EE141-Spring 2012 Digital Integrated Circuits

Lecture 13
Arithmetic

Administrativia

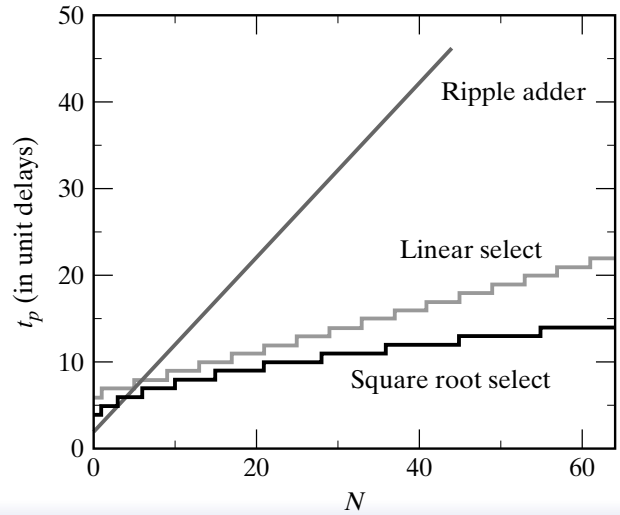
- THINK PROJECT!
- Make sure that all your lab reports are posted.

Class Material

- Last lecture
  - Adders
- Today's lecture
  - Multipliers
- Reading (Ch 11)

Adders

The quest for a better-than-linear adder



Generate (G) = AB
Propagate (P) = A ⊕ B
Kill = A B

A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

Look-Ahead Adders

$C_1 = g_0 + P_0 C_0$

$C_2 = g_1 + P_1 C_1$

$= g_1 + P_1(g_0 + P_0 C_0) = g_1 + P_1 g_0 + P_1 P_0 C_0$

$C_3 = g_2 + P_2(g_1 + P_1(g_0 + P_0 C_0))$

define recursively

因为不同的g1,P0,P1值是这个表达式不一样，可以利用这个性质来改善加法器的结构，因此每个加法器做2/k bits的加法。或之前定义的是1bit的P,G，现在定义2/k bits 的P，H。

P = propagate
G = generate

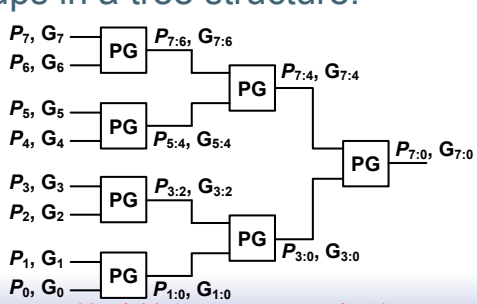
Logarithmic (Tree) Adders – Basic Idea

- "Look ahead" across groups of multiple bits to figure out the carry
  - Example with two bit groups: 这里定义了2bits的P，G。
    $P_{1:0} = P_1 \cdot P_0$, $G_{1:0} = G_1 + P_1 \cdot G_0$, → $C_{out1} = G_{1:0} + P_{1:0} \cdot C_{in}$

Logarithmic (Tree) Adders – Basic Idea

- "Look ahead" across groups of multiple bits to figure out the carry
  - Example with two bit groups:
    $P_{1:0} = P_1 \cdot P_0$, $G_{1:0} = G_1 + P_1 \cdot G_0$, → $C_{out1} = G_{1:0} + P_{1:0} \cdot C_{in}$
- Combine these groups in a tree structure:
  - Delay is now ~$\log_2(N)$
  - Instead of ~N

