Adder

Lecture 18

- Addition:

```
  111101   augend
+  10111   addend
```

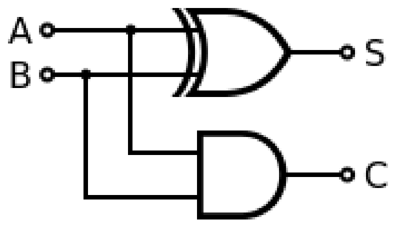
```
 111111     carries
  111101   augend
+  10111   addend
1010100   sum
```

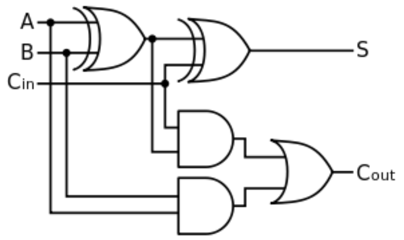
```
 111111     carries
  111101   augend
+  10111   addend
1010100   sum
```

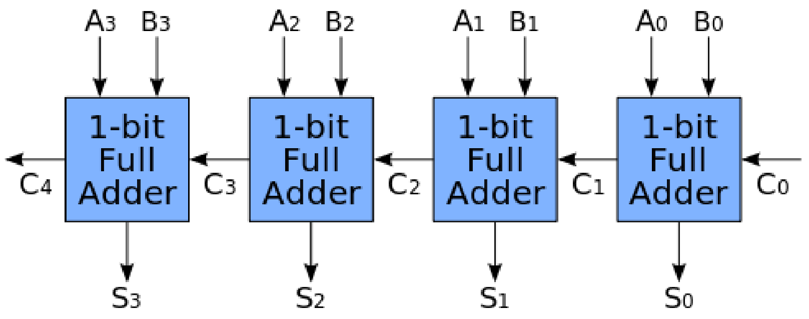
- both the sum bit and carry bit are 1's

$1+1+1=(1+1)+1=(10)_2+(01)_2=11$

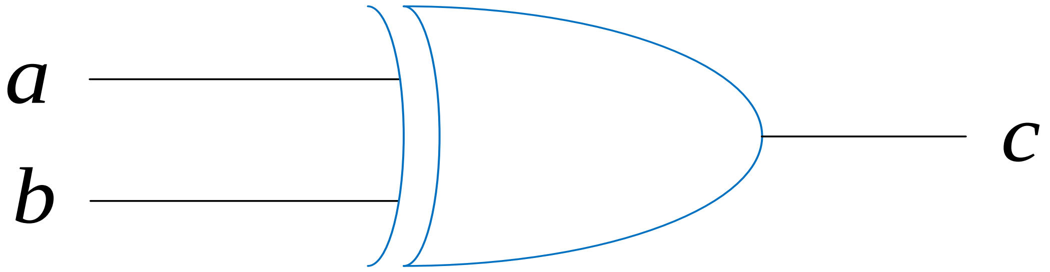
- Adders
- Adder or summer is a digital logic device used to carry binary addition of numbers.
- Typical uses involve calculating address in memory, table indices, incrementing or decrementing operators, etc.
- Two kinds of adder: Half Adder & Full Adder
- An n-th Bit Adder can be designed by cascading an arbitrary amount of Full Adders







- Quarter adder: Exclusive OR: No Carry



a	b	c
Low	Low	Low
Low	High	High
High	Low	High
High	High	Low

a	b	c
0+	0	0
0+	1	1
1+	0	1
1+	1	0

No Carry

$$f_{XOR}(a,b) = a \oplus b = a\bar{b} + \bar{a}b = \bar{a}a + \bar{a}b + \bar{b}a + \bar{b}a$$

$$= \bar{a}(a+b) + \bar{b}(a+b) = (\bar{a}+\bar{b})(a+b)$$

XOR

- Exclusive OR

$$a \oplus a = 0$$

$$a \oplus \overline{a} = 1$$

$$a \oplus 0 = a$$

$$a \oplus 1 = \overline{a}$$

$$\overline{a} \oplus \overline{b} = a \oplus b$$

$$a \oplus b = b \oplus a$$

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c$$

- Exclusive OR IEEE Standard

a	b	sum(a,b)	sum(a,b)=1?	f(a,b)= $a \oplus b$
0	0	0	false	0
0	1	1	true	1
1	0	1	true	1
1	1	2	false	0

