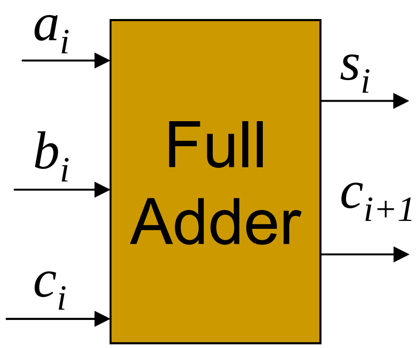
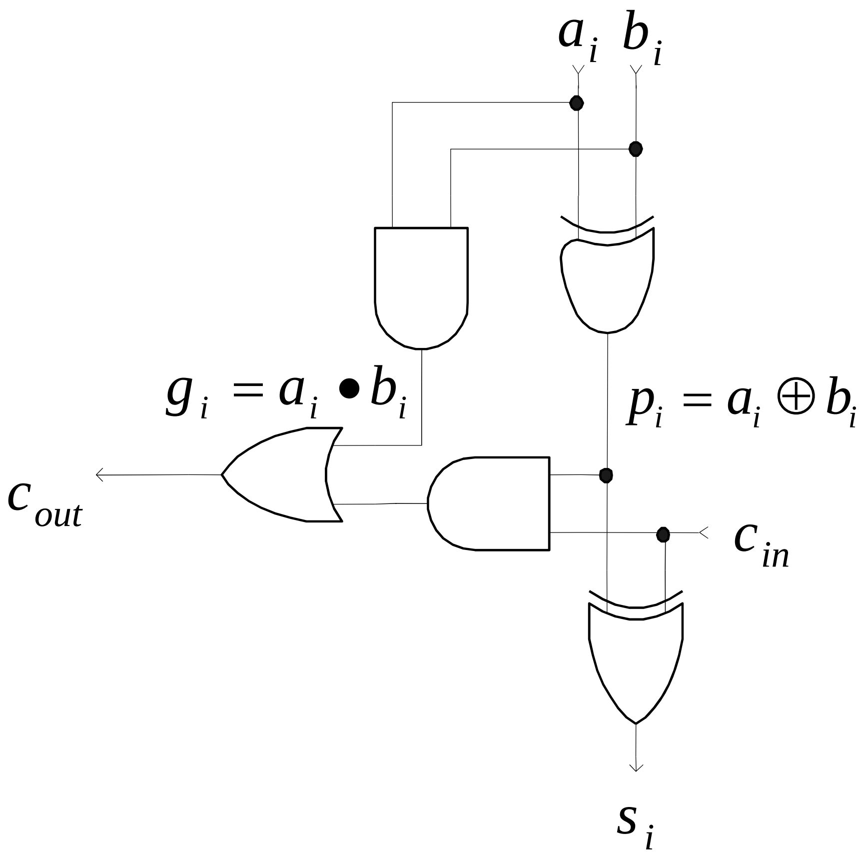
Functional Units for Addition and Subtraction

