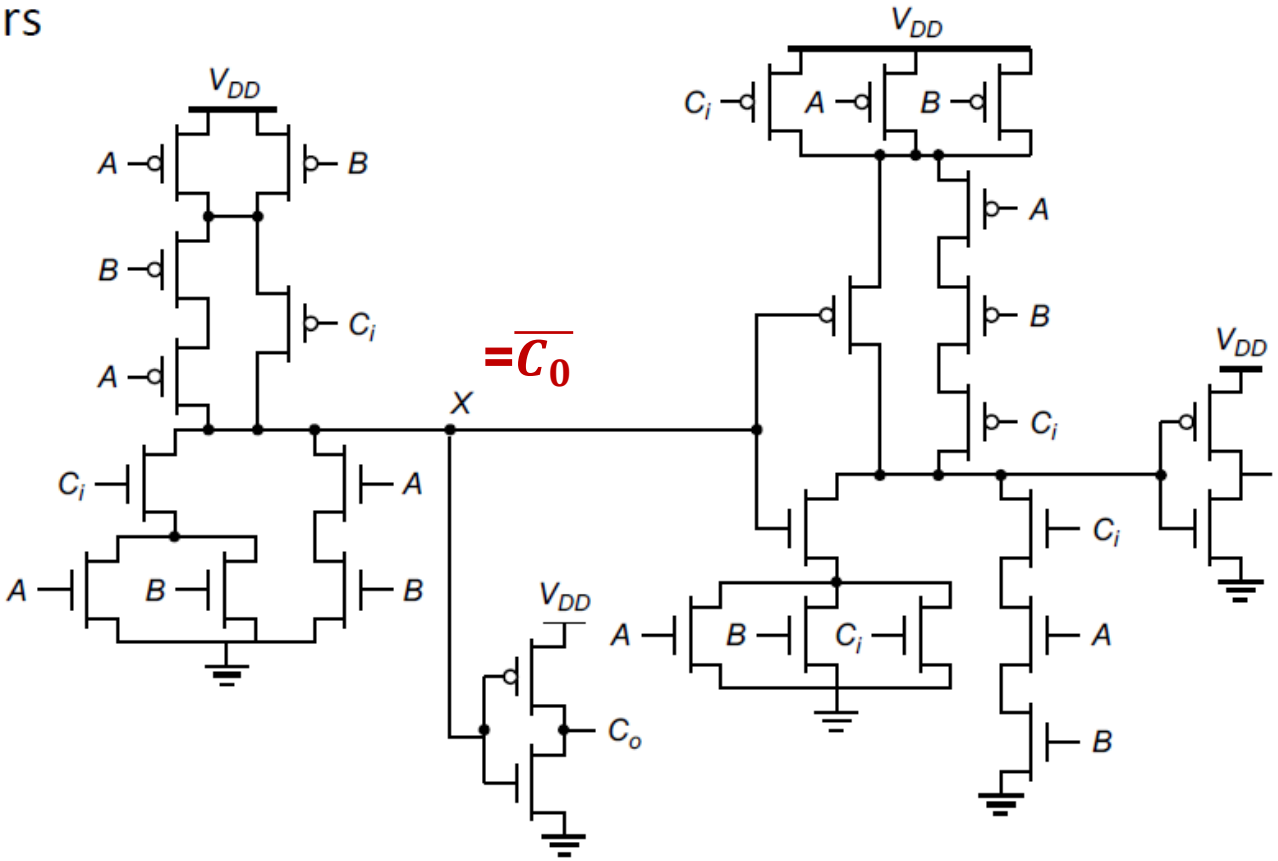
A	B	C_{in}	C_{out}	S	carry status
0	0	0	0	0	kill
0	0	1	0	1	kill
0	1	0	0	1	propagate
0	1	1	1	0	propagate
1	0	0	0	1	propagate
1	0	1	1	0	propagate
1	1	0	1	0	generate
1	1	1	1	1	generate

单比特加法器逻辑设计

- $C_o = AB + BC_i + AC_i = AB + (A + B)C_i$
- $S = A \oplus B \oplus C_i = ABC_i + \overline{C_o}(A + B + C_i)$
- $G = AB,\ K = \overline{A}\,\overline{B},\ P = A \oplus B$

经典静态逻辑电路

- 简单1bit加法器电路
  - $C_o = AB + BC_i + AC_i = AB + (A + B)C_i$
  - $S = A \oplus B \oplus C_i = ABC_i + \overline{C_o}(A + B + C_i)$
  - 28 transistors

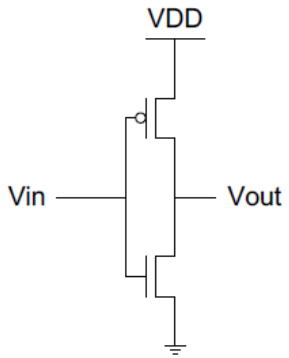


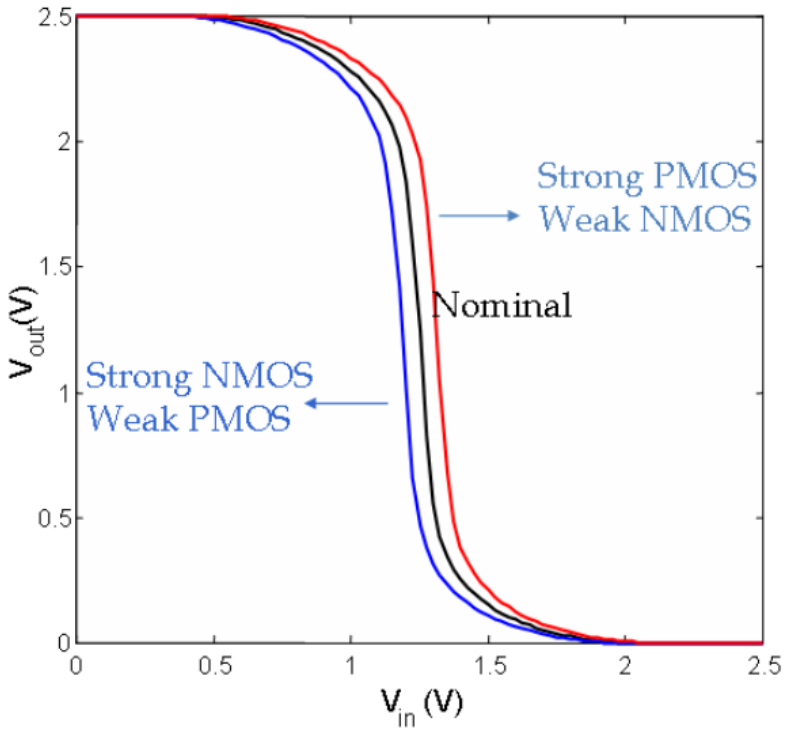
目 录

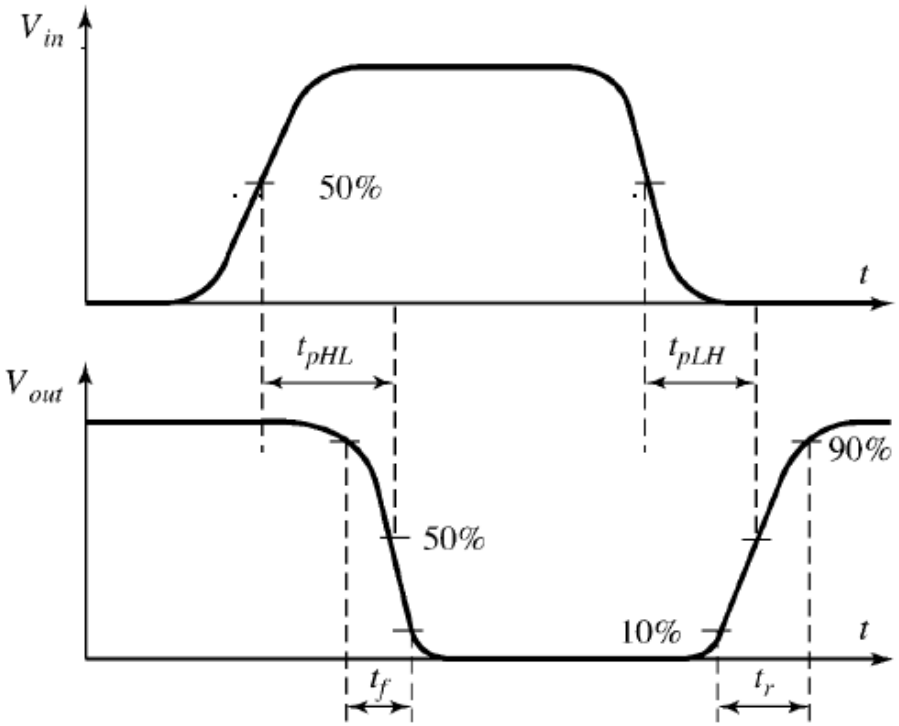
CONTENTS

什么是电路的延迟

- 以Inverter反相器为例



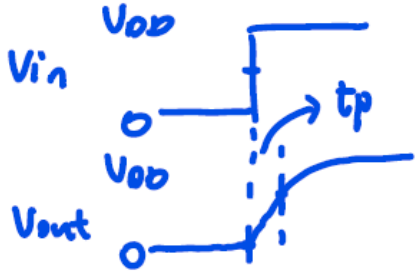


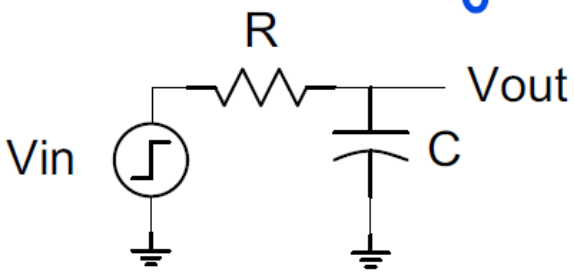


什么是电路的延迟

- 一阶RC延迟分析

A First-Order RC Network





$$V_{out}(t) = (1 - e^{-t/\tau})V_{DD} = \frac{1}{2}V_{DD}$$

$$e^{-t/\tau} = \frac{1}{2}$$

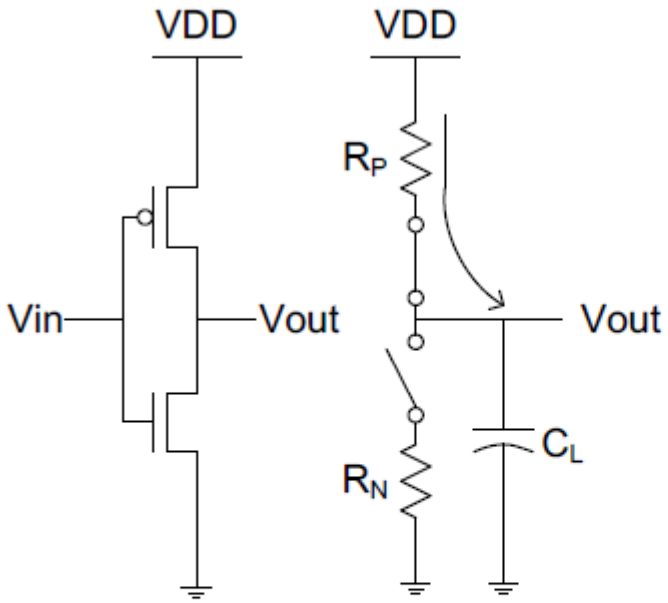
$$-t/\tau = -\ln 2$$

$$\tau = R \times C$$

$$t_p = \ln(2)\tau = 0.69R \times C \qquad t = \ln(2)\tau$$

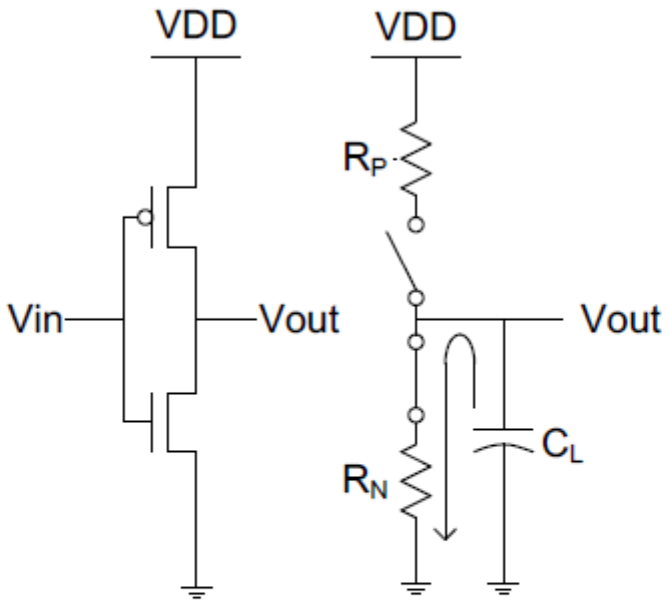
反相器延迟

- 利用一阶RC延迟分析方法



$V_{in} = 0$

(a) Low-to-high



$V_{in} = V_{DD}$

(b) High-to-low

$$t_{pHL} = f(R_N \times C_L)$$

$$t_{pHL} = 0.69 R_N \times C_L$$

$$t_{pLH} = 0.69 R_P \times C_L$$

降低延迟的设计方法

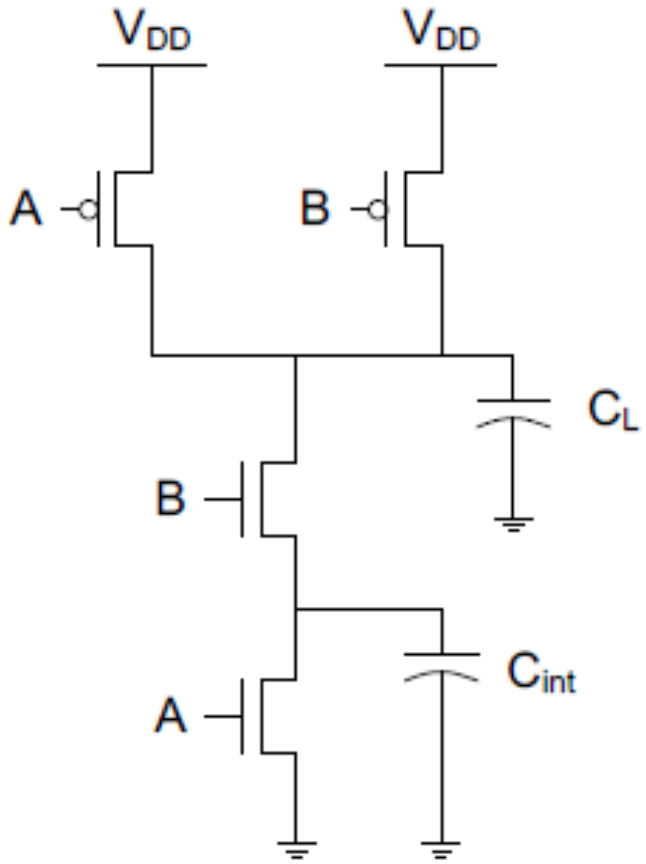
- 利用一阶RC延迟分析方法

$$t_{pHL} = f(R_N \times C_L)$$
$$t_{pHL} = 0.69 R_N \times C_L$$
$$t_{pLH} = 0.69 R_P \times C_L$$

