EECS150 - Digital Design
## Lecture 23 - Arithmetic Blocks, Part 1

April 11, 2010

John Wawrzynek

Carry-ripple Adder Revisited

- Each cell:

 $r_i = a_i$ XOR b_i XOR c_{in}

    $c_{out} = a_i c_{in} + a_i b_i + b_i c_{in} = c_{in}(a_i + b_i) + a_i b_i$
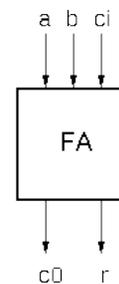
"Full adder cell"

- 4-bit adder:
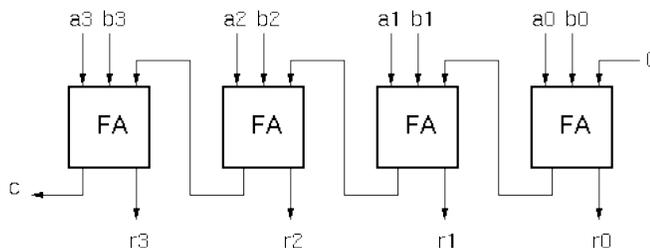
- What about subtraction?

Subtractor

A - B = A + (-B)

How do we form -B?
1. complement B
2. add 1

bn-1   b1   b0

an-1   a1   a0   SUB

cout   n-bit adder   cin

sn-1   s1   s0
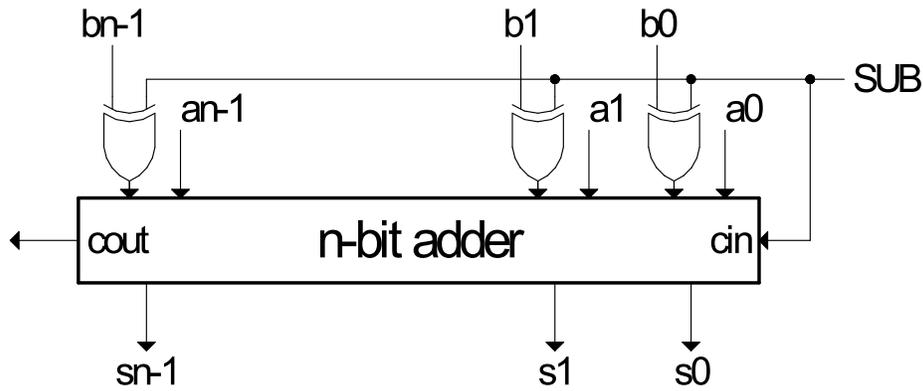
Delay in Ripple Adders

- Ripple delay amount is a function of the data inputs:
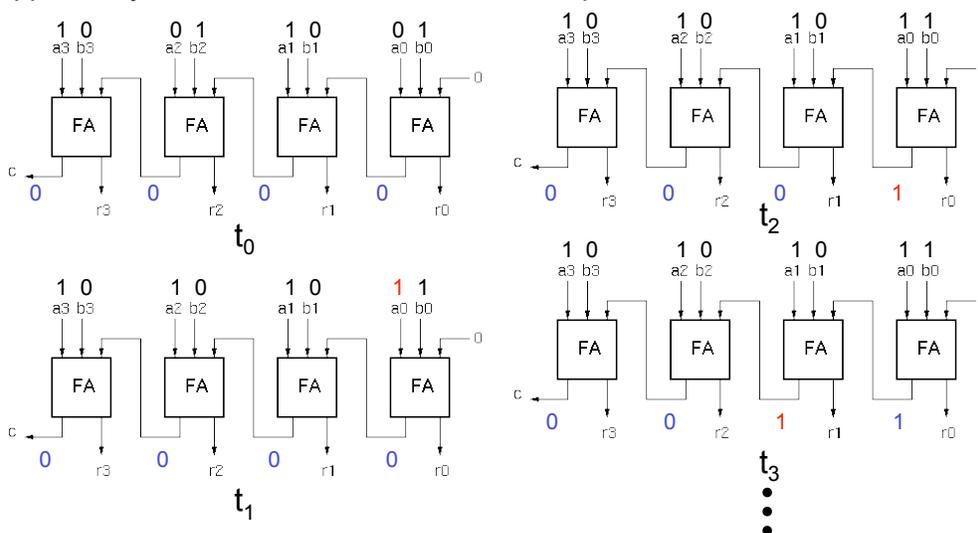
- However, we usually only consider the worst case delay on the critical path. There is usually at least one set of input data that exposes the worst case delay.