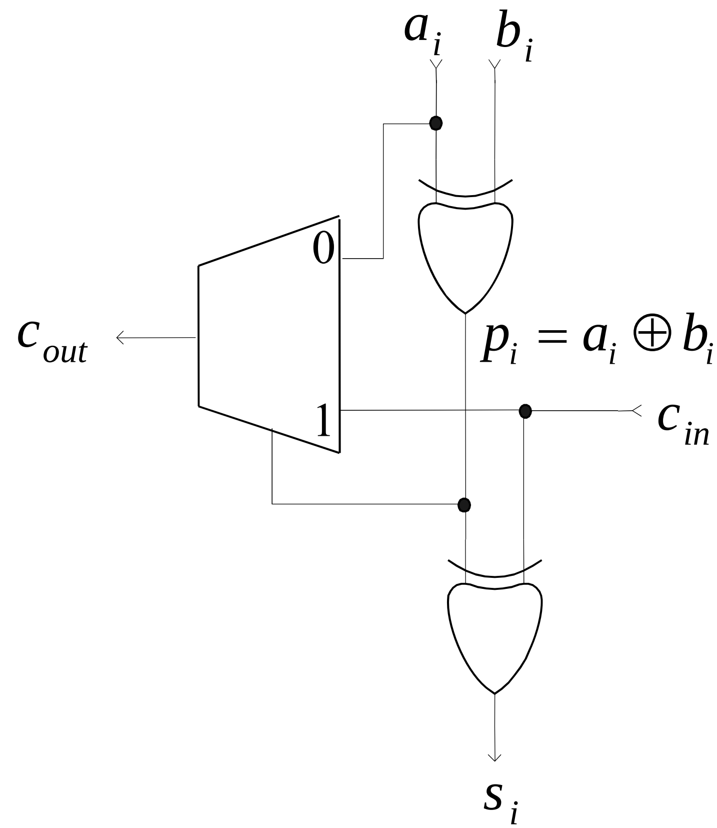


a_i	b_i	c_i	s_i	c_{i+1}	carry status
0	0	0	0	0	Delete
0	0	1	1	0	Delete
0	1	0	1	0	Propagate
0	1	1	0	1	Propagate
1	0	0	1	0	Propagate
1	0	1	0	1	Propagate
1	1	0	0	1	Generate
1	1	1	1	1	Generate

Full-Adder Implementation



Basic implementation



$$s_i = p_i \oplus c_i$$

more efficiently cell

(MUX is faster)

Express Sum and Carry as a Functions of *G & P*

* Generate: $g_i = a_i b_i$; Delete: $d_i = \overline{a}_i \overline{b}_i$
* Propagate: $p_i = a_i \oplus b_i$ (or sometime $a_i + b_i$)
* Then:

$$c_{i+1}(g_i, p_i) = \overline{a}_i b_i c_i + a_i \overline{b}_i c_i + a_i b_i = a_i b_i + a_i c_i + b_i c_i$$

$$= g_i + p_i c_i$$

$$s_i(p_i) = a_i \overline{b}_i \overline{c}_i + \overline{a}_i b_i \overline{c}_i + \overline{a}_i \overline{b}_i c_i + a_i b_i c_i$$

$$= a_i \oplus b_i \oplus c_i = p_i \oplus c_i$$

- Ignore sign-bit-carry for 2's complement system
- End-around carry for 1's complement system (MSB carry is added to the LSB of sum)
- Both systems have the same overflow mechanism

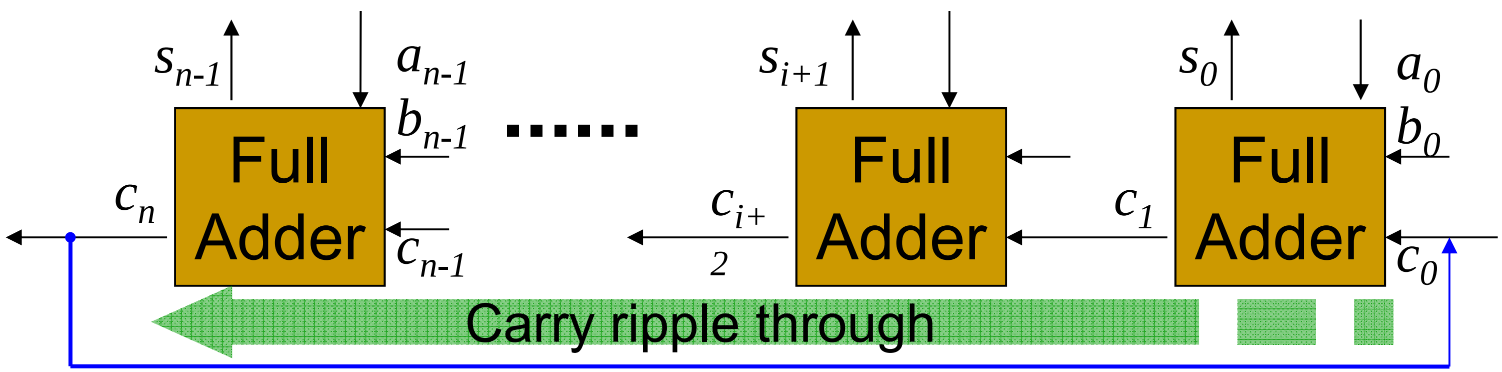
Ripple-Carry Adder (RCA)