Generate (G) = AB
Propagate (P) = A ⊕ B
Kill = A B

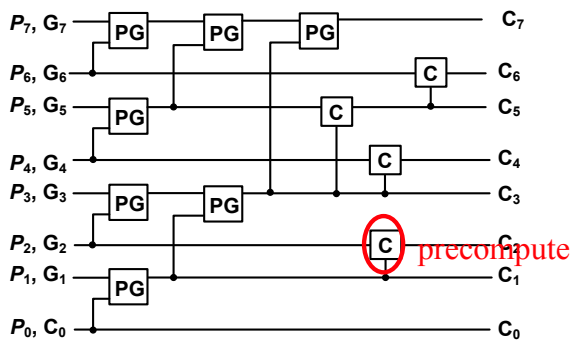


A	B	C_i	S	C_o	Carry status
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

Pi和Gi的计算仅于Ai和Bi有关。

Rest of the Tree

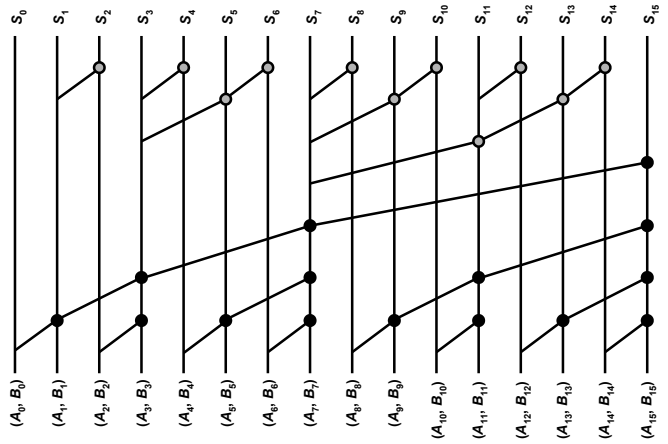
- Previous picture shows only half of the algorithm
  - Need to generate carries at individual bit positions too



Many Kinds of Tree Adders

- Many ways to construct these tree (or "carry lookahead") adders
  - Many of these variations named after the people who first came up with them
- Most of these vary three basic parameters:
  - Radix: how many bits are combined in each PG gate
    基数
    - Previous example was radix 2; often go up to radix 4
    一般选择2或者4，基数太大，门会变的太复杂，反而不会达到最初的目的。
  - Tree Depth: how many stages of logic you go through to get the final carry. Must be at least $\log_{Radix}(N)$
  - Fanout: Maximum logical branching in the tree

Tree Adders

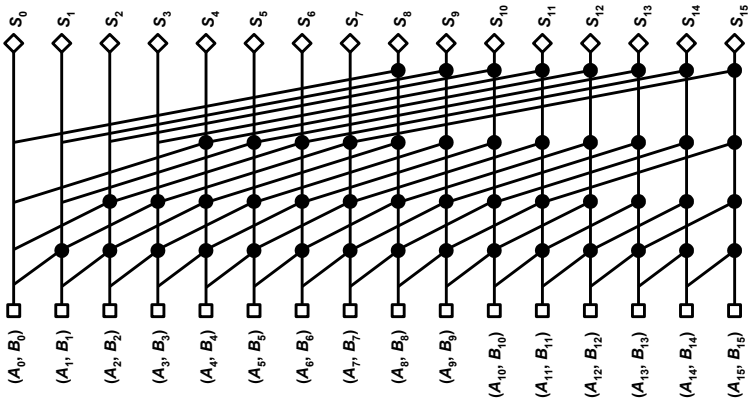


Brent-Kung Tree

但是这里的部分节点有太多fanout，fanout不是固定的，这会determine our delay

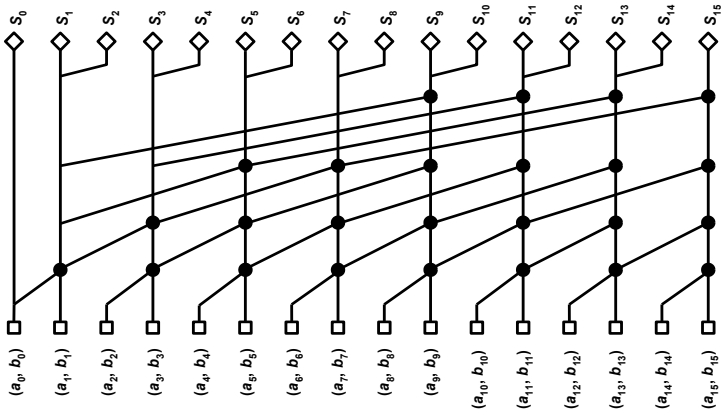
因此为了减少部分节点的fanout，我们generate all intermediate signals。