Adders (cont.)

Ripple Adder
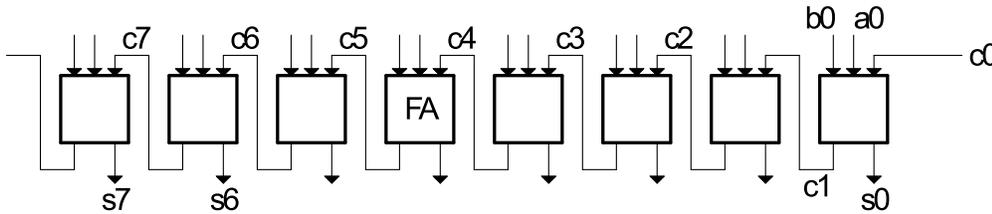
Ripple adder is inherently slow because, in general s7 must wait for c7 which must wait for c6 …

$$T \propto n, \quad Cost \propto n$$

How do we make it faster, perhaps with more cost?

Carry Select Adder
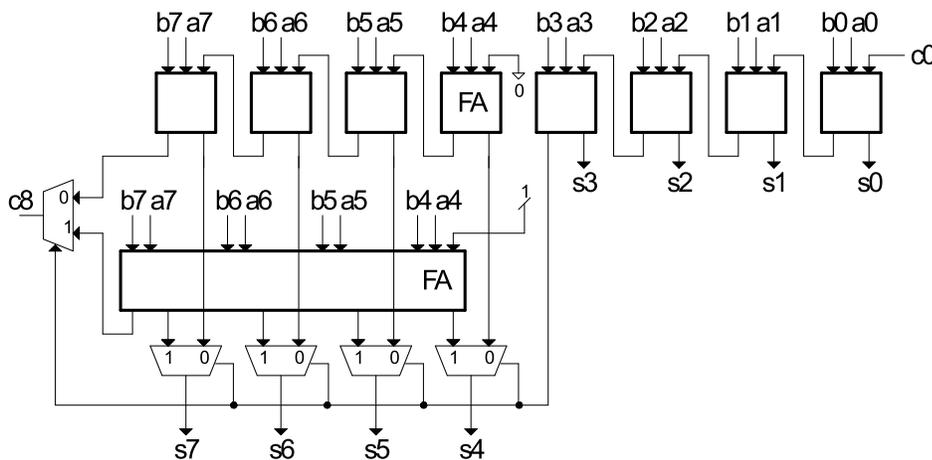
$$T = T_{ripple\_adder} / 2 + T_{MUX}$$

$$COST = 1.5 * COST_{ripple\_adder} + (n/2 + 1) * COST_{MUX}$$

Carry Select Adder

- Extending Carry-select to multiple blocks
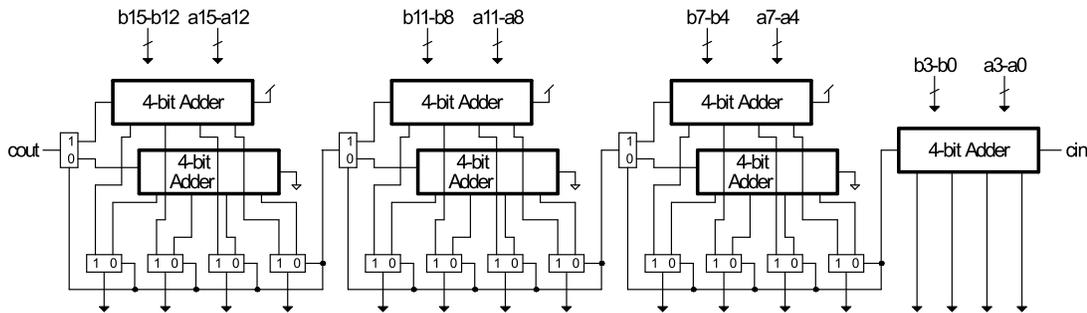