- Recursive design - Worst case delay linear with the number of bits (for no sign or 2's complement system)

$$t_d = O(n)$$

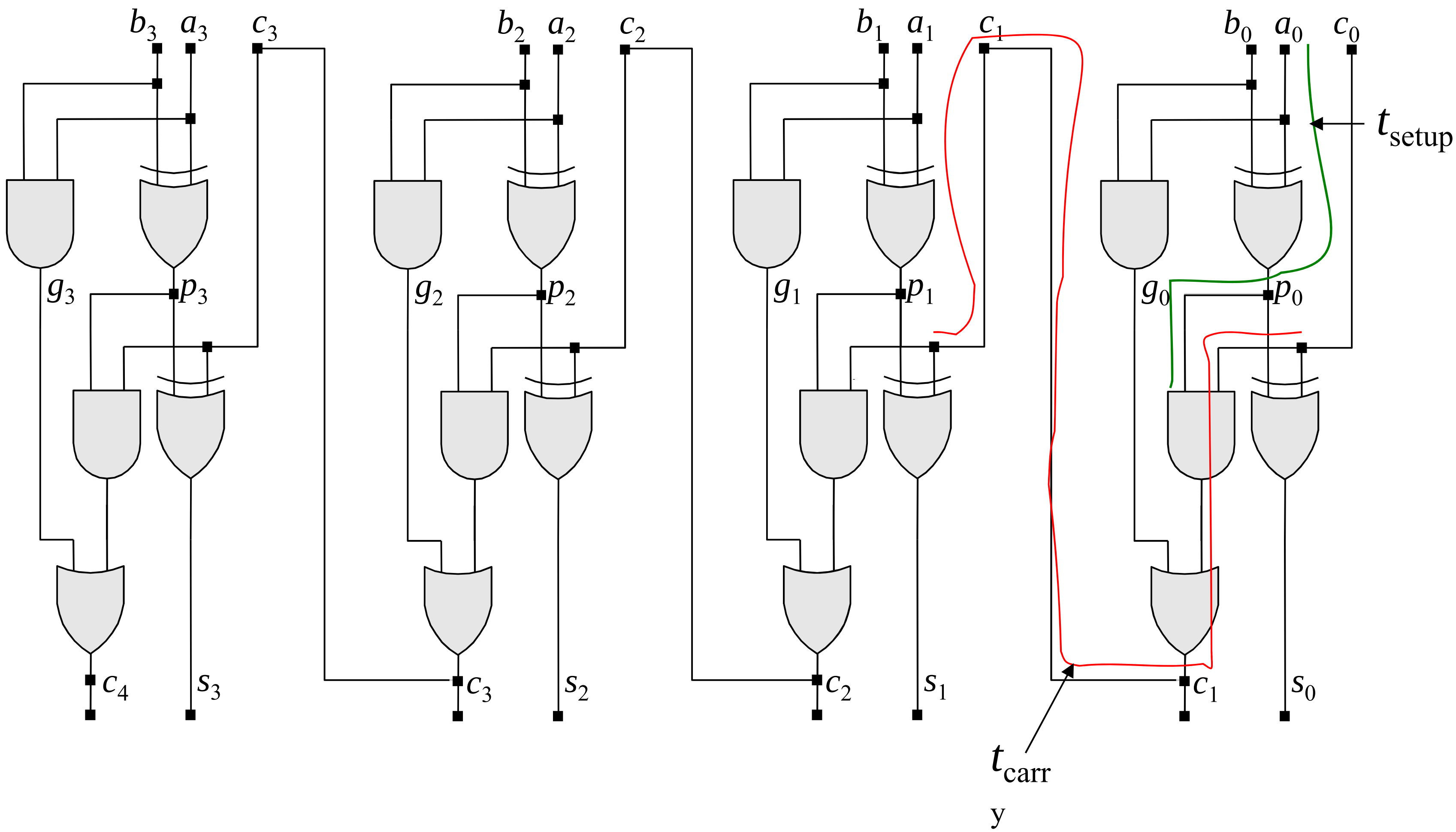
$$t_{add} \approx t_{setup} + (n-1)t_{carry} + t_{last\_sum}$$