- **利用较小的电容 – 降低C**
  - 版图紧凑, 布局合理
  - 保持较短走线&减少diffusion routing
- **增加晶体管尺寸 – 降低 R**
  - 避免self-loading出现，否则会导致寄生电容增大
- **增加电源电压**
  - 同时会影响可靠性与功耗，因而一般不采用

输入Pattern对延迟的影响

- 利用一阶RC延迟分析方法

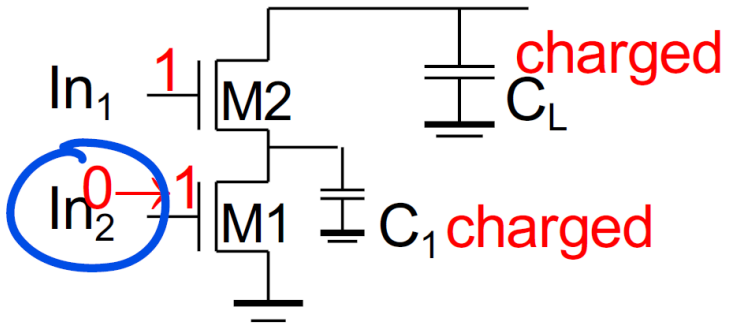


电路延迟与输入的顺序有关！

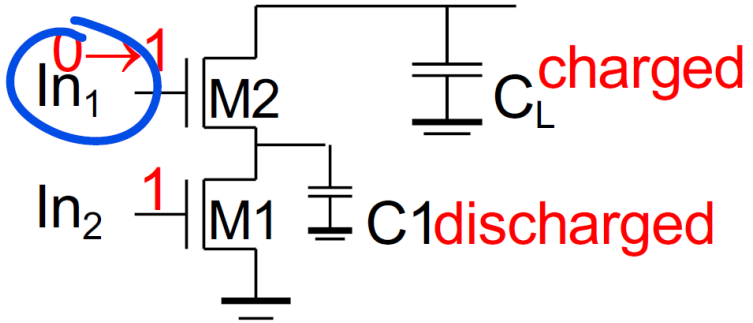
- *Ignore C_{int} for the moment!*
- Low to high transition
  - both inputs go low
    - delay is 0.69 $R_p/2$ C_L
  - one input goes low
    - delay is 0.69 R_p C_L
- High to low transition
  - both inputs go high
    - delay is 0.69 $2R_n$ C_L

利用Transistor Ordering提升逻辑速度

- 复杂的Transistor Ordering需要仿真工具支持



延迟由C_L与C_1放电时间决定



延迟由C_L放电时间决定

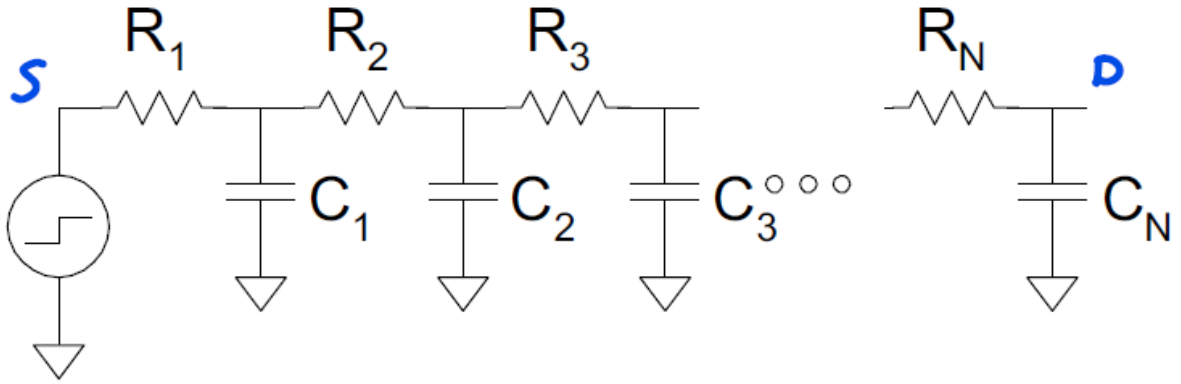
逻辑电路的Elmore Delay模型

- 拓展多级的RC模型

  - 导通晶体管看作电阻
  - 电路网络建模为RC阶梯
  - RC阶梯的Elmore延迟
  - Apply to complex gates（i.e.,stacks),also interconnect（later）

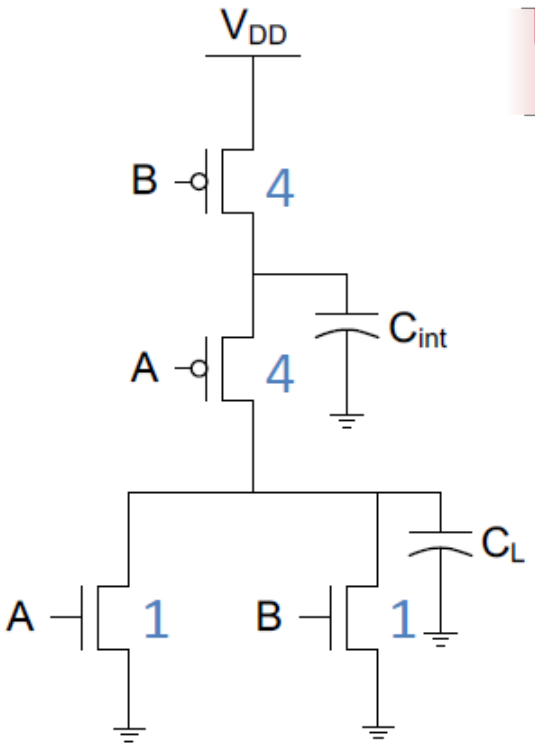
$$t_{pd} \approx \sum_{\text{nodes } i} R_{i-to-source} C_i$$

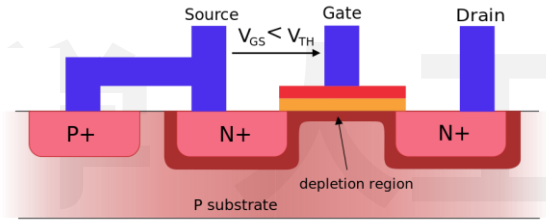
$$= R_1 C_1 + (R_1 + R_2) C_2 + \ldots + (R_1 + R_2 + \ldots + R_N) C_N$$

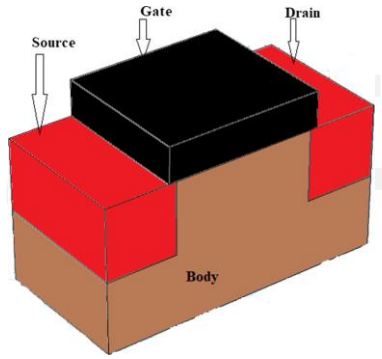


逻辑电路的Transistor Sizing

- 目标是将PDN和PUN的worst-case延迟进行匹配



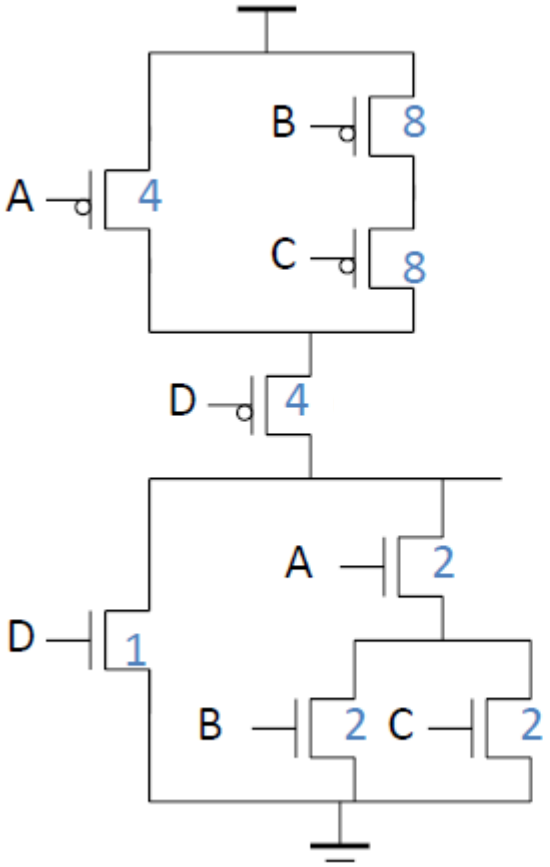




沟道长度由制程决定

沟道宽度由设计决定

R(NMOS/PMOS) ~ 1/(WL)



假设相同的W*L

PMOS的电阻是NMOS电阻的2倍

Rpe = 2*Rne

（由半导体特性决定，此处不做深入）

Rp = Rpe/(Wp*Lp)

Rn = Rne/(Wn*Ln)

$$\overline{OUT = D + A \bullet (B + C)}$$

逻辑电路功耗

- 静态功耗与动态功耗

静态功耗

Static Power

定义 电路处于稳态时消耗的功率

来源 亚阈值漏电流、栅极漏电流、结反向偏置漏电流

公式 $P_{static} = V_{DD} \times I_{leak}$

趋势 工艺缩小后显著增大，成为主要功耗来源之一

动态功耗

Dynamic Power

定义 电路状态翻转时消耗的功率

来源 负载电容充放电、短路电流

公式 $P_{dyn} = \alpha \times C_L \times V_{DD}^2 \times f$

优化 降低电压、减小电容、降频、减少翻转率

总功耗 $P_{total} = P_{static} + P_{dynamic}$ · 先进工艺节点中静态功耗占比持续上升

逻辑电路功耗

- 静态功耗与动态功耗

充电过程 $0 \rightarrow V_{DD}$

电源提供的总能量：

$$E_{supply} = C_L \times V_{DD}^2$$

电容实际储存的能量：

$$E_{stored} = \frac{1}{2} C_L \times V_{DD}^2$$

差值 $\frac{1}{2}C_L V_{DD}^2$ 在 PMOS 管上以热耗散

放电过程 $V_{DD} \rightarrow 0$

电容释放储存的能量：

$$E_{dissipated} = \frac{1}{2} C_L \times V_{DD}^2$$

全部通过 NMOS 管以热耗散

一次完整翻转 (0→1→0)：

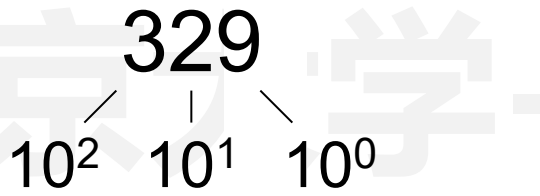
$$E_{total} = C_L \times V_{DD}^2$$

目 录

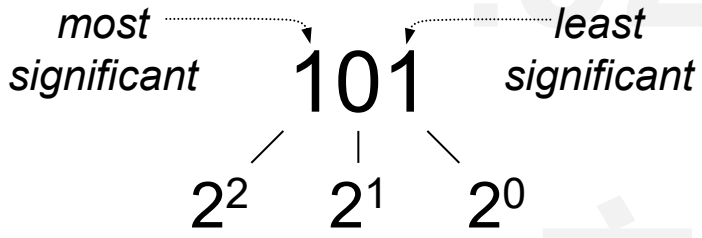
CONTENTS

量化与数据格式

- 最基础的二进制数据格式 – 原码（无符号数）



3x100 + 2x10 + 9x1 = 329



1x4 + 0x2 + 1x1 = 5

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

An *n*-bit unsigned integer

represents 2^n values:

from 0 to 2^n-1

量化与数据格式

- 最基础的二进制数据格式 – 原码（无符号数）

```
  10010        10010          1111
+  1001      +  1011        +    1
-------      -------        ------
  11011        11101          10000

                10111
              +   111
              -------
                11110
```

carry

量化与数据格式

- **有符号数的表现格式**

一个n bit数可以表示2^n不同的值

- 近一半赋值到正整数(1 ~ (2^{n-1}-1))

