# Carry lookahead

Can calculate carries in parallel

Tradeoff: need more hardware (space vs. time)

Boolean expression for carry-out:

$$c_{out} = \overline{x}yc_{in} + x\overline{y}c_{in} + xy\overline{c}_{in} + xyc_{in}$$

What does this expression look like?

$$z = \overline{a}bc + a\overline{b}c + ab\overline{c} + abc$$

How many inputs? 3

How many need to be true? 2

What is that function called?

Alternate expression for carry-out:

$$c_{out} = xy + xc_{in} + yc_{in}$$

This means that there is a carry whenever at least 2 of the bits are 1 (possibly all 3).

Call $c_{in}$ $c_i$ (carry-in for bit i) and $c_{out}$ $c_{i+1}$ (carry-in for bit i+1)

Distributive property:

$$c_{i+1} = x_iy_i + c_i(x_i + y_i)$$

Define 2 new terms:

$$g_i = x_iy_i$$

$$p_i = x_i + y_i$$

Then rewrite expression for $c_{i+1}$

$$c_{i+1} = g_i + p_ic_i$$

**$g_i$ is called the generate term**

**it always generates a carry out, if equal to 1**

**$p_i$ is called the propagate term.**

**it may generate a carry, depending on the carry-in ($c_i$)**

**if exactly one of $x_i$ or $y_i$ is 1, then the carry-in will determine the carry-out**

**How can we get rid of the dependency on the carry-in?**

**Let's look at the first carry:**

$$c_1 = g_0 + p_0c_0$$

**The next one is:**

$$c_2 = g_1 + p_1c_1$$

**Now we can plug in the expression for $c_1$, which we just calculated:**

$$c_2 = g_1 + p_1(g_0 + p_0c_0)$$
$$= g_1 + p_1g_0 + p_1p_0c_0$$

**How do we get the right side of this expression?**

**Since $g_1$ and $p_1$ depend only on $x_1$ and $y_1$,**

**and $g_0$ and $p_0$ depend only on $x_0$ and $y_0$,**

**and $c_0$ depends only on $x_0$ and $y_0$ (there is no carry-in for the addition of bits 0),**

**we can get $c_2$ right away without waiting for $c_1$!**

**Likewise,**

$$c_3 = g_2 + p_2c_2$$
$$= g_2 + p_2(g_1 + p_1g_0 + p_1p_0c_0)$$
$$= g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$