- Goal: Make the fastest possible carry path circuit



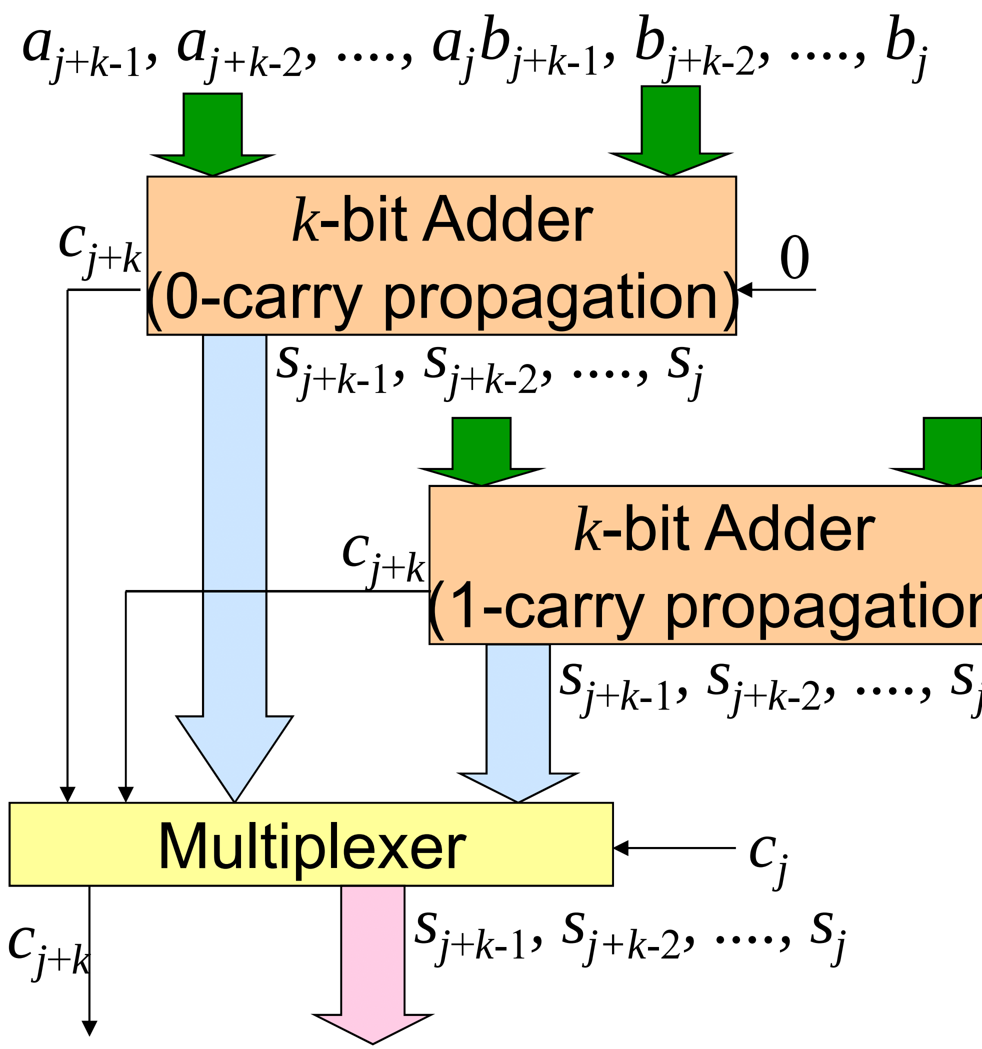
End-around carry for 1's complement or
Ignore sign-bit carry for 2's complement