  近一半赋值到负整数((- (2^{n-1}-1)) ~ (-1))

- 还剩下两个值：表示0

正整数

同无符号数– 最高位为0

00101 = 5

负整数

对于原码来说，将最高位设为1代表负数，其他比特同无符号数一样

10101 = -5

北京大学 PEKING UNIVERSITY

- **有符号数的表现格式 – 补码**

原码(sign-magnitude)有什么问题？

0有两种重复表示 (+0 and –0)

计算电路复杂

对负数做加法时，实际上需要减法操作

需要考虑减法中的借位操作

```
  00101  (5)        01001  (9)
+ 11011  (-5)     + 10111  (-9)
  00000  (0)        00000  (0)
```


2的补码表示方法可以让计算电路更简单

- 对于每个正数 X，保证其相反数(-X)满足 X + (-X) = 0,其中的加法为忽略最高位进位的普通加法

量化与数据格式

- **有符号数的表现格式 – 补码**

若数字为正数或是**0**

- 正常二进制表示方法

若数字为负数

- 写出和它互为相反数的那个正数
- 翻转每一个比特
- 最后加**1**

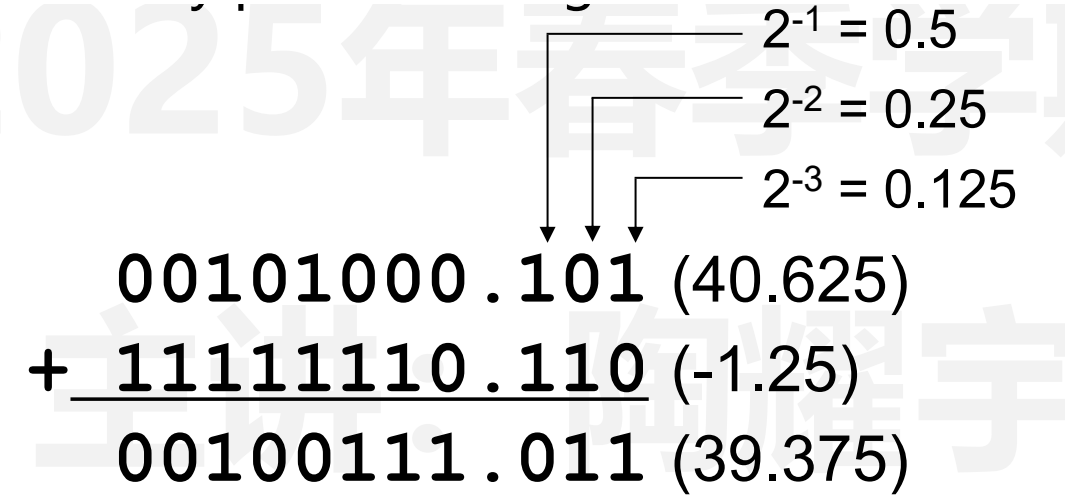
```
  00101  (5)          01001  (9)
  11010  (1's comp)   10110  (1's comp)
+     1             +     1
  11011  (-5)         10111  (-9)
```

北京大学 PEKING UNIVERSITY

- 定点数 – Fixed-point

如何表示分数？

- 使用二进制小数点来分开2的正数次幂和负数次幂（同十进制相似）
- 2的补码加法和减法依然成立
  - 前提时小数点对齐



No new operations -- same as integer arithmetic.

量化与数据格式

- 特别大和特别小的数：浮点数 – Floating-point

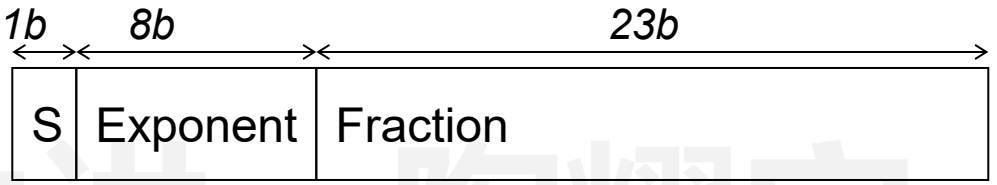
Large values: 6.023×10^{23} -- requires 79 bits

Small values: 6.626×10^{-34} -- requires >110 bits

使用科学计数法的等效： $F \times 2^{E}$

需要表示分数F (fraction), 指数E (exponent), and 符号位S(sign).

IEEE 754 浮点数标准(32-bits):



$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

量化与数据格式

- 特别大和特别小的数：浮点数 – Floating-point

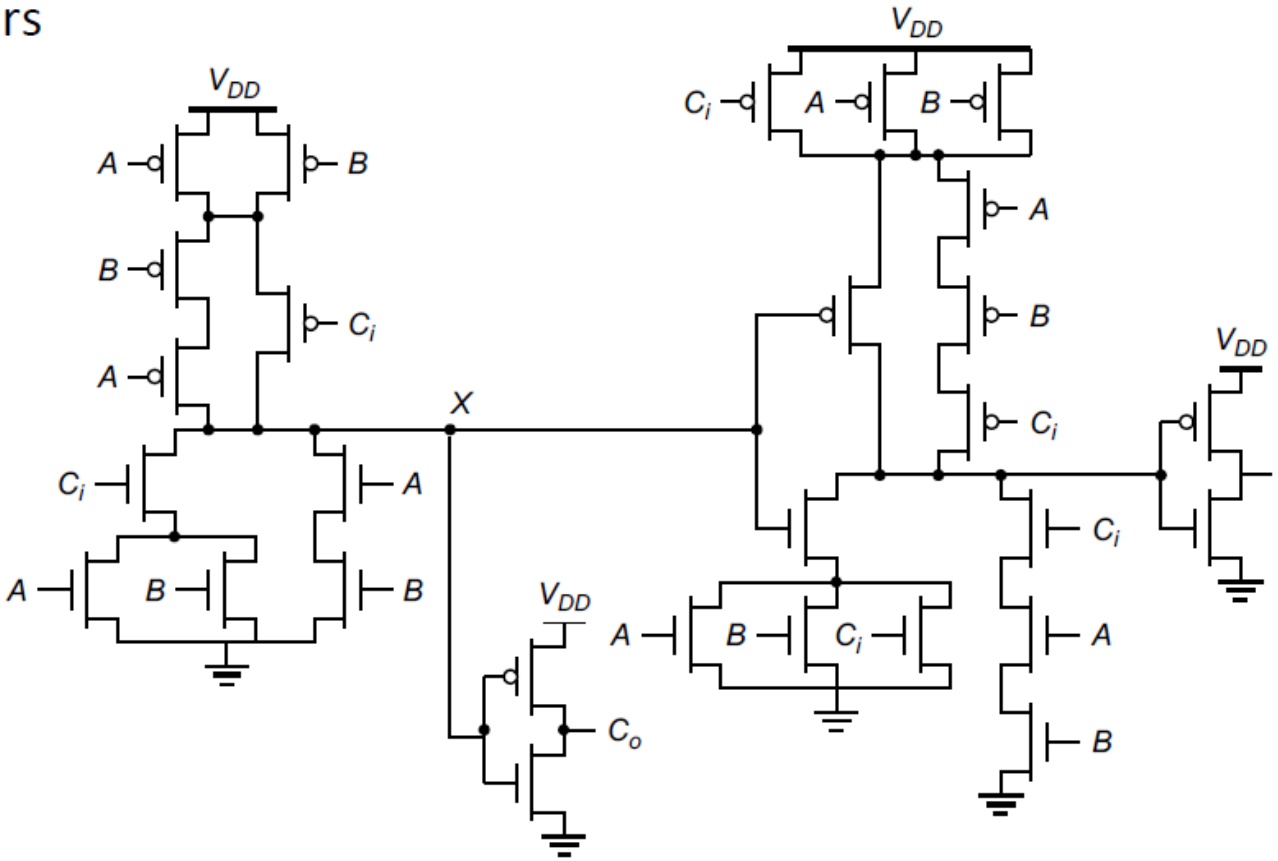
Single-precision IEEE floating point number:

- Sign is 1 – number is negative.
- Exponent field is 01111110 = 126 (decimal).
- Fraction is 0.100000000000… = 0.5 (decimal).

Value = $-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75$.

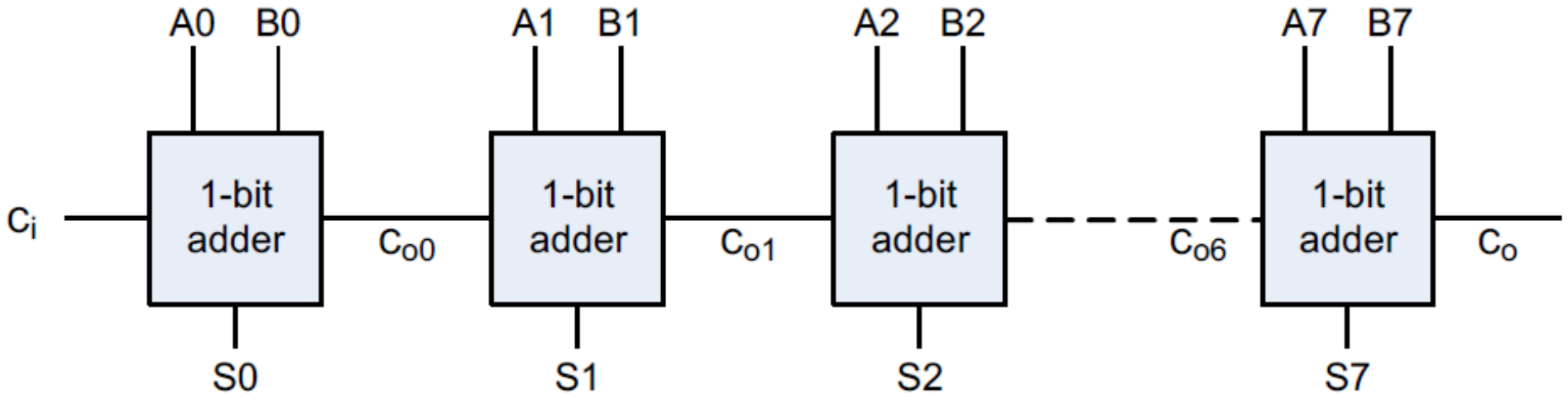
加法器设计

- 简单1bit加法器电路
  - $C_o = AB + BC_i + AC_i = AB + (A + B)C_i$
  - $S = A \oplus B \oplus C_i = ABC_i + \overline{C_o}(A + B + C_i)$
  - 28 transistors



加法器设计

- Ripple Carry加法器电路



最差延迟与比特数呈线性关系

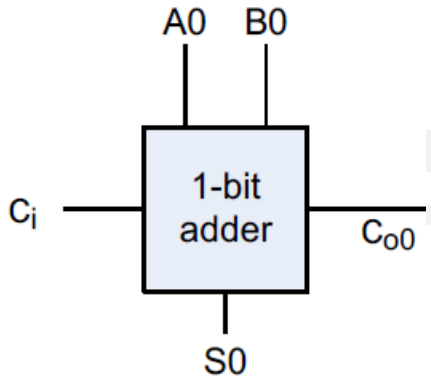
$$t_d = O(N)$$

$$t_{adder} = (N\text{-}1)t_{carry} + t_{sum}$$

目标：设计拥有最快可能进位路径的电路

加法器设计

- 基于PGK的加法器设计方法



Generate (G) = AB

Propagate (P) = A ⊕ B

- Generate: C_{out} = 1 independent of C_{in}
  - G = A • B
- Propagate: C_{out} = C_{in}
  - P = A ⊕ B
- Kill: C_{out} = 0 independent of C_{in}
  - K = ~A • ~B

$$C_o(G,P) = G + PC_i$$

$$S(G,P) = P \oplus C_i$$

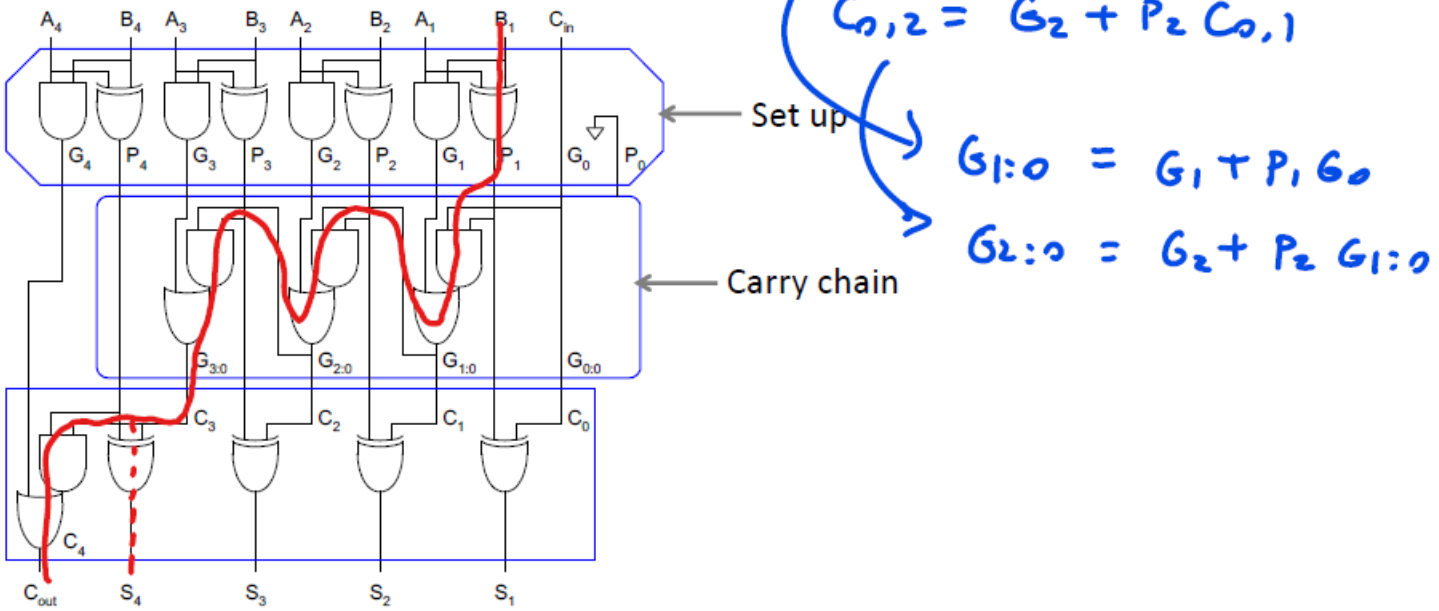
$P = A \oplus B$

$P = A + B$

- 基于PGK的加法器设计方法

Carry-Ripple using P and G

$$C_{i:0} = G_i + P_i \cdot C_{i-1:0}$$
$$G_{0:0} = C_{in}$$
$$P_{0:0} = 0$$
$$C_{out,i} = G_{i:0}$$