Output is modulo-2 sum of input

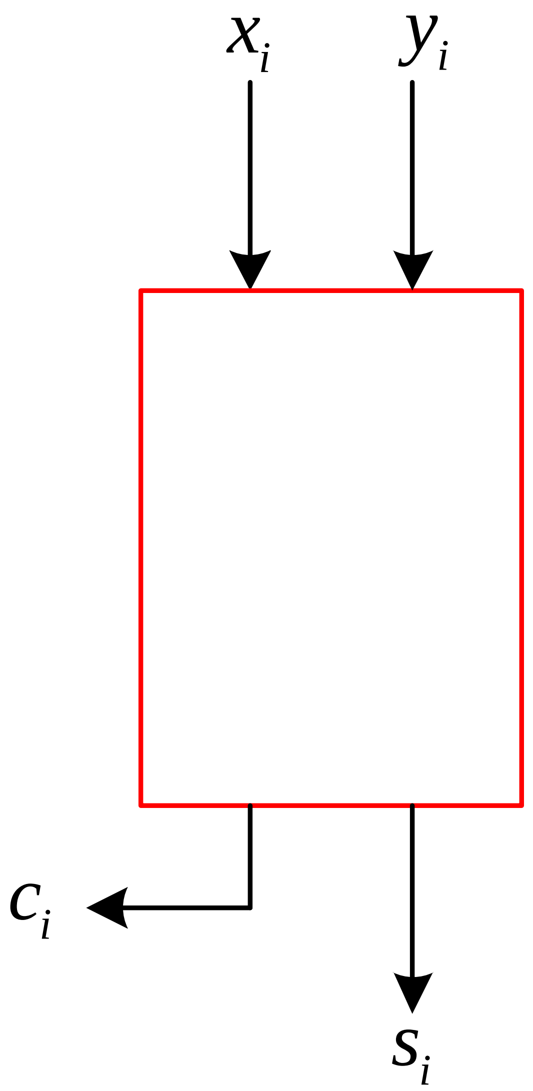
- Half-Adder

$$s_i = x_i \oplus y_i$$

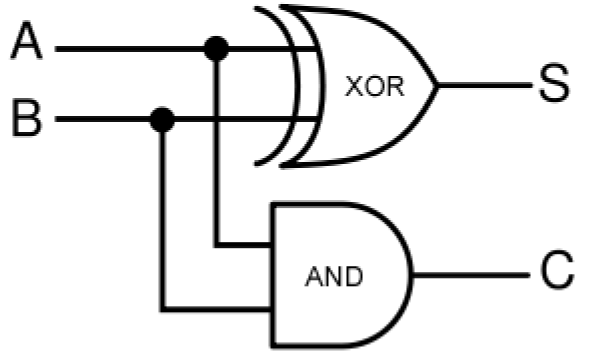
$$c_i = x_i y_i$$

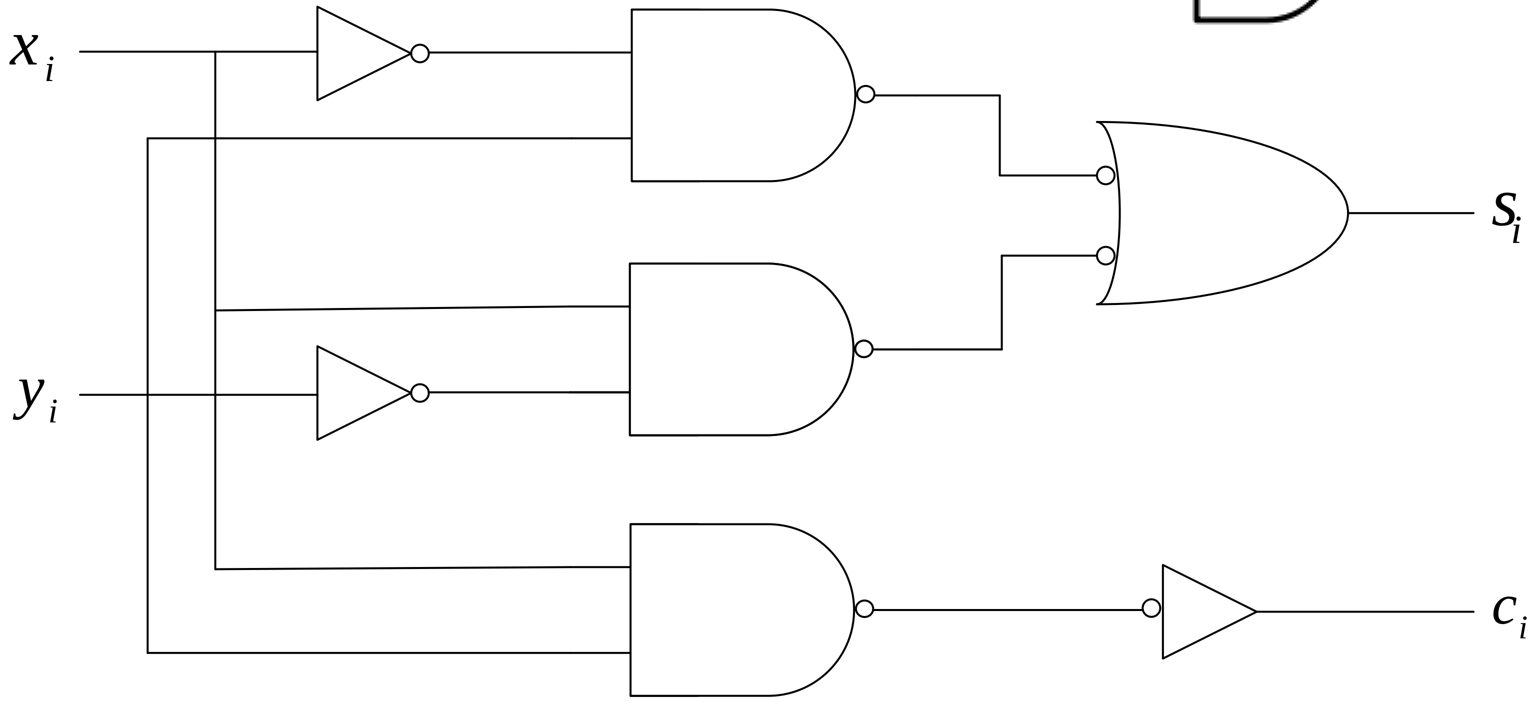
- Truth table

x_i	y_i	c_i	s_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

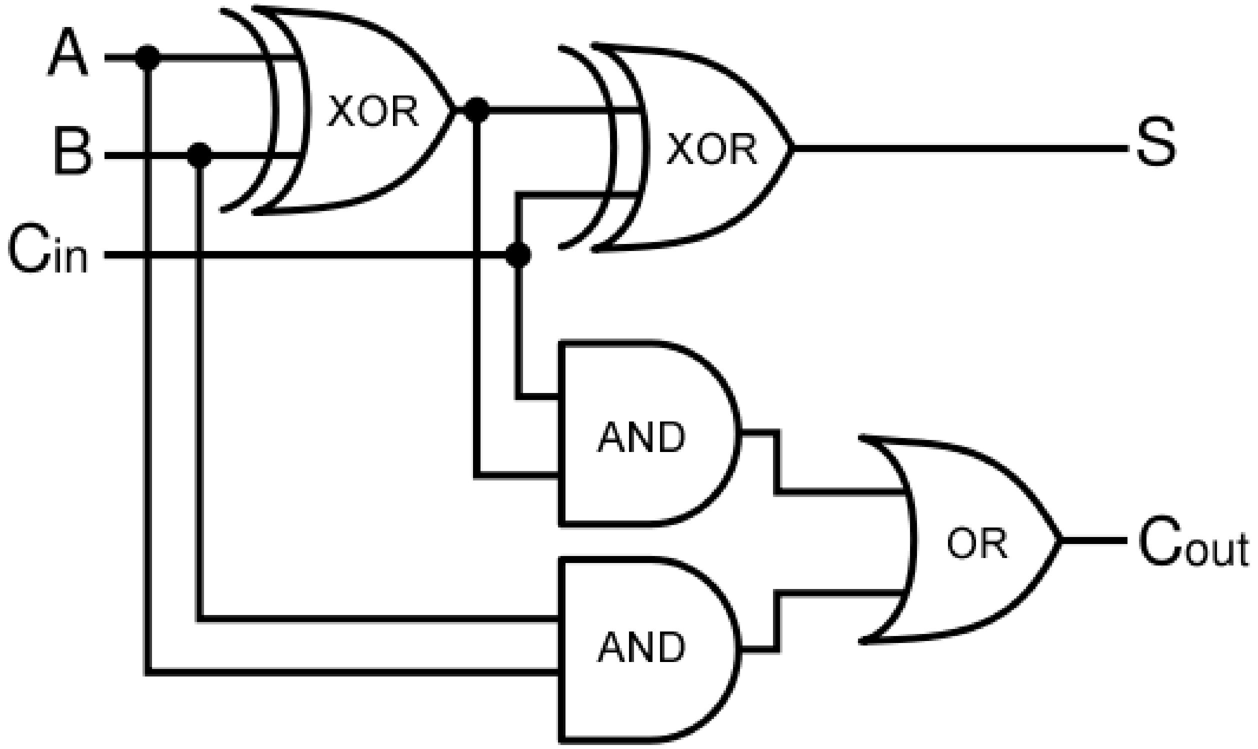


- Half-Adder



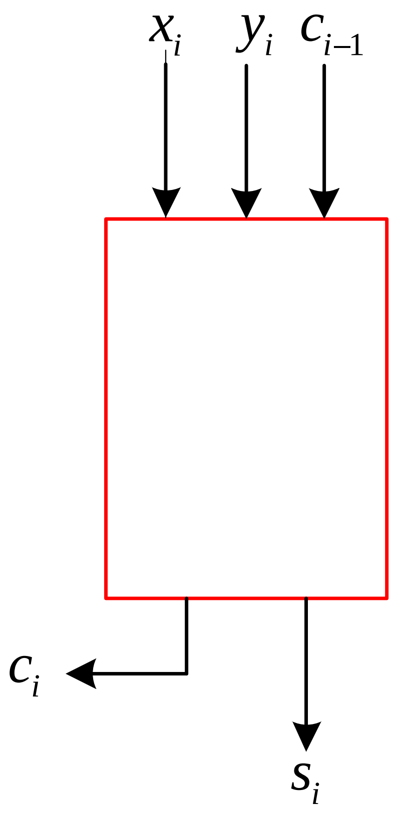


- Full-Adder