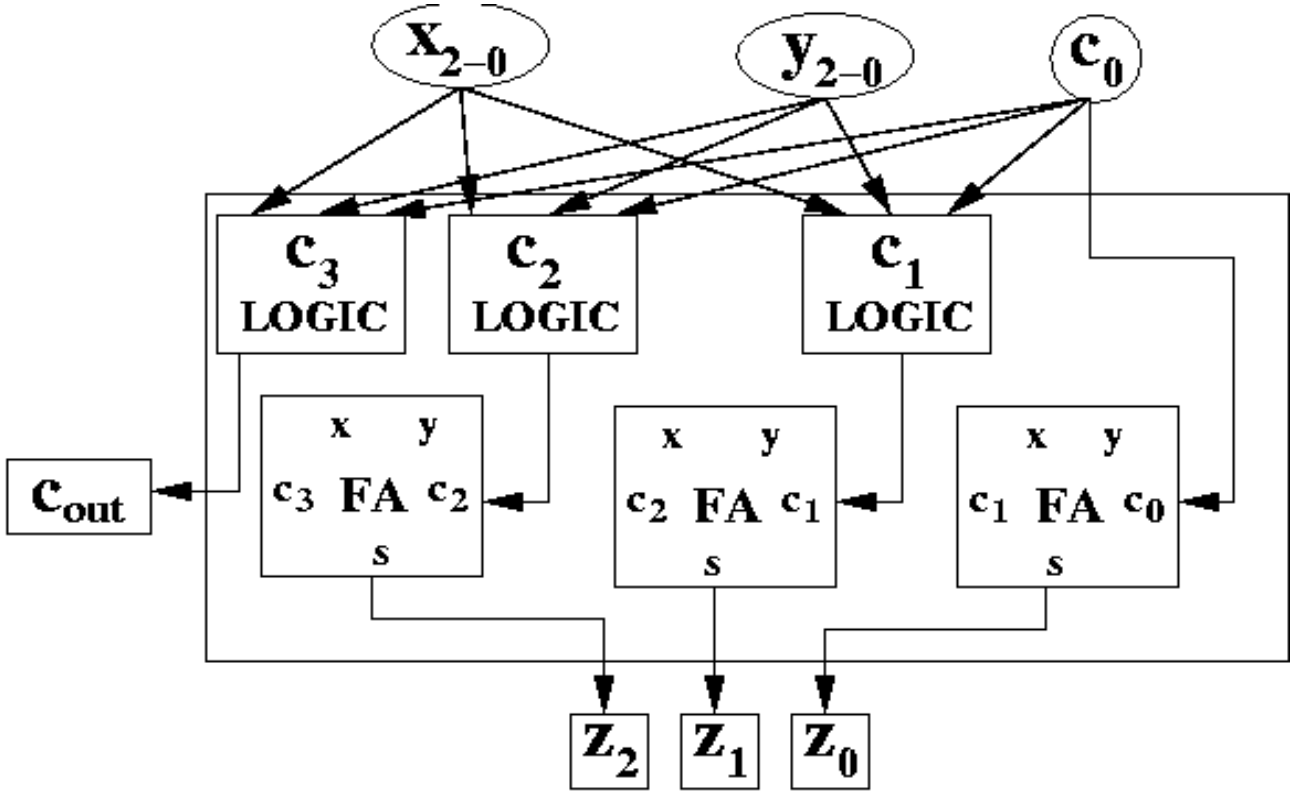
**Following the same pattern,**

$$c_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$$

**The circuit uses full adders, but the output of one is not directly connected to the input of the next**

## Carry lookahead

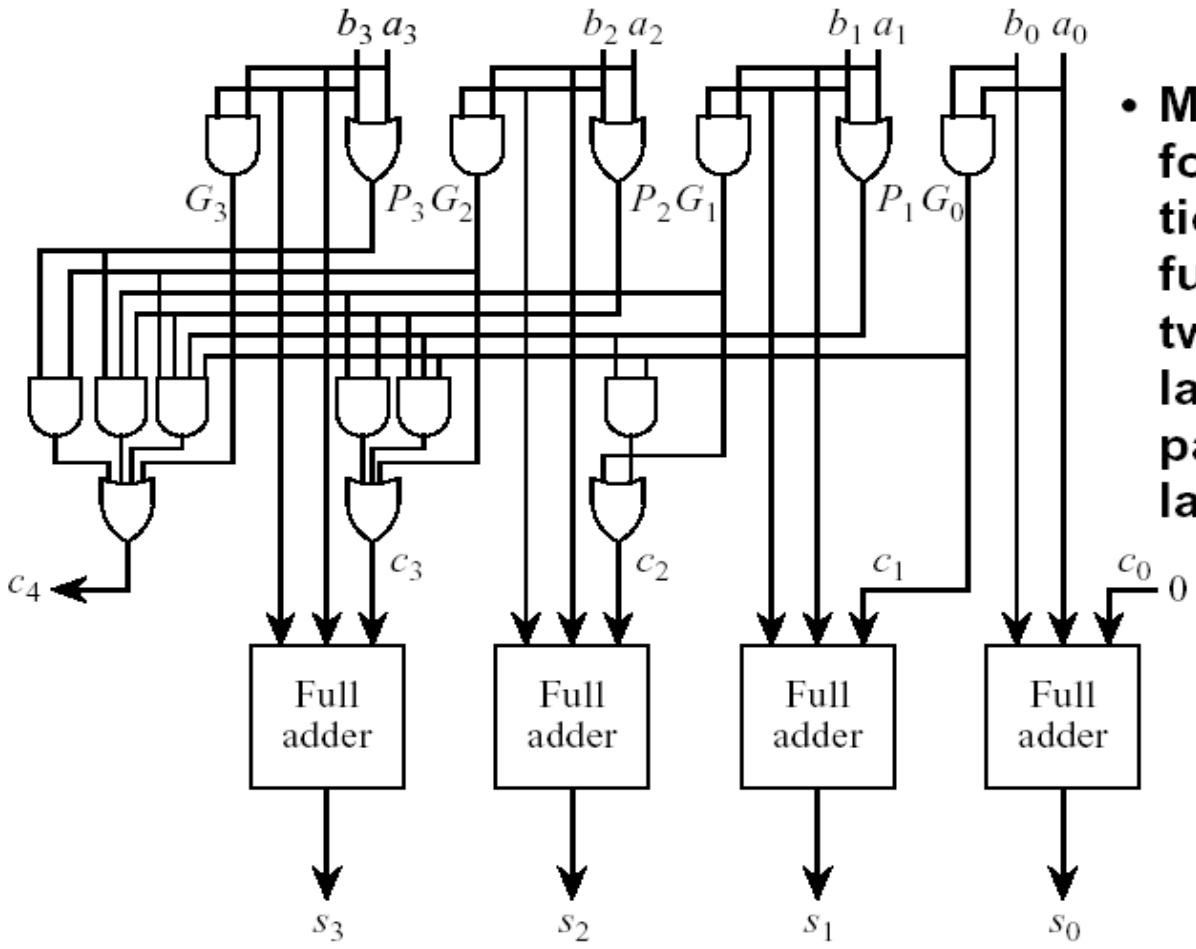
**Approximate circuit for 3 bits:**




**What's missing?**

**Need to show connections from $x_i$ and $y_i$ to the appropriate adders.**

## Carry lookahead: circuit




- **Maximum gate delay for the carry generation is only 3. The full adders introduce two more gate delays. Worst case path is 5 gate delays.**

## Carry lookahead: group

**How much have we reduced the delay?**

**We would like to have O(1) time, but**

**note that there are i OR operations for the ith carry ci,**

**and there are also i AND operations for the biggest term**

**There is a practical limit to the number of inputs to a single gate (fan-in).**

**We could build everything out of 2-input AND gates and OR gates, and the**

**delay would be only O(lg n) for n bits, which is still much better than O(n).**

**Another approach:**

**Build 4-bit carry-lookahead units, then cascade them together in group of**

**4 to get 16-bit adder.