- What is the optimal # of blocks and # of bits/block?
  - If # blocks too large delay dominated by total mux delay
  - If # blocks too small delay dominated by adder delay

$$\sqrt{N} \text{ stages of } \sqrt{N} \text{ bits}$$

$T \propto$ sqrt(N),
Cost $\approx$ 2*ripple + muxes

Carry Select Adder
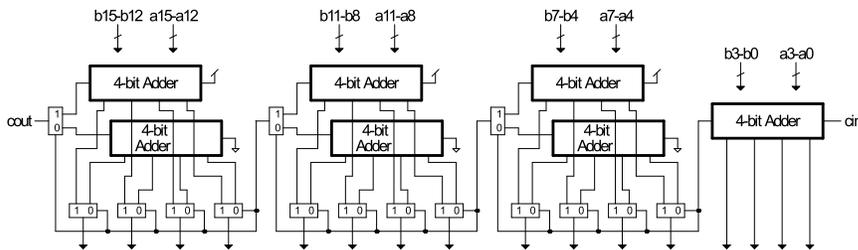
- Compare to ripple adder delay:

  T_{total} = 2 sqrt(N) T_{FA} – T_{FA}, assuming T_{FA} = T_{MUX}

  For ripple adder T_{total} = N T_{FA}

  "cross-over" at N=3, Carry select faster for any value of N>3.

- Is sqrt(N) really the optimum?
  - From right to left increase size of each block to better match delays
  - Ex: 64-bit adder, use block sizes [12 11 10 9 8 7 7]

- How about recursively defined carry select?

Carry Look-ahead Adders

- In general, for n-bit addition best we can achieve is
   delay α log(n)
- How do we arrange this? (think trees)
- First, reformulate basic adder stage:

a b c_i	c_{i+1}	s	
0 0 0	0	0	carry "kill"
0 0 1	0	1	$k_i = a_i' \, b_i'$
0 1 0	0	1	
0 1 1	1	0	carry "propagate"
1 0 0	0	1	$p_i = a_i \oplus b_i$
1 0 1	1	0	
1 1 0	1	0	carry "generate"
1 1 1	1	1	$g_i = a_i \, b_i$

$$c_{i+1} = g_i + p_i c_i$$
$$s_i = p_i \oplus c_i$$

Carry Look-ahead Adders

- Ripple adder using p and g signals:

$$p_i = a_i \oplus b_i$$
$$g_i = a_i \, b_i$$

c_0

$a_0 \rightarrow$ □ $\rightarrow p_0 \rightarrow$ | $s_0 = p_0 \oplus c_0$ | $\rightarrow s_0$
$b_0 \rightarrow$ □ $\rightarrow g_0 \rightarrow$ | $c_1 = g_0 + p_0 c_0$ |

$a_1 \rightarrow$ □ $\rightarrow p_1 \rightarrow$ | $s_0 = p_1 \oplus c_1$ | $\rightarrow s_1$
$b_1 \rightarrow$ □ $\rightarrow g_1 \rightarrow$ | $c_2 = g_1 + p_1 c_1$ |

$a_2 \rightarrow$ □ $\rightarrow p_2 \rightarrow$ | $s_2 = p_2 \oplus c_2$ | $\rightarrow s_2$
$b_2 \rightarrow$ □ $\rightarrow g_2 \rightarrow$ | $c_3 = g_2 + p_2 c_2$ |