Gate-level schematic of a basic 4-bit adder

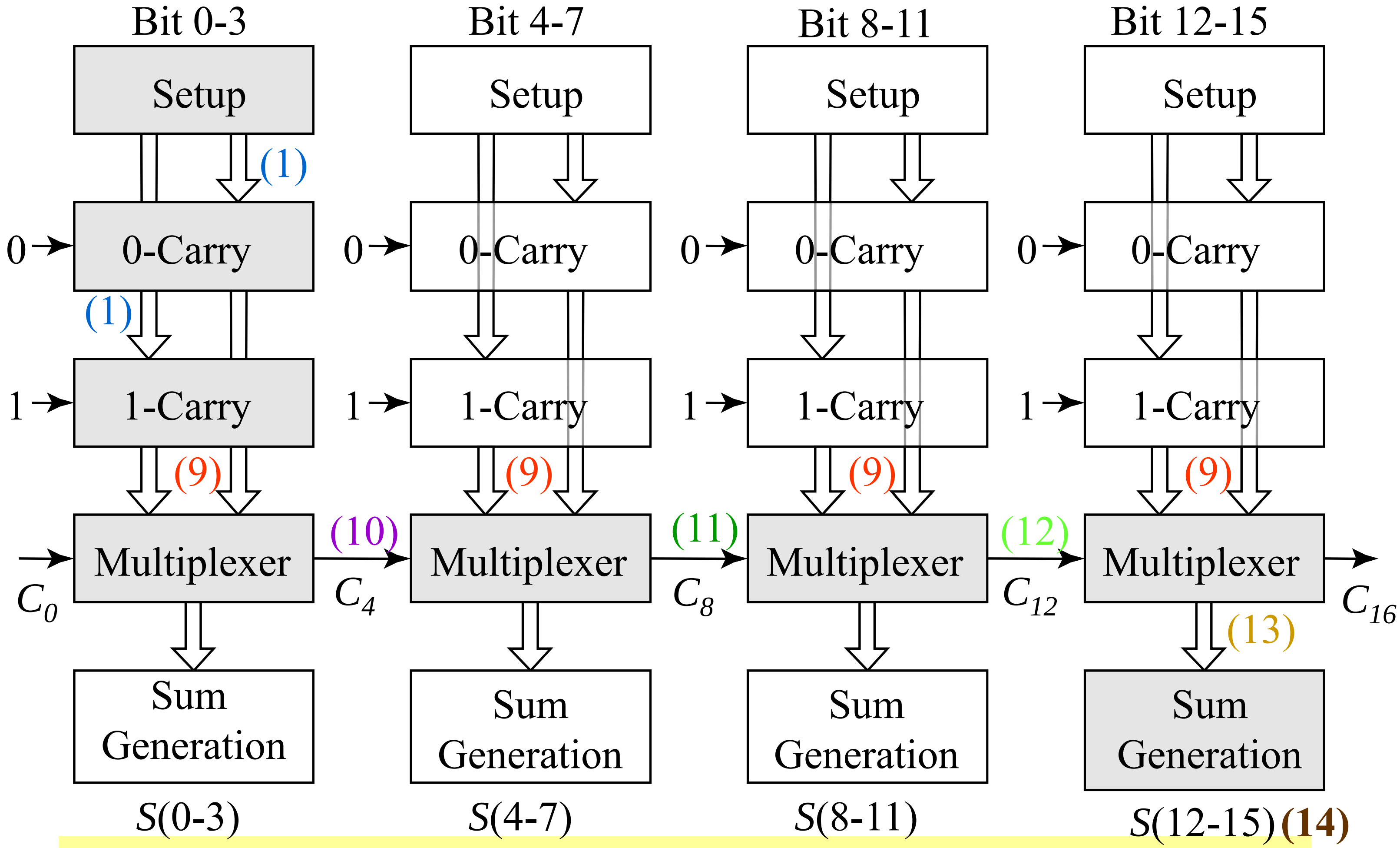


Carry-Selected Adder (CSeA)

- *n* bits divided into non-overlapping groups (even better for possibly different lengths)
- Each group generates two sets of sum and carry:
  - one assumes incoming carry into group is 0
  - one assumes incoming carry into group is 1
- Two sets of outputs can be calculated in a ripple-carry manner