$C_{0,1} = G_1 + P_1 C_{in}$

$C_{0,2} = G_2 + P_2 C_{0,1}$

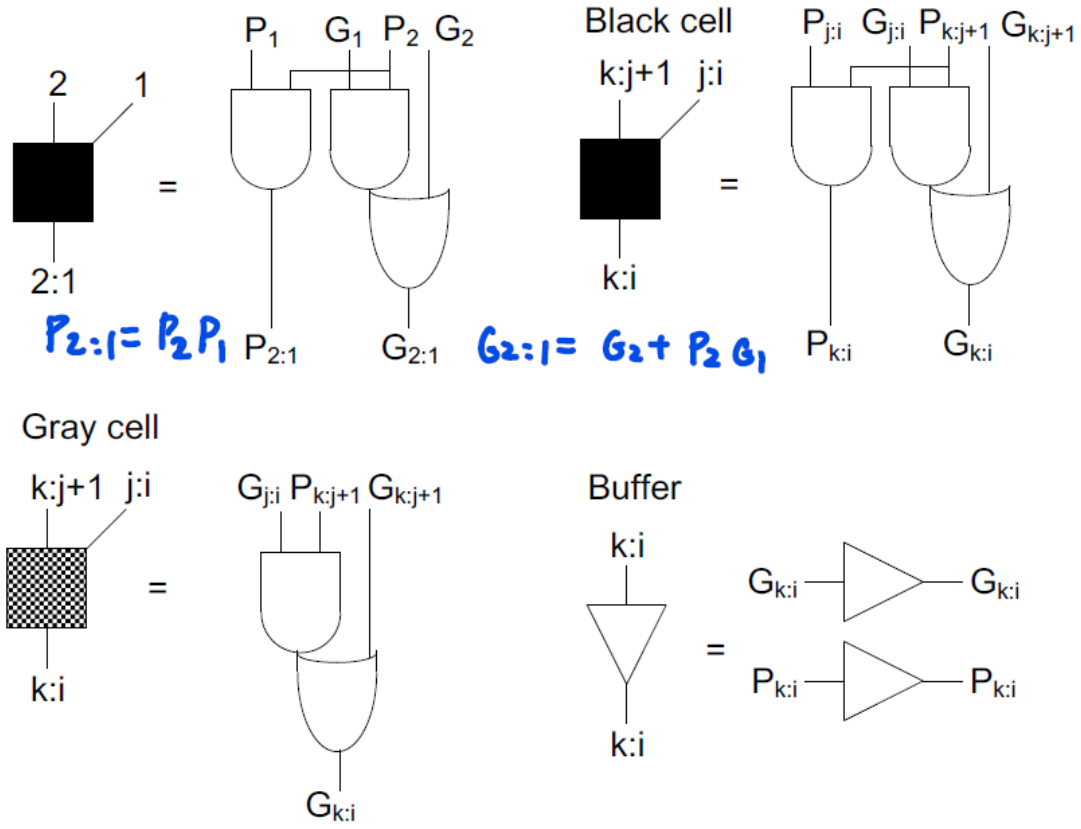
$G_{1:0} = G_1 + P_1 G_0$

$G_{2:0} = G_2 + P_2 G_{1:0}$

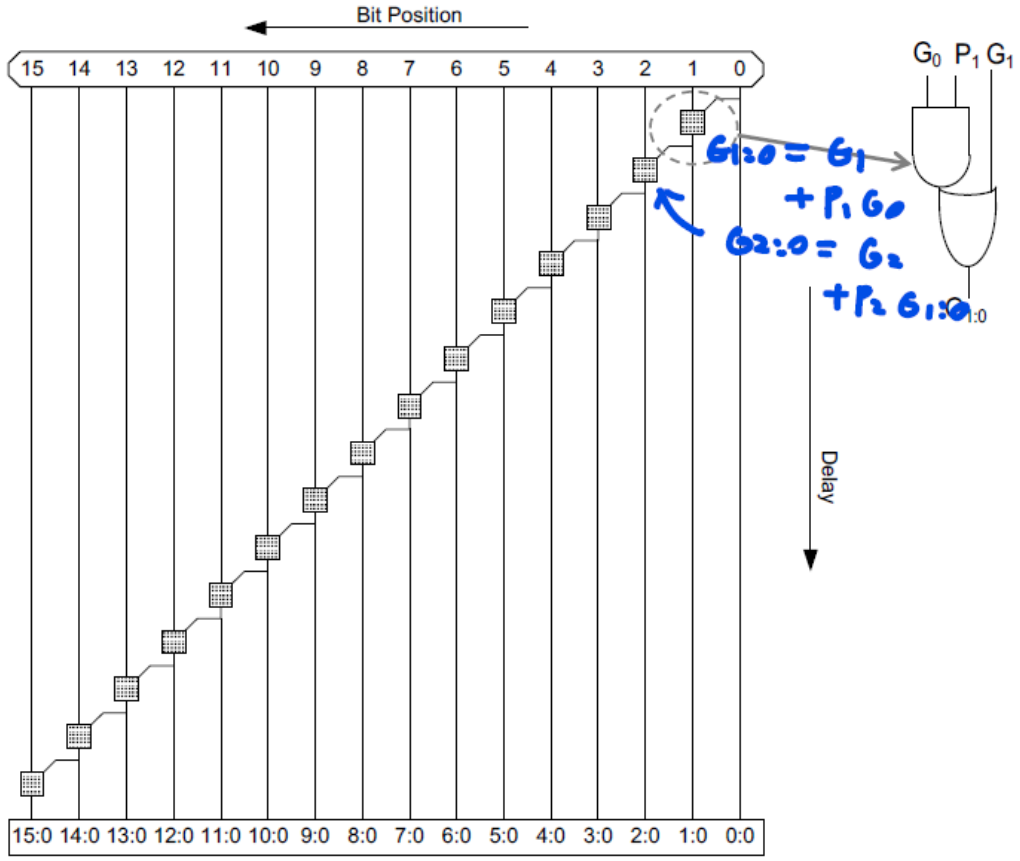
$t_{adder} = t_{setup} + (N-1)\, t_{carry} + \max(t_{carry}, t_{sum})$