Tree Adders



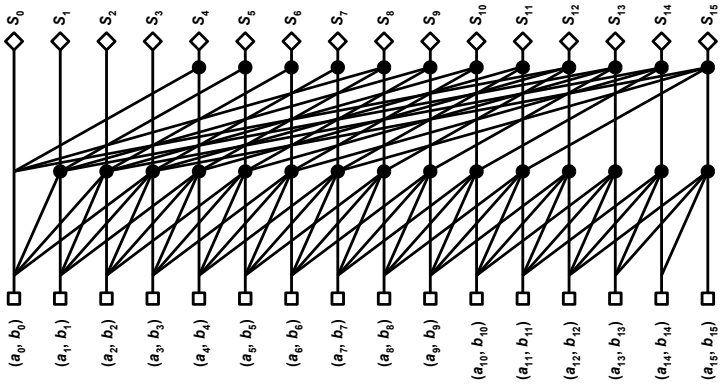
16-bit radix-2 Kogge-Stone tree

Tree Adders



16-bit radix-2 sparse tree with sparseness of 2

Tree Adders



16-bit radix-4 Kogge-Stone Tree

Arithmetic Beyond Adders

Other Logic Functions

可能存在一种结构使加法器计算加法时与输入的位数无关，例如下面这个例子：

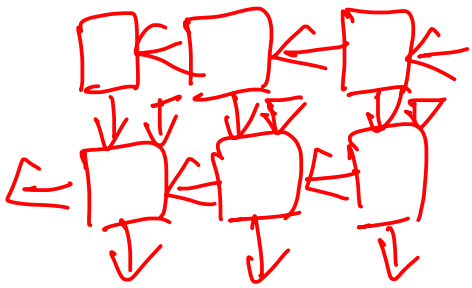
$\{0, 1, \bar{1}\}$, $1 \Rightarrow 2^{k-1}$ $\bar{1} \Rightarrow -2^{k-1}$

用这三个数表示同一个二进制数可以有多种不同的表示方法，这就是representation redundant

如果我们做A和B的加法，我们可以选择适当的表示方式，使得A和B的1不同时出现在同一位上，或$\bar{1}$不同时出现在同一位上，避免进位的出现，那么这个加法的计算所需时间就是一个常数。

Chaining Adders

a+b+c



Multipliers

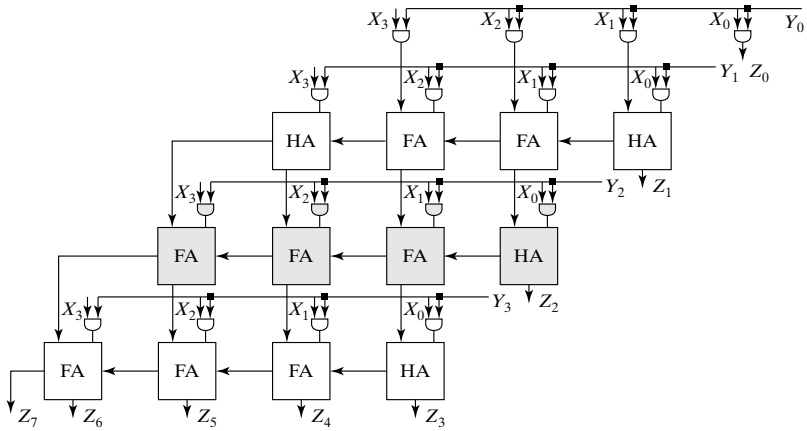
Binary Multiplication

	1 0 1 0 1 0	Multiplicand
×	1 0 1 1	Multiplier
	1 0 1 0 1 0	Partial products
	1 0 1 0 1 0	
	0 0 0 0 0 0	
+	1 0 1 0 1 0	
	1 1 1 0 0 1 1 1 0	Result

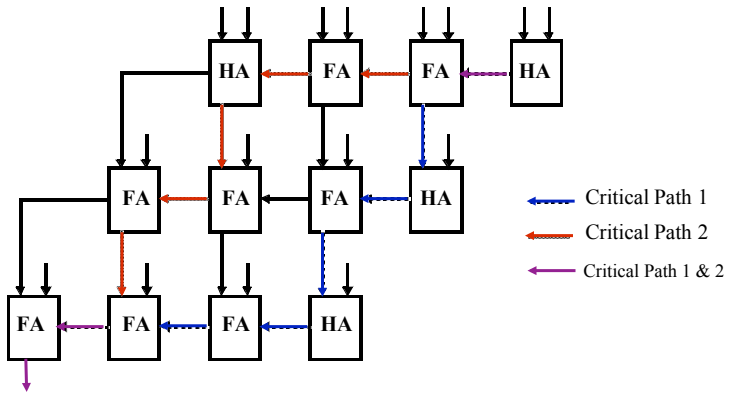
通常考虑电路延迟时，一般考虑critical path。因为这将会决定电路的实际延迟。但是，下面这个例子的critical path不只一条，这需要一些特别的方法：例如改变电路结构，使critical path只有一条，然后使用已经知道的结构或方法加速critical path。