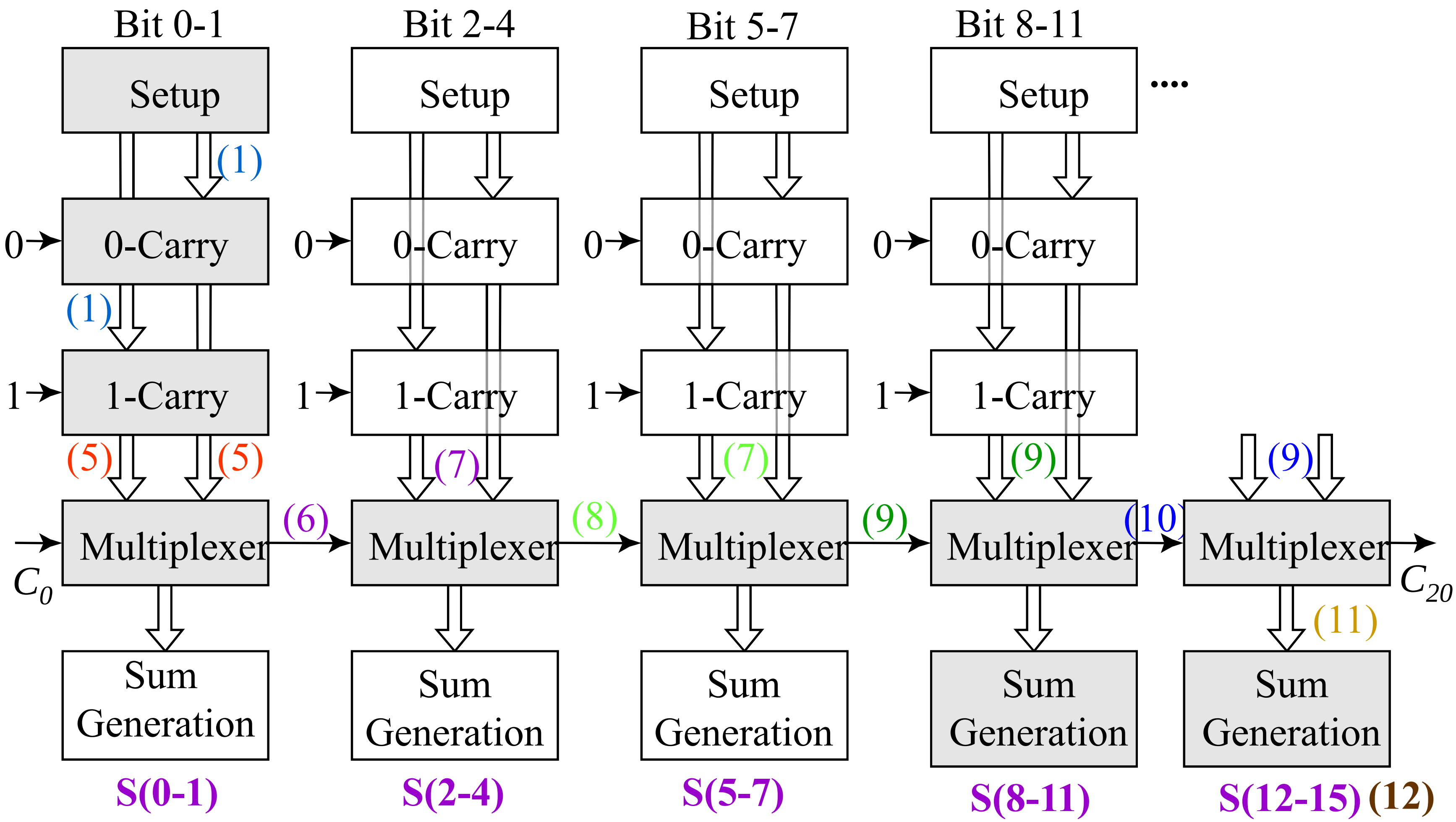


Carry-Select with 4-bit groups (linear)



$$t_{add} \approx t_{setup} + 4t_{carry} + \left(\frac{n}{4}\right) t_{mux} + t_{extra\_for\_last\_sum\_selection}$$