加法器设计

- 基于PGK的加法器设计方法



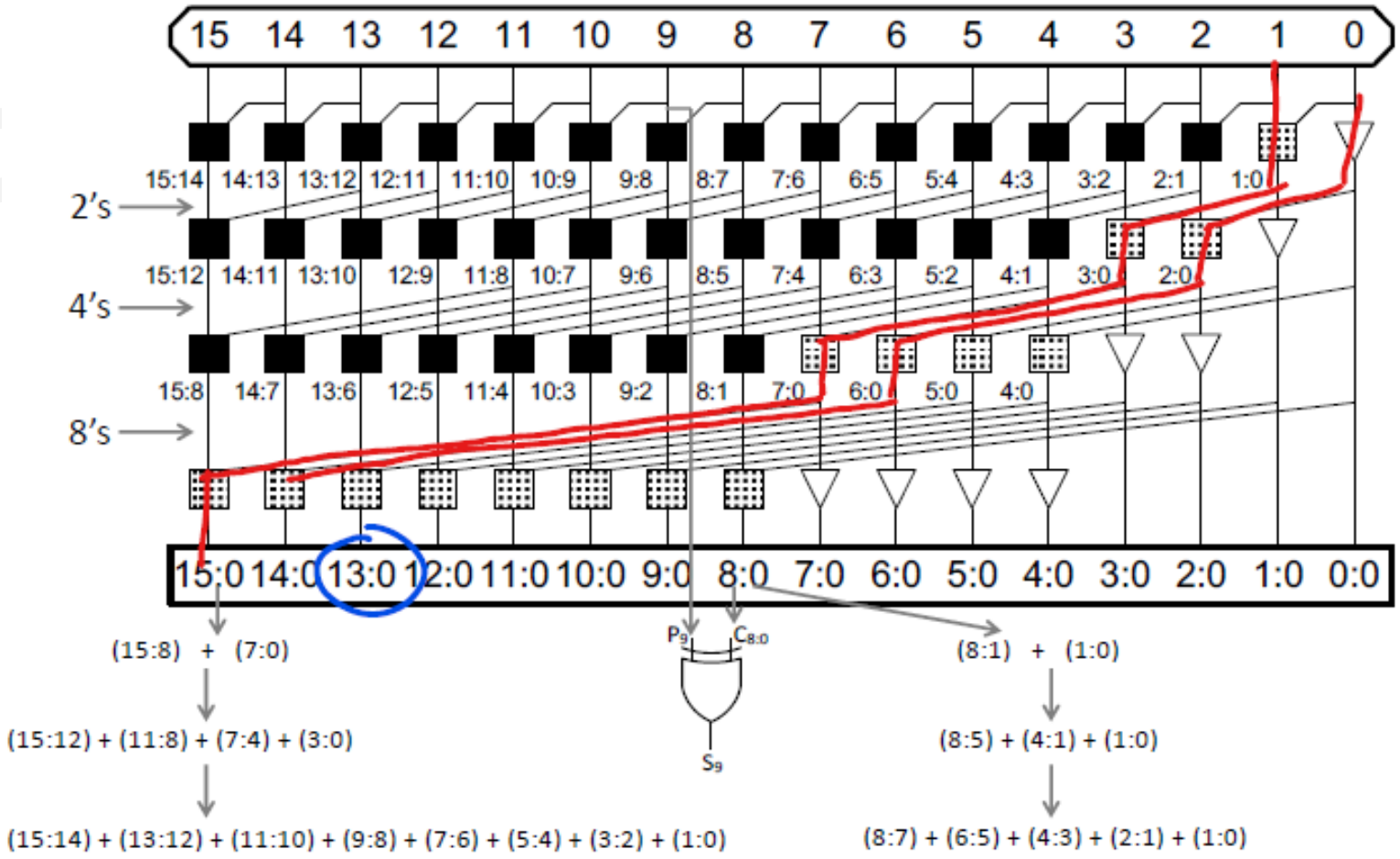
PG生成逻辑



Carry Ripple的PG图

加法器设计

- 基于PGK的加法器设计方法 – 复杂PG树加法器



$\log_2(N)$

$G_{15:0} = G_{13:6} + P_{13:6}\, G_{5:0}$

$G_{13:6} = G_{13:10} + P_{13:0}\, G_{9:6}$

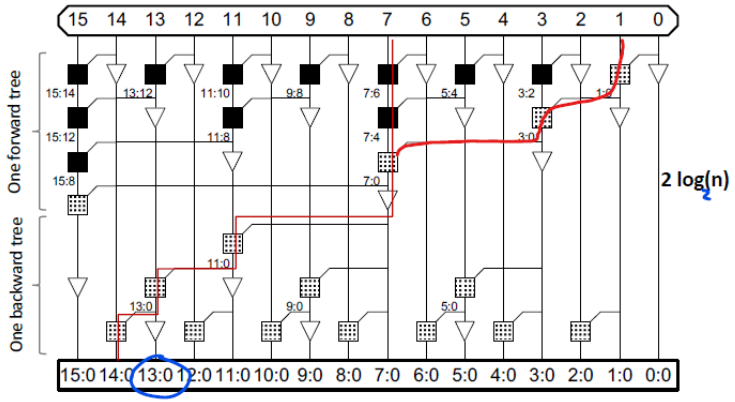
$P_{13:6} = P_{13:10}\, P_{9:6}$

$G_{5:0} = G_{5:2} + P_{5:2}\, G_{1:0}$

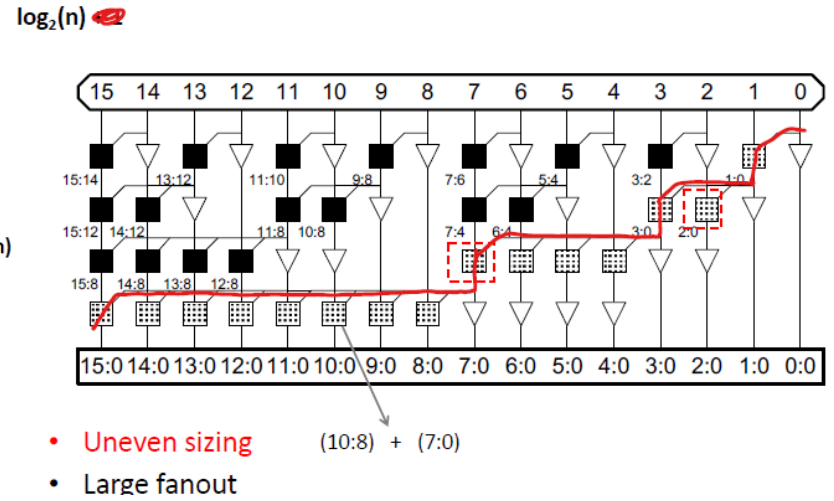
加法器设计

- 基于PGK的加法器设计方法 – 复杂PG树加法器

Brent-Kung

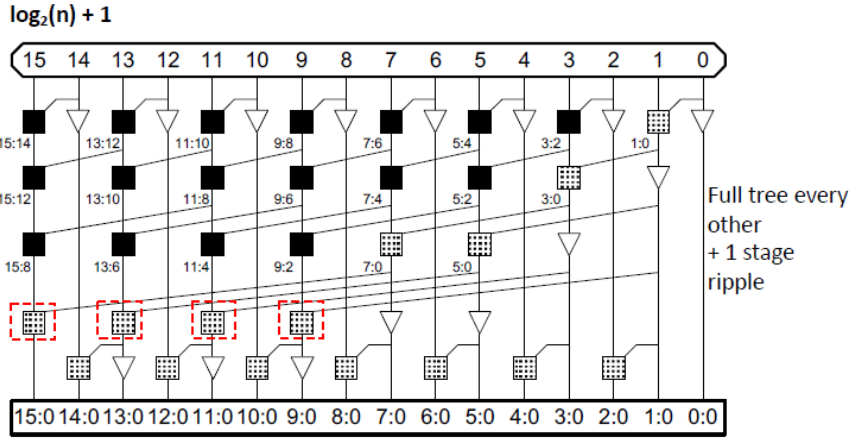


Sklansky



- Uneven sizing (10:8) + (7:0)
- Large fanout

Han-Carlson



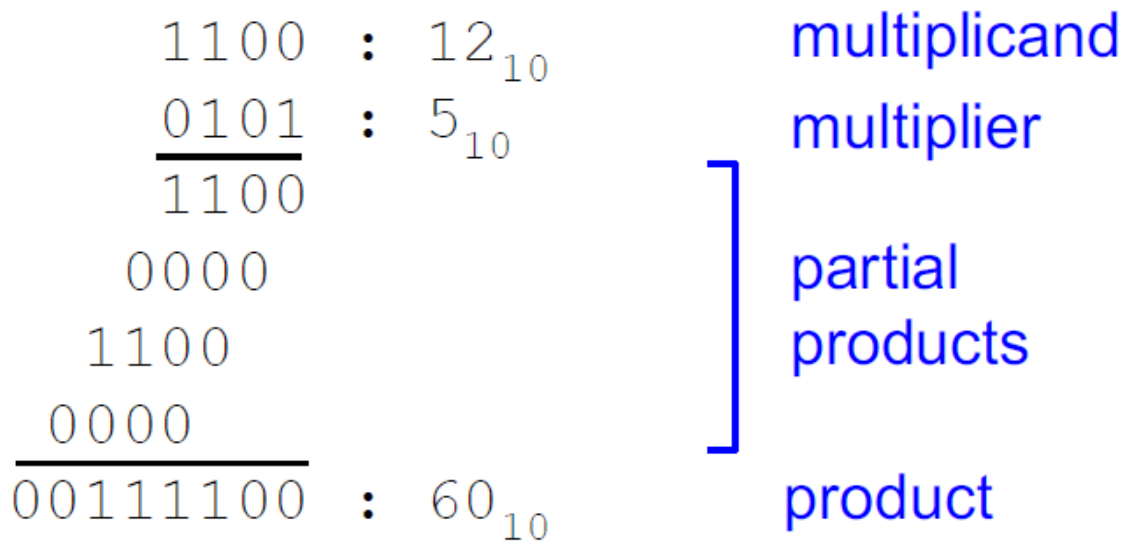
Low fanout, tradeoff between logic levels and wiring
Reduces wire length by half ! → half power compared to Kogge Stone

- Kogge-Stone: low logic levels, low fanout, high wiring
- Brent-Kung: low fanout, low wiring, high logic levels
- Sklansky: low logic levels, low wiring, high fanout

乘法器设计

- 乘法器设计的核心是部分和累加

Example:



M x N比特乘法

- 产生N个M比特部分乘积
- 求和得到M+N比特的结果

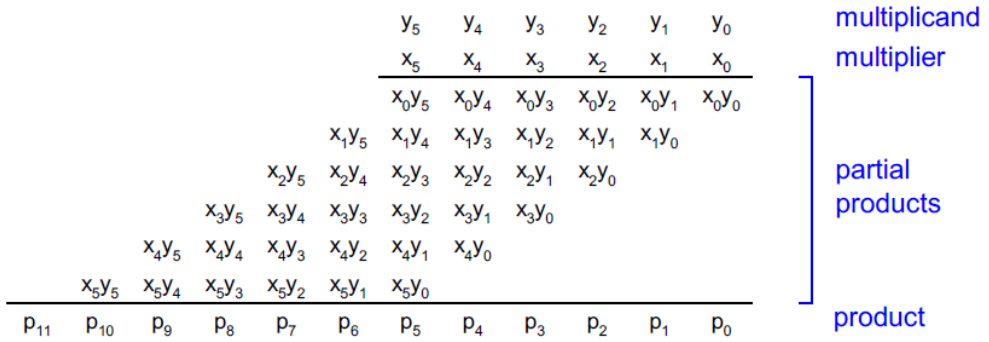
乘法器设计

- 乘法器设计的核心是部分和累加

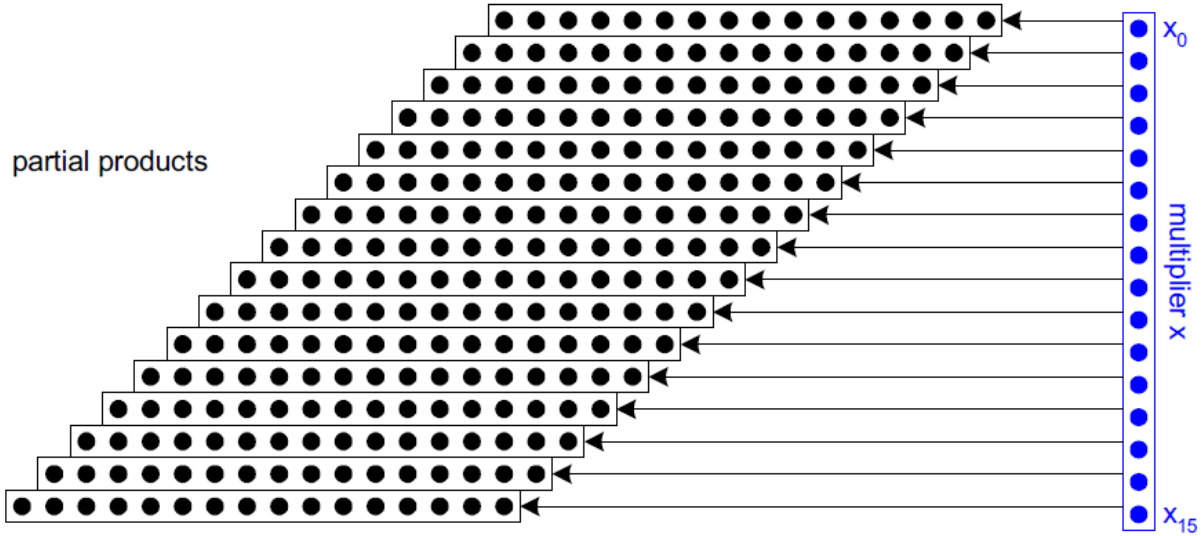
Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$

Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$

Product: $$P = \left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$

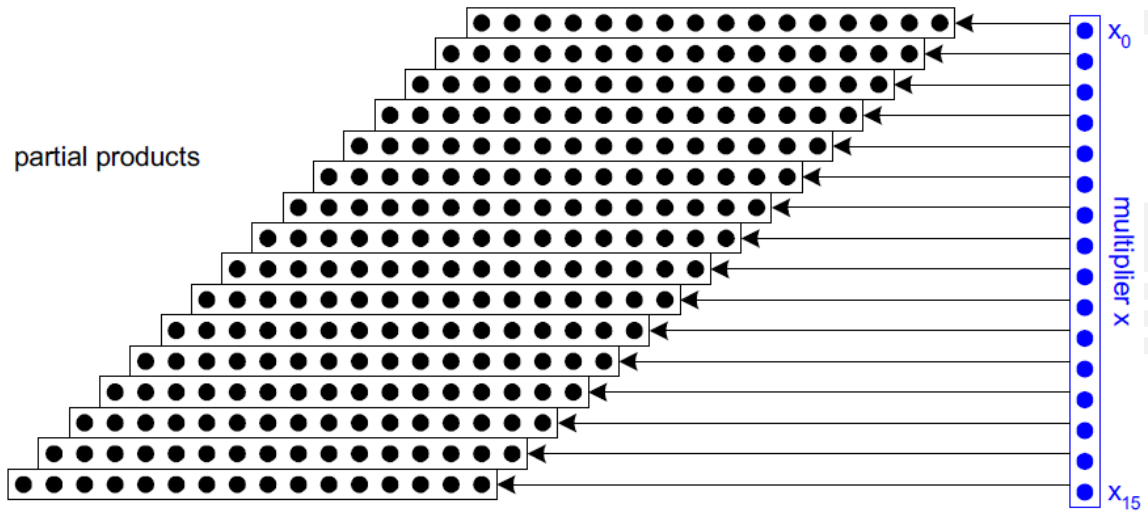


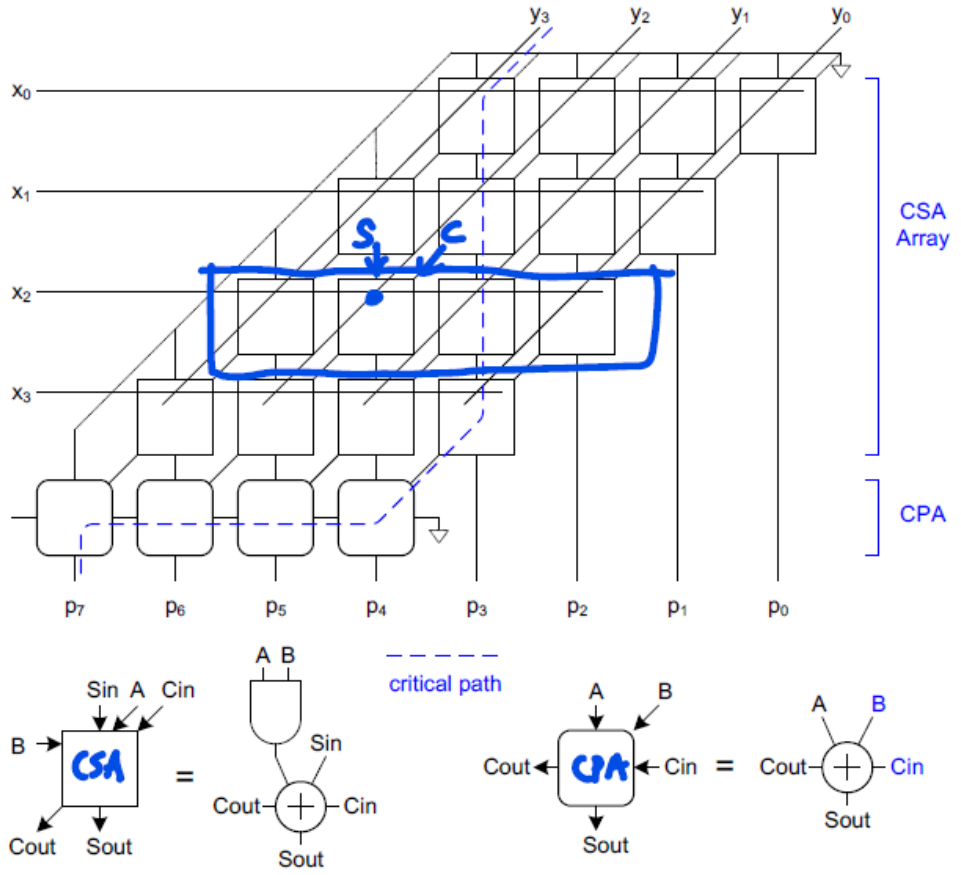
Each dot represents a bit



乘法器设计

- 乘法器设计的核心是部分和累加





乘法器设计

- 如何减少部分和累加的次数?