$a_3 \rightarrow$ □ $\rightarrow p_3 \rightarrow$ | $s_3 = p_3 \oplus c_3$ | $\rightarrow s_3$
$b_3 \rightarrow$ □ $\rightarrow g_3 \rightarrow$ | $c_4 = g_3 + p_3 c_3$ |

c_4
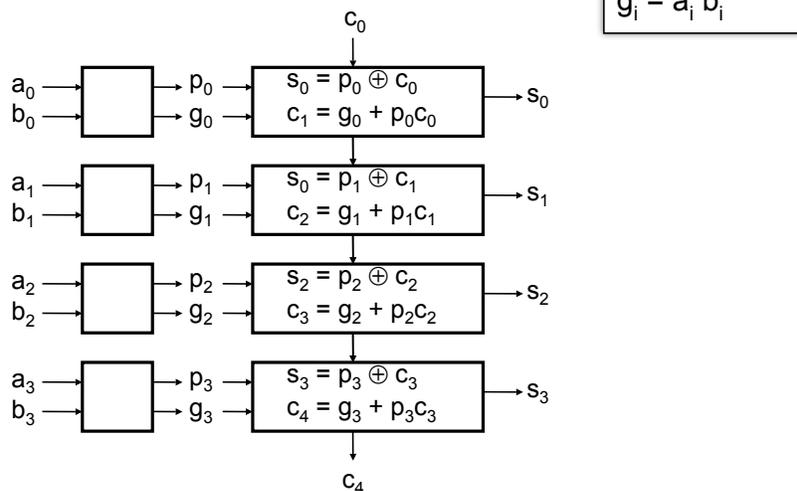
- So far, no advantage over ripple adder: $T \, \alpha \, N$

Carry Look-ahead Adders

- Expand carries:

c_0

$c_1 = g_0 + p_0\, c_0$

$c_2 = g_1 + p_1 c_1 = g_1 + p_1 g_0 + p_1 p_0 c_0$

$c_3 = g_2 + p_2 c_2 = g_2 + p_2 g_1 + p_1 p_2 g_0 + p_2 p_1 p_0 c_0$

$c_4 = g_3 + p_3 c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + \ldots$

$$.$$
$$.$$
$$.$$

- Why not implement these equations directly to avoid ripple delay?
  - Lots of gates. Redundancies (full tree for each).
  - Gate with high # of inputs.
- Let's reorganize the equations.

Carry Look-ahead Adders

- "Group" propagate and generate signals:

$P = p_i\, p_{i+1} \cdots p_{i+k}$

$G = g_{i+k} + p_{i+k} g_{i+k-1} + \ldots + (p_{i+1} p_{i+2} \cdots p_{i+k}) g_i$
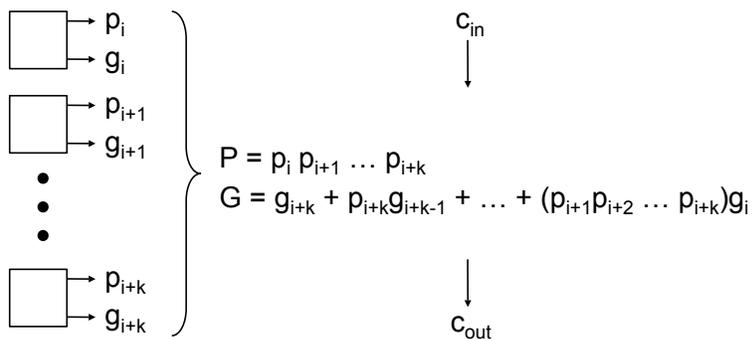
- P true if the group as a whole propagates a carry to c_{out}
- G true if the group as a whole generates a carry

$$c_{out} = G + P c_{in}$$

- Group P and G can be generated hierarchically.

c_0 <u>Carry Look-ahead Adders</u>

9-bit Example of hierarchically generated P and G signals:

a_0
b_0
a_1
b_1
a_2
b_2

a → P_a , G_a

$$P = P_a P_b P_c$$

$c_3 = G_a + P_a c_0$

a_3
b_3
a_4
b_4
a_5
b_5

b → P_b , G_b

$c_6 = G_b + P_b c_3$

a_6
b_6
a_7
b_7
a_8
b_8

c → P_c , G_c

$$G = G_c + P_c G_b + P_b P_c G_a$$

$$c_9 = G + P c_0$$

8-bit Carry Look-ahead Adder

$p = a \oplus b$
$g = ab$

$s = p \oplus c_i$

$c_{i+1} = g + c_i p$

$P = P_a P_b$
$G = G_b + G_a P_b$

$C_{out} = G + c_{in} P$

8-bit Carry Look-ahead Adder with 2-input gates.