The Array Multiplier

当没有进位时，FA可以换成HA。



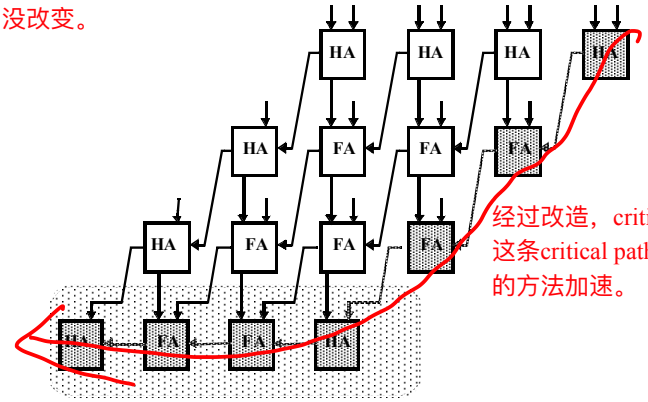
The M-by-N Array Multiplier: Critical Path



$$t_{mult} \approx \left[(M-1)+(N-2)\right]\cdot t_{carry} + (N-1)\cdot t_{sum} + t_{and}$$

Carry-Save Multiplier

这个比上面的多了一层，但实际结果是一样的。除此以外，唯一的不同是，进位的输出层改变了，但是输出列没改变。



经过改造，critical path只剩下一条，这条critical path可以使用加速加法器的方法加速。

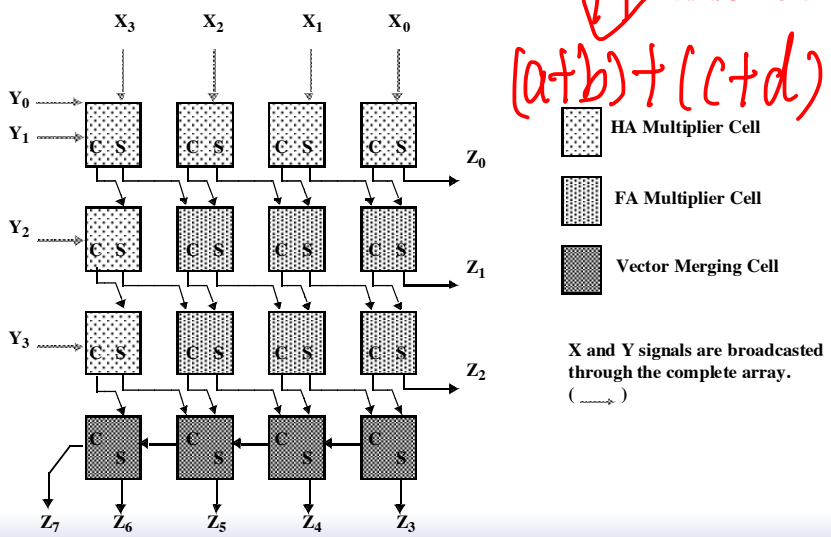
$$t_{mult} = t_{and} + (N-1)\cdot t_{carry} + t_{merge}$$