- 阵列乘法器需要N个部分结果

- 如果我们将乘数以r bits为单位分组做乘法，我们将获得N/r个部分结果
  - Faster and smaller?
  - Called radix-2^r encoding

X | PP
(0 0) | 0
(0 1) | Y
(1 0) | 2Y (4Y-2Y)
(1 1) | 3Y = (4Y-Y)

- Ex: r = 2: look at pairs of bits
  - Form partial products of 0, Y, 2Y, 3Y
  - First three are easy, but 3Y requires adder ☹

```
          1   1   0   0
        (0   1) (0   1)
        ---------------
          a   a   a   a
    b   b   b   b
    -------------------
```

乘法器设计

- 如何减少部分和累加的次数?

$$x \times y$$

$$y = -2^{n-1}y_{n-1} + 2^{n-2}y_{n-2} + \cdots 2^2 y_2 + 2y_1 + y_0 + y_{-1}, \quad y_{-1} = 0$$

$$\begin{aligned} y = {} & -2^{n-1}y_{n-1} + 2^{n-2}y_{n-2} + 2^{n-2}y_{n-2} - 2^{n-2}y_{n-2} + 2^{n-3}y_{n-3} + 2^{n-3}y_{n-3} \\ & - 2^{n-3}y_{n-3} + \cdots + 2y_1 + 2y_1 - 2y_1 + y_0 + y_0 - y_0 + y_{-1} \end{aligned}$$

$$\begin{aligned} = {} & -2^{n-1}y_{n-1} + 2 \times 2^{n-2}y_{n-2} - 2^{n-2}y_{n-2} + 2 \times 2^{n-3}y_{n-3} - 2^{n-3}y_{n-3} + \cdots + 2 \times 2y_1 \\ & - 2y_1 + 2 \times y_0 - y_0 + y_{-1} \end{aligned}$$

$$\begin{aligned} = {} & -2^{n-1}y_{n-1} + 2^{n-1}y_{n-2} - 2^{n-2}y_{n-2} + 2^{n-2}y_{n-3} - 2^{n-3}y_{n-3} + \cdots + 2^2 y_1 - 2y_1 \\ & + 2y_0 - y_0 + y_{-1} \end{aligned}$$

$$= 2^{n-1}(-y_{n-1} + y_{n-2}) + 2^{n-2}(-y_{n-2} + y_{n-3}) + \cdots 2(-y_1 + y_0) + (-y_0 + y_{-1})$$

部分和累加次数是否减少?

乘法器设计

- 如何减少部分和累加的次数?

$$y = -2 \times 2^{n-2} y_{n-1} + 2^{n-2} y_{n-2} + 2^{n-3} y_{n-3} + 2^{n-3} y_{n-3} - 2^{n-3} y_{n-3} + 2^{n-4} y_{n-4}$$
$$+ 2^{n-5} y_{n-5} + 2^{n-5} y_{n-5} - 2^{n-5} y_{n-5} + \cdots + 2^3 y_3 - 2^3 y_3 + 2^2 y_2 + 2 y_1$$
$$+ 2 y_1 - 2 y_1 + y_0 + y_{-1}$$

$$= -2 \times 2^{n-2} y_{n-1} + 2^{n-2} y_{n-2} + 2 \times 2^{n-3} y_{n-3} - 2 \times 2^{n-4} y_{n-3} + 2^{n-4} y_{n-4}$$
$$+ 2 \times 2^{n-5} y_{n-5} - 2 \times 2^{n-6} y_{n-5} + \cdots - 2 \times 2^2 y_3 + 2^2 y_2 + 2 \times 2 y_1 - 2 y_1$$
$$+ y_0 + y_{-1}$$

$$= -2 \times 2^{n-2} y_{n-1} + 2^{n-2} y_{n-2} + 2^{n-2} y_{n-3} - 2 \times 2^{n-4} y_{n-3} + 2^{n-4} y_{n-4} + 2^{n-4} y_{n-5}$$
$$- 2 \times 2^{n-6} y_{n-5} + \cdots - 2 \times 2^2 y_3 + 2^2 y_2 + 2^2 y_1 - 2 y_1 + y_0 + y_{-1}$$

$$= 2^{n-2}(-2y_{n-1} + y_{n-2} + y_{n-3}) + 2^{n-4}(-2y_{n-3} + y_{n-4} + y_{n-5}) + \cdots + 2^2(-2y_3 +$$
$$y_2 + y_1) + (-2y_1 + y_0 + y_{-1})$$

部分和累加次数是否减少?

乘法器设计

- 如何减少部分和累加的次数 – 布斯编码（Radix-2^r）

  - PPi = 3Y时，可以用-Y表示并在下一级部分积中加4Y
    通过这种方式，部分积的计算中只用到了移位和补码计算
  - 相似的，PPi = 2Y时，可以用-2Y表示并在下一级部分积中加4Y

Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$SINGLE_i$	$DOUBLE_i$	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0\ (=0)$	0	0	1

乘法器设计

- 如何减少部分和累加的次数 – 布斯编码（Radix-2^r）

布斯编码的几点要求：

- 乘数、被乘数、结果均为补码
- 乘法计算前应在乘数末尾补零
- 被乘数双符号位
- 符号位参与计算

Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$SINGLE_i$	$DOUBLE_i$	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0\ (=0)$	0	0	1

假设计算 Y x Q = -6 x -7，Q 是乘数，Y 是被乘数（4bit）

1、Y = -6 =1010　Q = -7 = 1001　-Y = 6 = 0110

2、乘数 Q 后补零，Q = 10010

3、被乘数双符号位，Y = 11010，-Y = 00110

3、乘法步骤（A为部分和、Q为乘数）

Step 1：Q = 10010

A = 11111010　Q = 1001　Q-1 = 0　补码符号扩展

Step 2：Q = 10010

A = 00110000　Q =1001　Q-1 = 0　左移补零

结果：11111010 （-6）+ 00110000 （48） = 42

乘法器设计

- 如何减少部分和累加的次数 – 布斯编码（Radix-2^r）

布斯编码的几点要求：

- 乘数、被乘数、结果均为补码
- 乘法计算前应在乘数末尾补零
- 被乘数双符号位
- 符号位参与计算

Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	SINGLE$_i$	DOUBLE$_i$	NEG$_i$
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0\ (=0)$	0	0	1

假设计算 Y x Q = -6 x 7，Q 是乘数，Y 是被乘数（6bit）

1、Y = -6 =111010　Q = 7 = 000111　-Y = 6 = 000110

2、乘数 Q 后补零，Q = 0001110

3、被乘数双符号位，Y = 1111010，-Y = 0000110

3、乘法步骤（A为部分和、Q为乘数）

Step 1：Q = 0001110

A = 000000000110　Q = 000111　Q-1 = 0　补码符号扩展

Step 2：Q = 0001110

A = 111111010000　Q =000111　Q-1 = 0　左移/符号位扩展

Step3：Q = 0001110

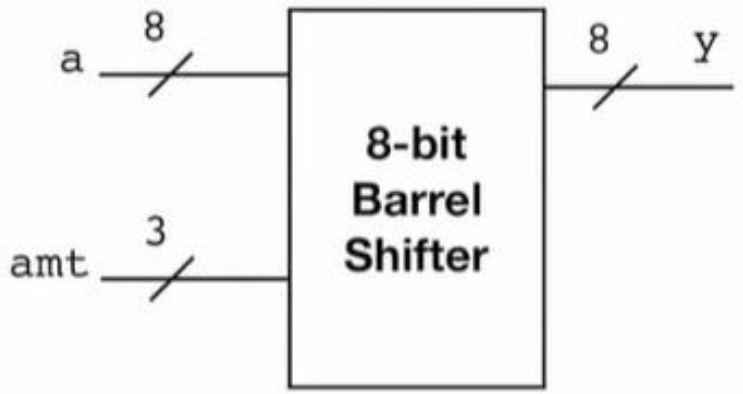
结果 = 000000000110（6）+ 111111010000（-48）= -42

位移器设计

- Shifter也是重要的数字电路模块之一

```
module barrel_shifter
    (
        input logic [7:0] a,
        input logic [2:0] amt,
        output logic [7:0] y
    );

    always_comb
        case(amt)
            3'b000: y = a;
            3'b001: y = {a[0], a[7:1]};
            3'b010: y = {a[1:0], a[7:2]};
            3'b011: y = {a[2:0], a[7:3]};
            3'b100: y = {a[3:0], a[7:4]};
            3'b101: y = {a[4:0], a[7:5]};
            3'b110: y = {a[5:0], a[7:6]};
            3'b111: y = {a[6:0], a[7]};
            default: y = a;
        endcase
endmodule
```

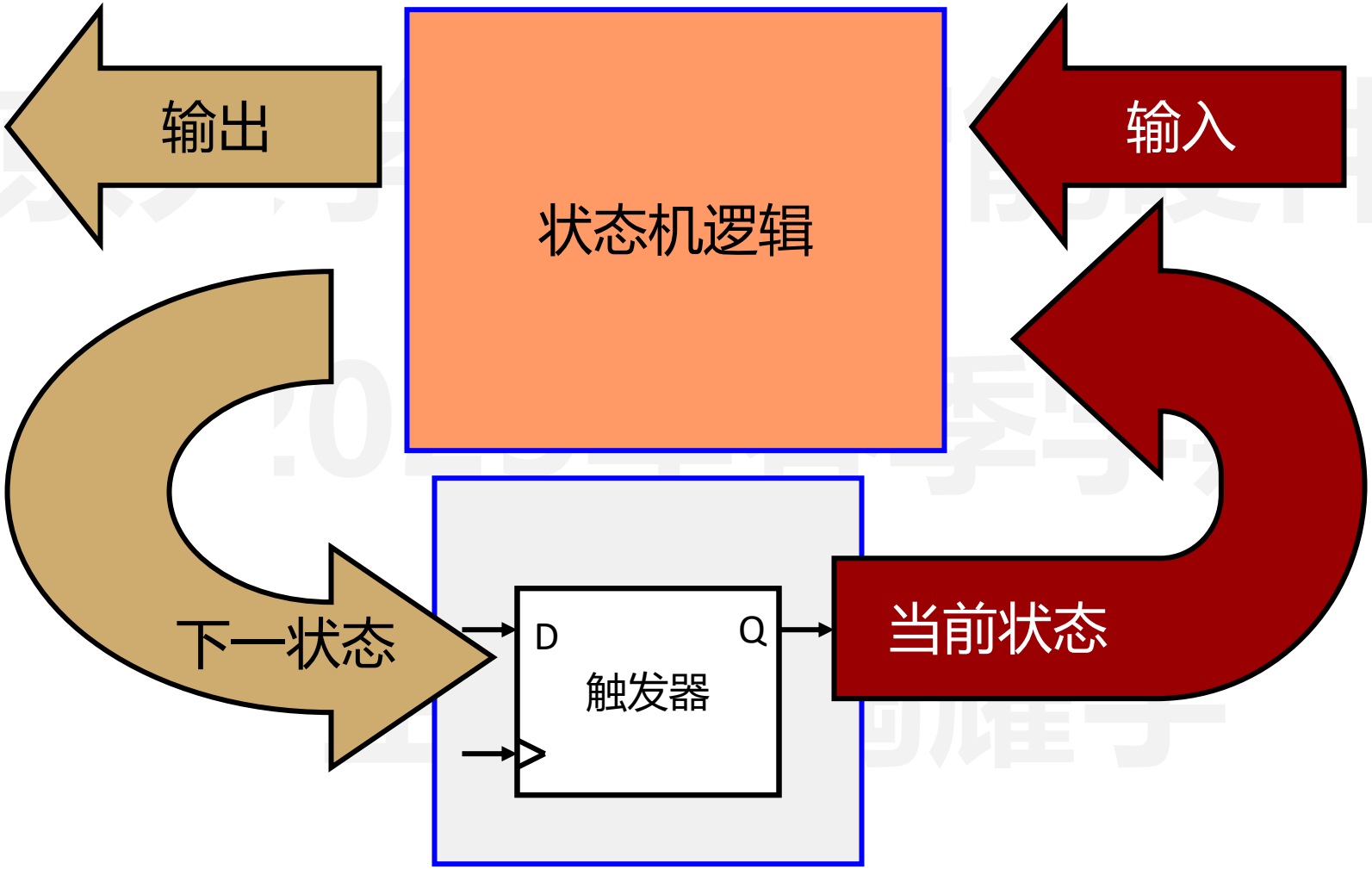


目 录

CONTENTS

为什么需要状态机

- 控制电路的基石



为什么需要状态机

- 控制电路的基石

状态机实例1 – 控制一个红绿灯

- 仅考虑红灯和绿灯，灯转换的速度不快于每次30s（0.033 Hz 时钟）
- 2个输出
  - NSlight：1=南北向为绿灯；0=南北向红灯
  - EWlight：1=东西向为绿灯；0=东西向为红灯
- 2个输入
  - Nscar：1=南北向有车等；0=南北向无车等
  - Ewcar：1=东西向有车等；0=南北向无车等
- 规则
  - 交通灯切换到另一个方向当且仅当另一方向有车等
  - 否则，保持当前交通灯不变