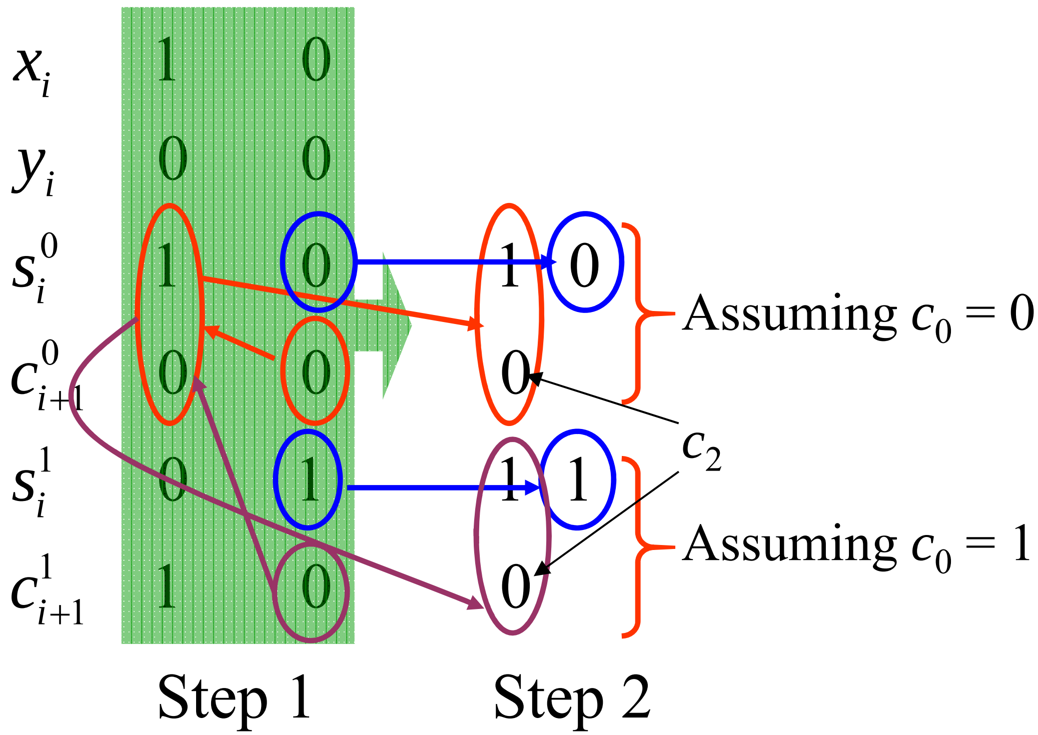
16-bit square-root carry-select



$$t_{add} \approx t_{setup} + 2t_{carry} + \sqrt{2n}t_{mux} + t_{extra\_for\_last\_sum\_selection}$$

Conditional Sum Adder (CSuA)

Example: Combining two 1-bit into one 2-bit operands



Step 1: Each bit constitutes a separate group

Step 2: Two bit positions combined into one group of size 2.

- Carry-out from lower position becomes internal (to group) carry and appropriate set of outputs for higher position selected

Example: Combining 1-bit into 8-bit operands – $c_0 = 0$

x_i		1	0	1	1	0	1	1	0
y_i		0	0	1	0	1	1	0	1
s_i^0	Step 1	1	0	0	1	1	0	1	1
c_{i+1}^0		0	0	1	0	0	1	0	0
s_i^1		0	1	1	0	0	1	0	0
c_{i+1}^1		1	0	1	1	1	1	1	1
s_i^0	Step 2	1	0	0	1	0	0	1	1
c_{i+1}^0		0		1		1		0	
s_i^1		1	1	1	0	0	1		
c_{i+1}^1		0		1		1			
s_i^0	Step 3	1	1	0	1	0	0	1	1
c_{i+1}^0		0				1			
s_i^1		1	1	1	0				
c_{i+1}^1		0							
Result		1	1	1	0	0	0	1	1