**

**This can be done with a maximum fan-in of only 4.**

**This is called group carry-lookahead (GCLA)**

**Need to deal with propagates and generates between 4-bit blocks.**

**"Super" propagate:**

**A propagate will occur from one group of 4 to the next**

**if every propagate in the first group is true.**

$P_0 = p_3 * p_2 * p_1 * p_0$

**Similarly:**

$P_1 = p_7 * p_6 * p_5 * p_4$

$P_2 = p_{11} * p_{10} * p_9 * p_8$

$P_3 = p_{15} * p_{14} * p_{13} * p_{12}$

**"Super" generate:**

**A generate will occur between 4-bit groups**

**if there is a carry out from the most signficant bit in the 4-bit group.**

**This occurs when:**

**Generate occurs for the most significant bit OR**

**Generate occurs for a lower bit and all intermediate propagates are true**

$G_0 = g_3 + (p_3 * g_2) + (p_3 * p_2 * g_1) + (p_3 * p_2 * p_1 * g_0)$

$G_1 = g_7 + (p_7 * g_6) + (p_7 * p_6 * g_5) + (p_7 * p_6 * p_5 * g_4)$

$G_2 = g_{11} + (p_{11} * g_{10}) + (p_{11} * p_{10} * g_9) + (p_{11} * p_{10} * p_9 * g_8)$

$G_3 = g_{15} + (p_{15} * g_{14}) + (p_{15} * p_{14} * g_{13}) + (p_{15}*p_{14}*p_{13}*g_{12})$