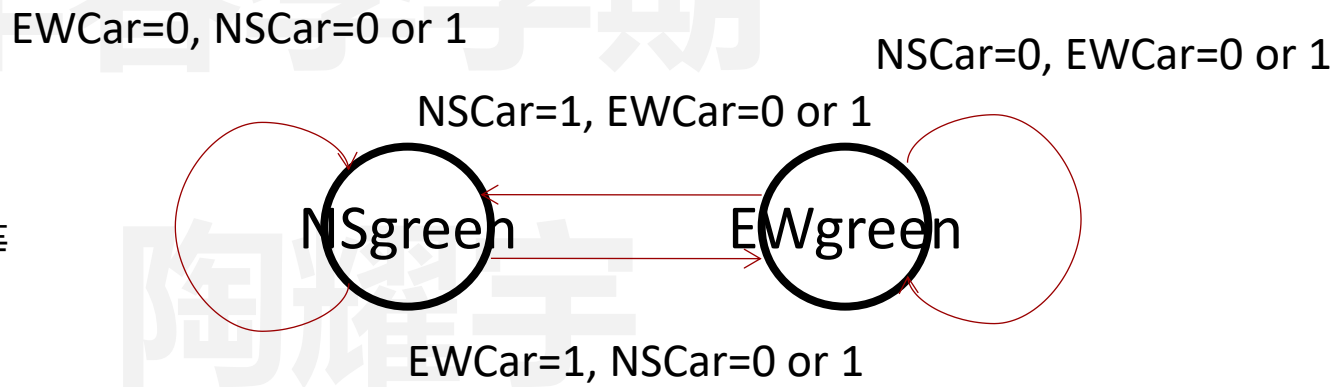
为什么需要状态机

- 控制电路的基石

状态机实例1 – 控制一个红绿灯

- 2个输出
  - NSlight：1=南北向为绿灯；0=南北向红灯
  - EWlight：1=东西向为绿灯；0=东西向为红灯
- 2个输入
  - Nscar：1=南北向有车等；0=南北向无车等
  - Ewcar：1=东西向有车等；0=南北向无车等
- 规则
  - 交通灯切换到另一个方向当且仅当另一方向有车等
  - 否则，保持当前交通灯不变

- 需要2个状态
  - Nsgreen、EWgreen



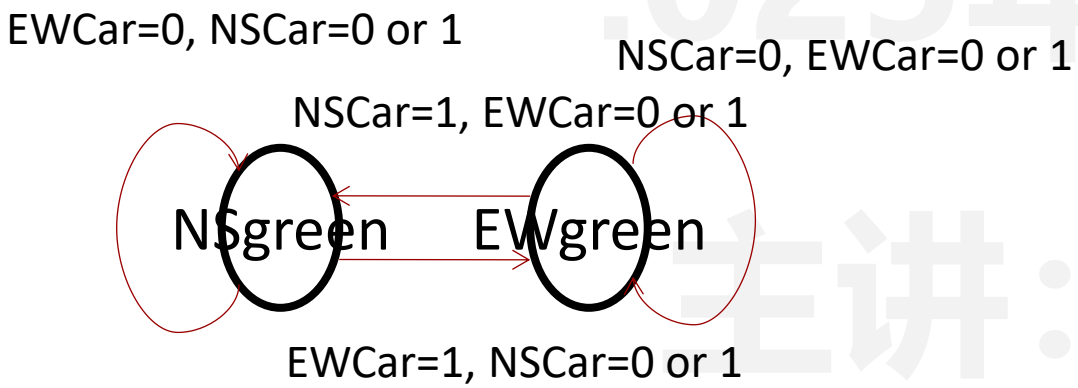
为什么需要状态机

- 控制电路的基石

状态机实例1 – 控制一个红绿灯

- 需要2个状态
  - Nsgreen、EWgreen

Current state	Inputs		Next state
	NScar	EWcar	
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen



Current state	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

$$\text{NextState} = (\overline{\text{CurrentState}} \cdot \text{EWcar}) + (\text{CurrentState} \cdot \overline{\text{NScar}})$$

$$\text{NSlite} = \overline{\text{CurrentState}}$$

$$\text{EWlite} = \text{CurrentState}$$

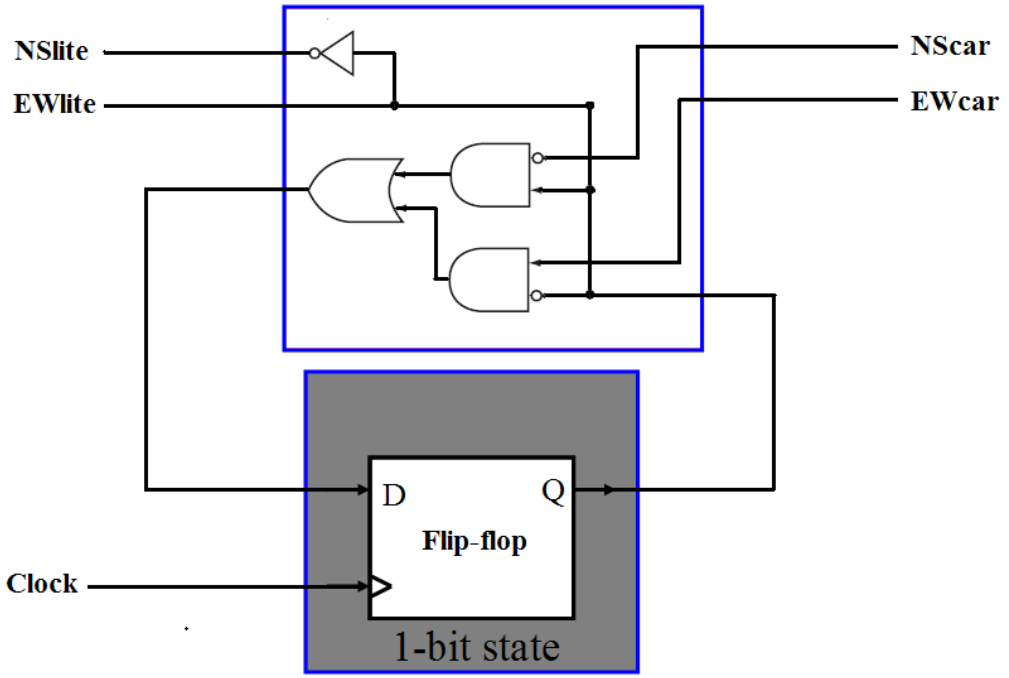
为什么需要状态机

- 控制电路的基石

状态机实例1 – 控制一个红绿灯

Current state	Inputs		Next state
	NScar	EWcar	
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

Current state	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1



$$\text{NextState} = (\overline{\text{CurrentState}} \cdot \text{EWcar}) + (\text{CurrentState} \cdot \overline{\text{NScar}})$$

$$\text{NSlite} = \overline{\text{CurrentState}}$$

$$\text{EWlite} = \text{CurrentState}$$

为什么需要状态机

- 控制电路的基石

Step 1 – 定义状态并画出状态转换图

Step 2 – 给每一个状态赋值并更新状态转换图

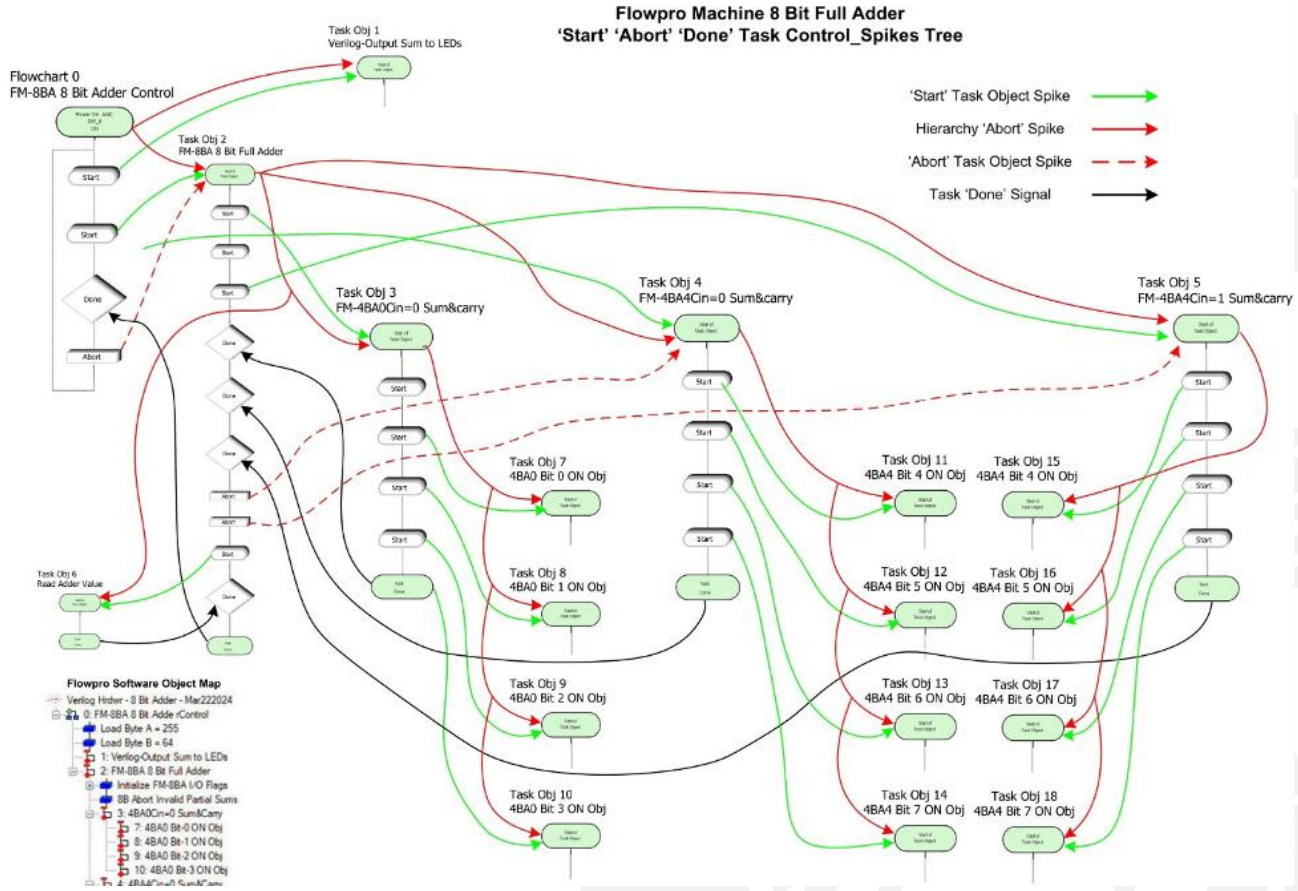
Step 3 – 根据状态转换图写出下一状态和输出的逻辑表达式

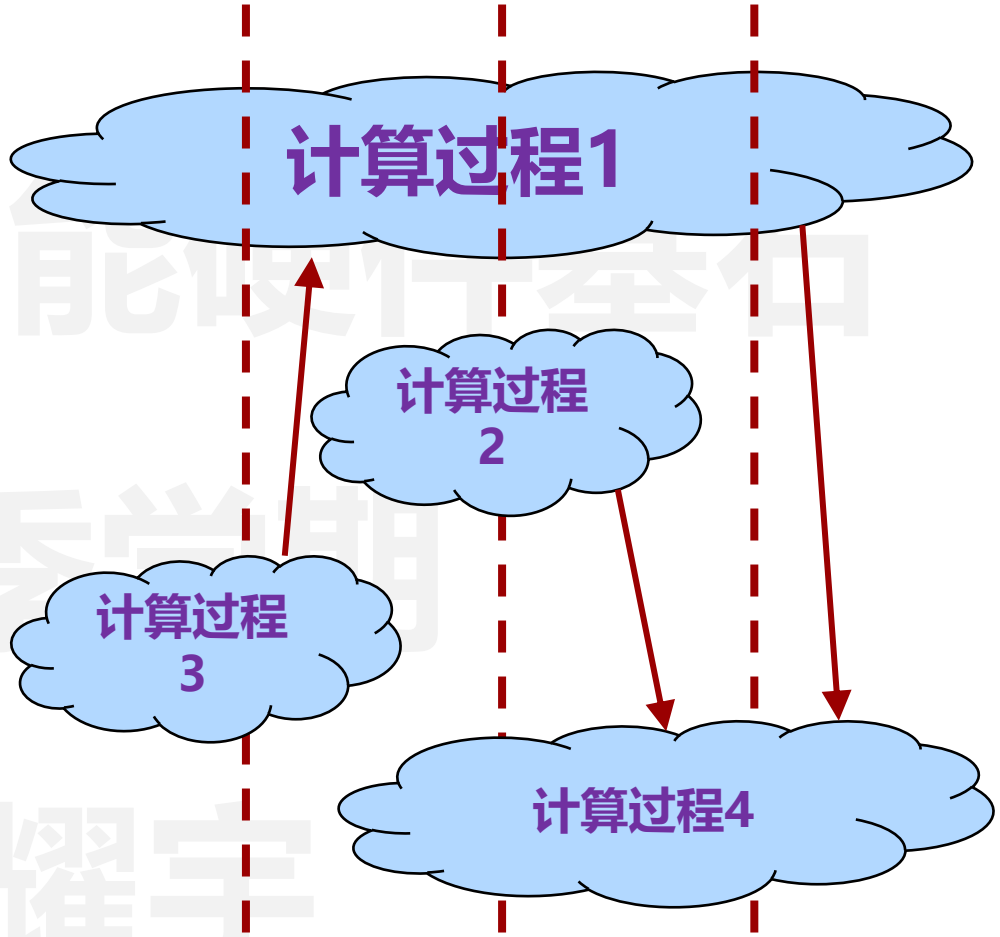
Step 4 – 画出实际电路图

状态将在每一个时钟上升沿更新

电路时序的基本概念

- 电路为什么需要一个时钟?