c_0

p_0
s_0 g_0

$c_1 = g_0 + p_0 c_0$

p_1
s_1 g_1

$c_2 = G_8 + P_8 c_0$

c_2

p_1
s_2 g_2

$c_3 = g_2 + p_2 c_2$

p_3
s_3 g_3

c_4

p_4
s_4 g_4

$c_5 = g_4 + p_4 c_4$

p_5
s_5 g_5

$c_6 = G_a + P_a c_4$

c_6

p_6
s_6 g_6

$c_7 = g_6 + p_6 c_6$

p_7
s_7 g_7

$\downarrow c_0$

$P_8 = p_0 p_1$

$G_8 = g_1 + p_1 g_0$

$P_9 = p_2 p_3$

$G_9 = g_3 + p_3 g_2$

$\downarrow c_4$

$P_a = p_4 p_5$

$G_a = g_5 + p_5 g_4$

$P_b = p_6 p_7$

$G_b = g_7 + p_7 g_6$

$\downarrow c_8$

$P_c = P_8 P_9$

$G_c = G_9 + P_9 G_8$

$c_4 = G_c + P_c c_0$

$P_e = P_c P_d$

$G_e = G_d + P_d G_c$

$c_8 = G_e + P_e c_0$

$P_d = P_a P_b$

$G_d = G_b + P_b G_a$

Carry look-ahead Wrap-up

- Adder delay $O(\log N)$ (up then down the tree).
- Cost? Energy per add?
- Can be applied with other techniques. Group P & G signals can be generated for sub-adders, but another carry propagation technique (for instance ripple) used within the group.
  - For instance on FPGA. Ripple carry up to 32 bits is fast (1.25ns), CLA used to extend to large adders. CLA tree quickly generates carry-in for upper blocks.
- Other more complex techniques exist that can bring the delay down below $O(\log N)$, but are only efficient for very wide adders.

Bit-serial Adder

n-bit shift registers
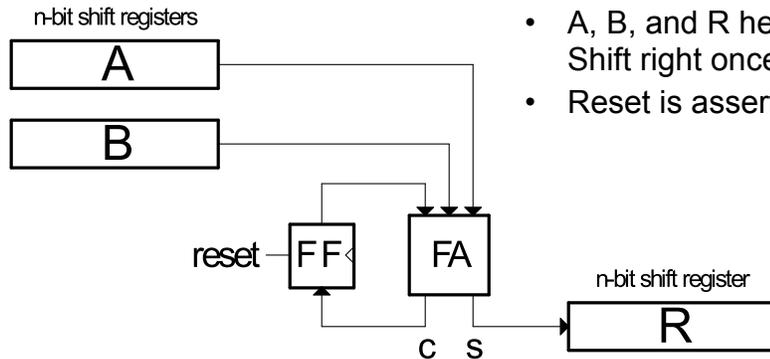
- A, B, and R held in shift-registers. Shift right once per clock cycle.
- Reset is asserted by controller.

n-bit shift register

- Addition of 2 n-bit numbers:
  - takes n clock cycles,
  - uses 1 FF, 1 FA cell, plus registers
  - the bit streams may come from or go to other circuits, therefore the registers might not be needed.

# Adders on the Xilinx Virtex-5

- Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions.
- Cin to Cout (per bit) delay = 40ps, versus 900ns for F to X delay.
- 64-bit add delay = 2.5ns.

UG190_5_04_032606

Virtex 5 Vertical Logic

COUT (To Next Slice)

Carry Chain Block (CARRY4)

CIN (From Previous Slice)