**Carry out:**

**Carry out for the 4-bit group is similar to the carry out for each bit:**

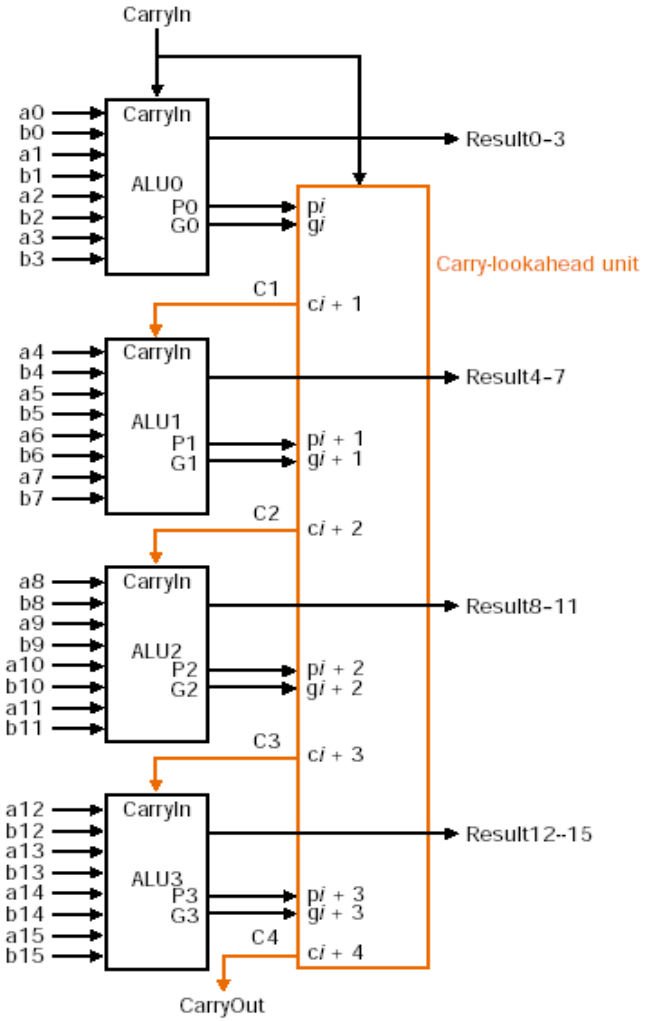
$C_1 = G_0 + P_0c_0$

$C_2 = G_1 + P_1G_0 + P_1P_0c_0$

$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0$

$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$

# Carry lookahead: group




**16-bit adder using carry-lookahead with 4-bit adders (Fig. 4.24)**

**Note that carry-in for each 4-bit adder is generated by carry-lookahead unit, not individual adders**

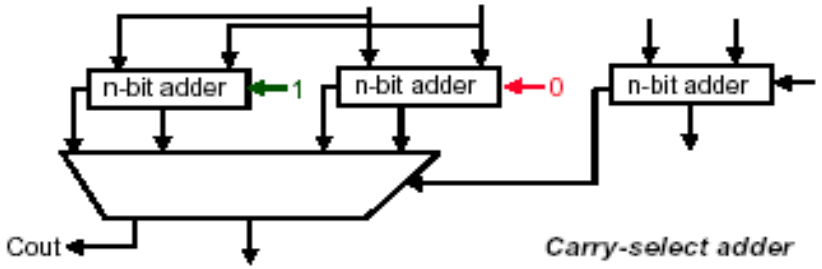
## Carry-select

**Another solution: carry-select adder**

- **Design trick: When all else fails, GUESS! (precompute)**

**To build 8-bit adder:**

- **Lower 4 bits: any adder (ripple-carry, carry-lookahead)**
- **Upper 4 bits: 2 adders**
  - **First adder has carry-in of 1**
  - **Second adder has carry-in of 0**
- **Select between 2 upper results based on carry-out from lower result**




*Carry-select adder*



**Reference:**

**http://www-inst.eecs.berkeley.edu/~cs152/**

**Time required:**

- **4-bit add time**
- **Multiplexor to select**

## Adder complexity

| | Time | Space |
|---|---|---|
| Ripple-carry (RCA) | O(n) | O(n) |
| Carry-lookahead (CLA) | O(log n) | O(n log n) |
| Carry-select (CSA) | O(sqrt n) | O(n) |