Multiplier Floorplan

$a+(b+(c+d))$

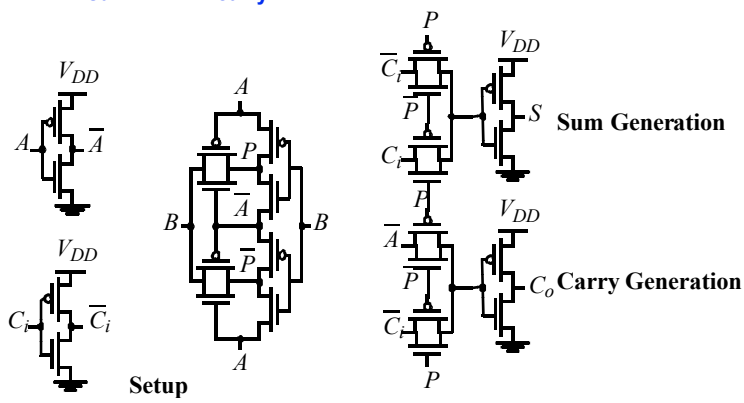
改变同一列的bit相加的先后顺序

$(a+b)+(c+d)$

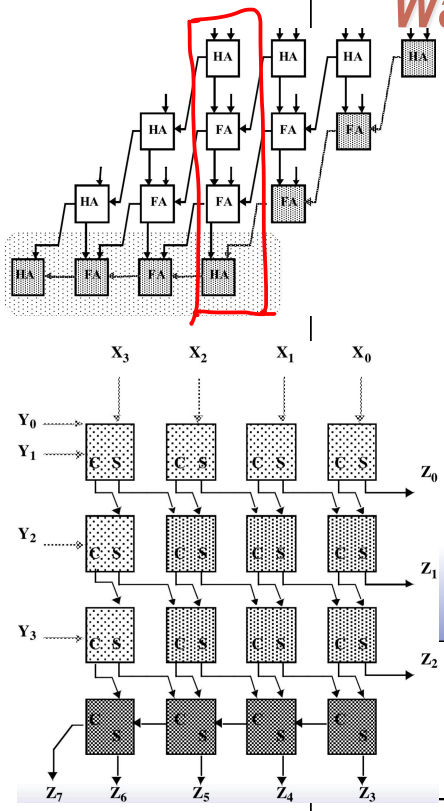


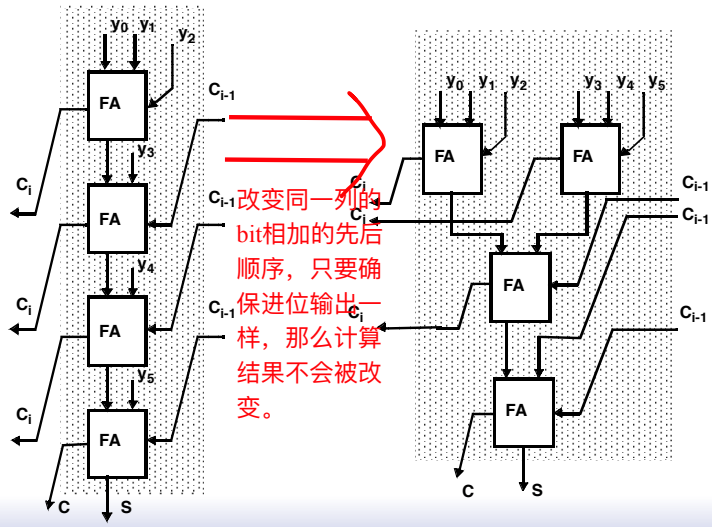
Transmission-Gate Full Adder

Balanced t_{sum} and t_{carry}



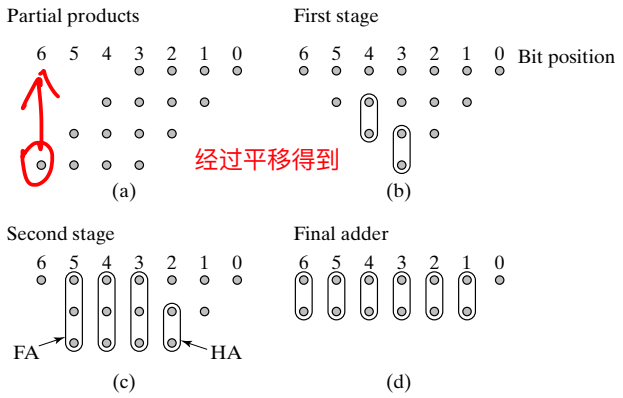
Wallace-Tree Multiplier





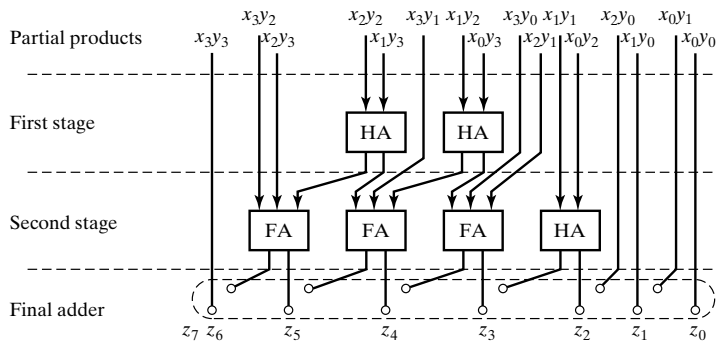
改变同一列的bit相加的先后顺序，只要确保进位输出一样，那么计算结果不会被改变。

Wallace-Tree Multiplier



经过平移得到

Wallace-Tree Multiplier



Boot multiplier {0, 1, T} Booth encode.

x

y (判断是否需要移位或取反)

Multipliers – Summary

- Optimization constraints different than in binary adder
  - Once again:
    - Need to identify critical path
    - And find ways to use parallelism to reduce it
- Other possible techniques
  - Logarithmic versus linear (Wallace Tree Mult)
  - Data encoding (Booth)
  - Pipelining

First glimpse at system level optimization