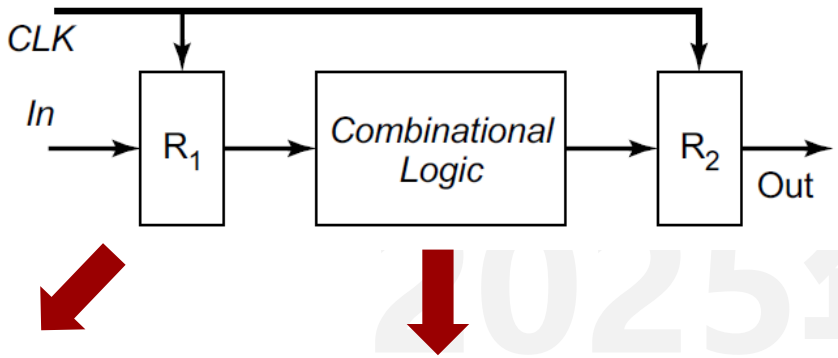


无时钟：非常难以控制每一个信号的有效时间

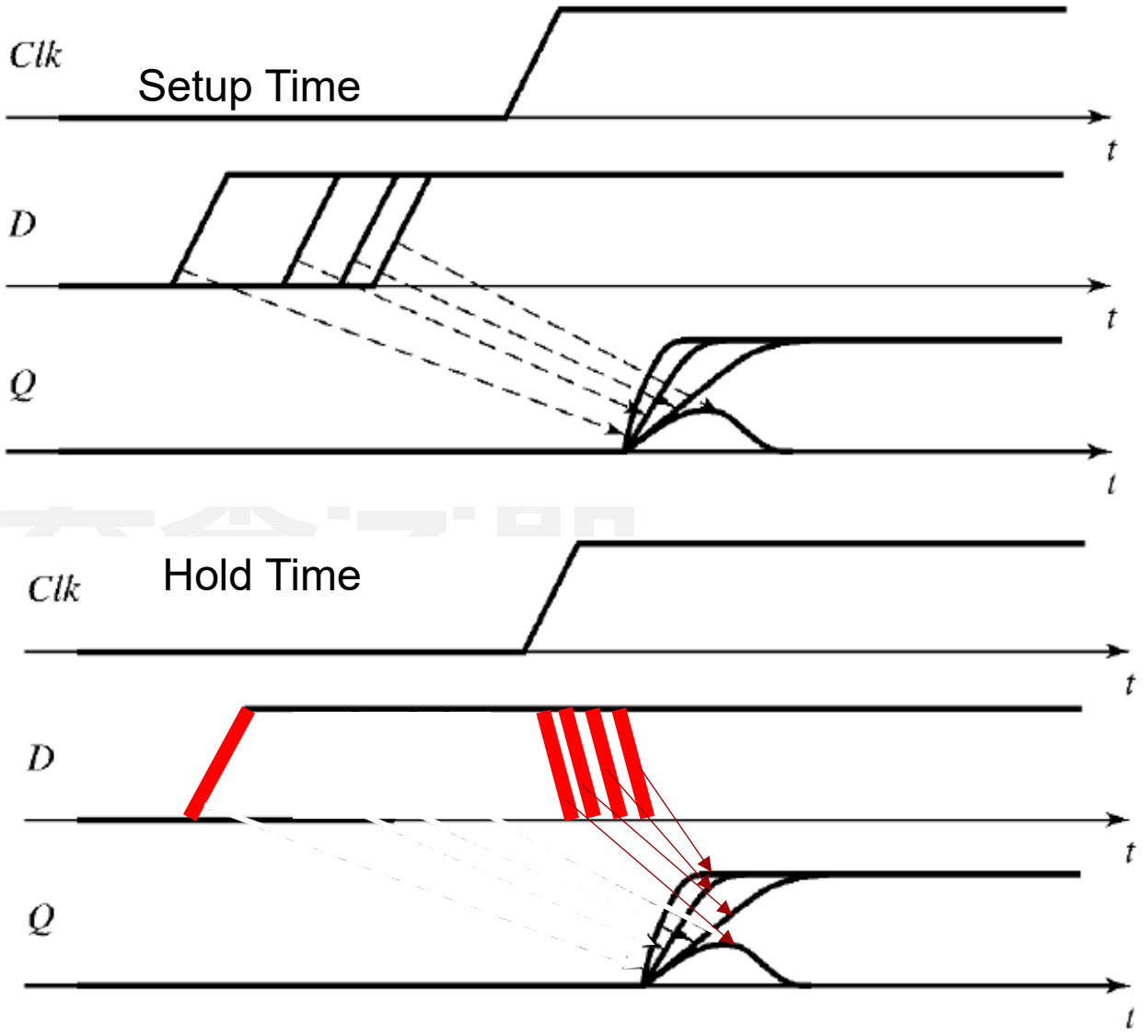
引入时钟：每隔一段计算将结果同步一次

电路时序的基本概念

- 同步时序（Synchronous Timing）

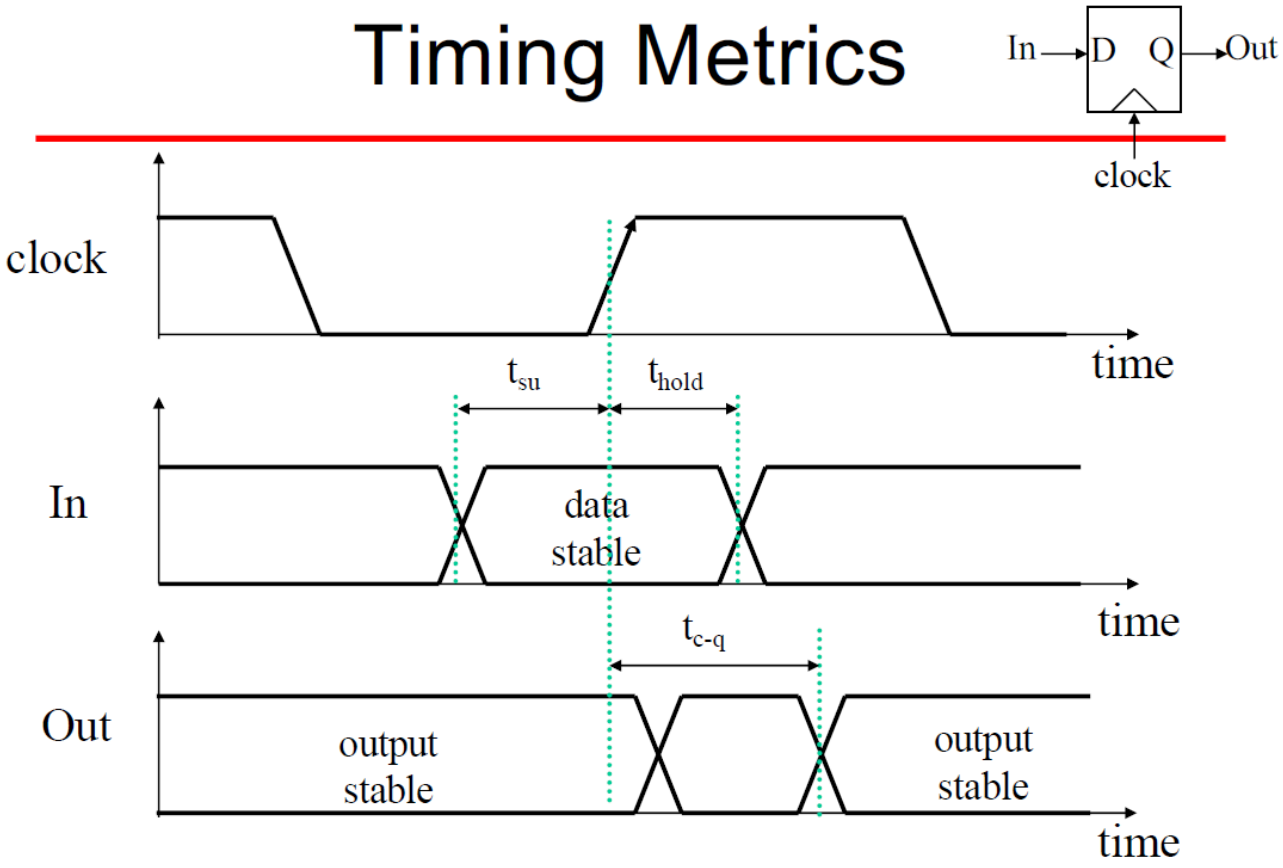


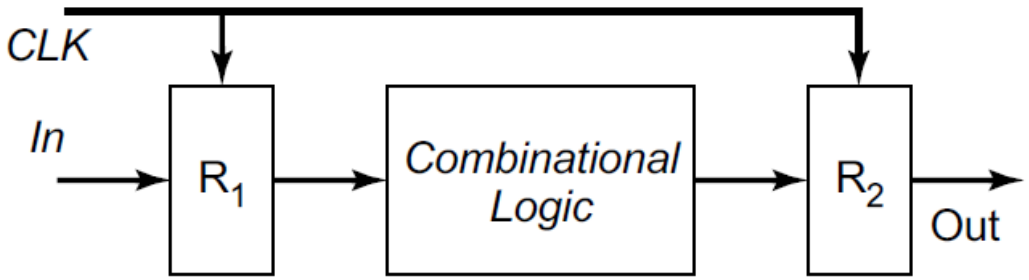
触发器　组合逻辑（各种逻辑门电路）
(Register)



电路时序的基本概念

- 同步时序（Synchronous Timing）





$$T_{c\text{-}q} + t_{plogic,min} \geq t_{hold}$$

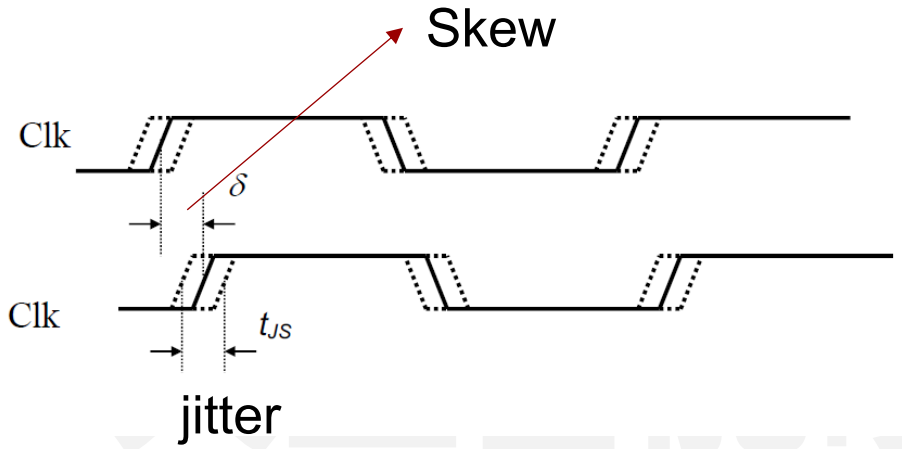
min

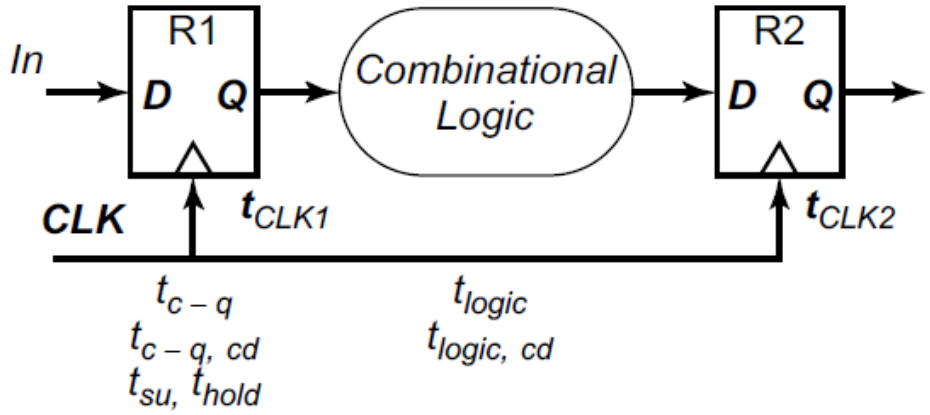
$$T \geq t_{c\text{-}q} + t_{plogic,max} + t_{su}$$

max

电路时序的基本概念

- 时钟的不稳定性

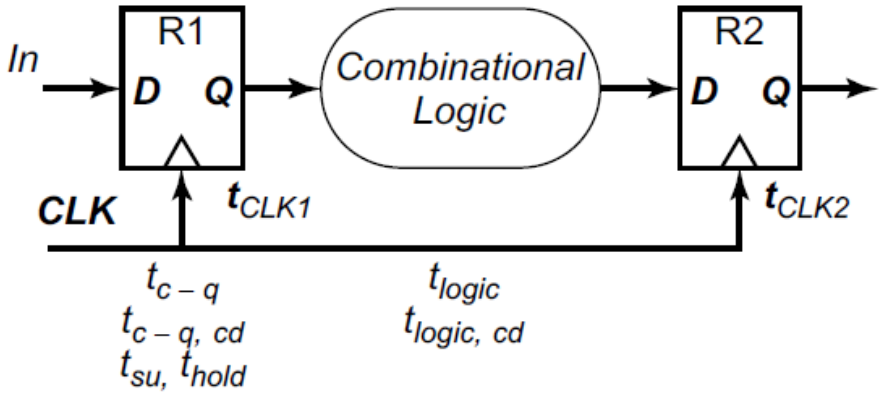




Minimum cycle time:

$T \geq t_{c\text{-}q} + t_{su} + t_{logic} - \delta$

最坏情况为接收边沿过早到达（negative δ）



Hold time constraint:

$t_{(c\text{-}q,\ cd)} + t_{(logic,\ cd)} > t_{hold} + \delta$

最坏情况为接收边沿过晚到达（正偏差）
数据和时钟之间的竞争
Cd：contamination delay（最快可能延迟）