UG190_5_24_050506
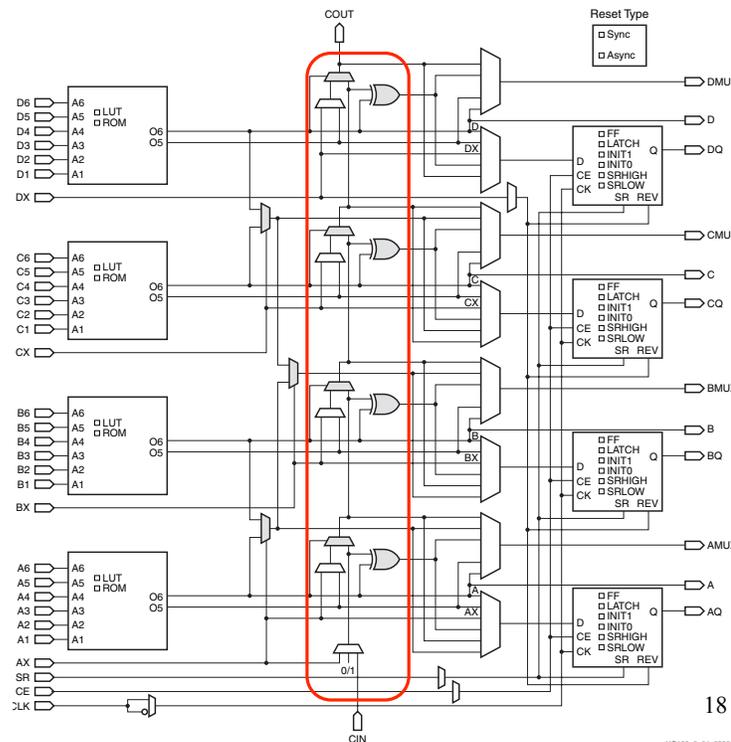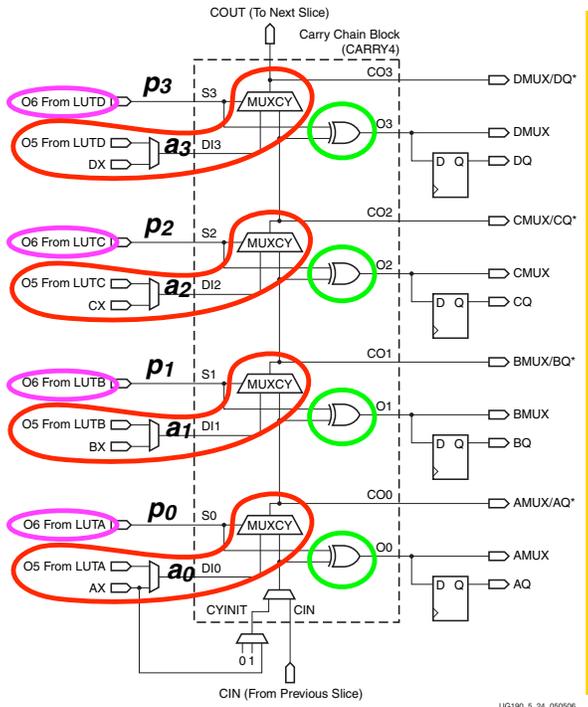
We can map ripple-carry addition onto carry-chain block.
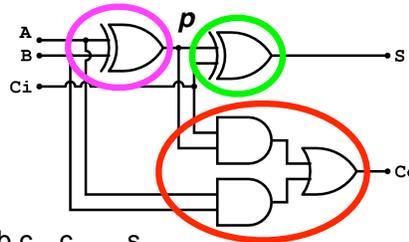
a b c$_i$	c$_{i+1}$	s
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1

$c_{out} = p_i\, c_{in} + a_i\, b_i$

$c_{out} = p_i\ ?\ c_{in} : a_i$

$p_i = a_i\ xor\ b_i$

The carry-chain block also useful for speeding up other adder structures and counters.

Adder Final Words

Type	Cost	Delay
Ripple	O(N)	O(N)
Carry-select	O(N)	O(sqrt(N))
Carry-lookahead	O(N)	O(log(N))

- Dynamic energy per addition for all of these is O(n).
- "O" notation hides the constants. Watch out for this!
- The cost of the carry-select is at least 2X the cost of the ripple. Cost of the CLA is probably at least 2X the cost of the carry-select.
- The actual multiplicative constants depend on the implementation details and technology.
- FPGA and ASIC synthesis tools will try to choose the best adder architecture automatically
  – assuming you specify addition using the "+" operator, as in "assign A = B + C"