- One-bit Full-Adder
- Binary addition adding 2 bits + one carry bit
- Full adder 2-output combinational logic network that add 3 binary bits

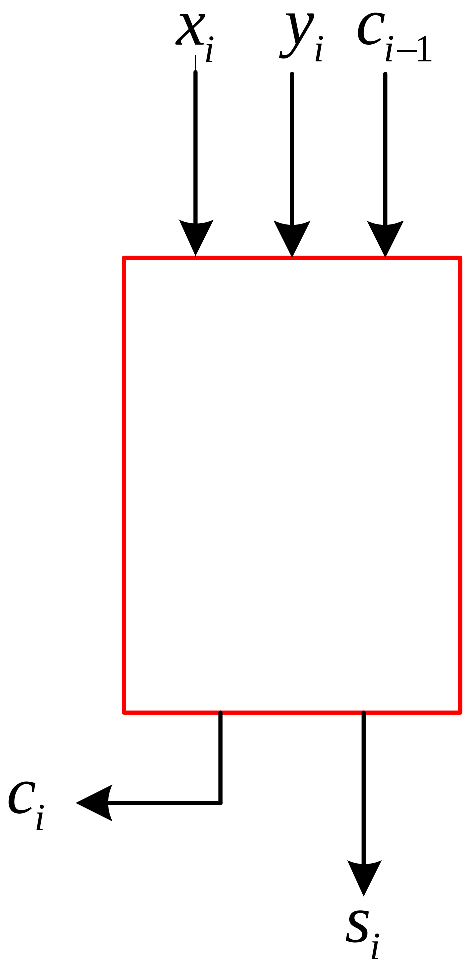
x_i	y_i	c_{i-1}	c_i	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$s_i = x_i \oplus y_i \oplus c_{i-1}$$

$$c_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$$

Full-Adder



$$s_i = x_i \oplus y_i \oplus c_{i-1}$$

$$s_i = x_i \oplus y_i \oplus c_{i-1}$$

$$= x_i \overline{y_i}\,\overline{c_{i-1}} + \overline{x_i}\,\overline{y_i} c_{i-1} + \overline{x_i} y_i \overline{c_{i-1}} + x_i y_i c_{i-1}$$

$$c_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$$

- Steps to Designing the Full Adder
- 1 Derive the Truth Table
- 2 Devise the K-Map
- 3 Obtain the Logic Expression
- Draw out the Logic Circuit
- Building Blocks
- 1 NAND Gate
- 2 NOR Gate
- 3 NOT Gate

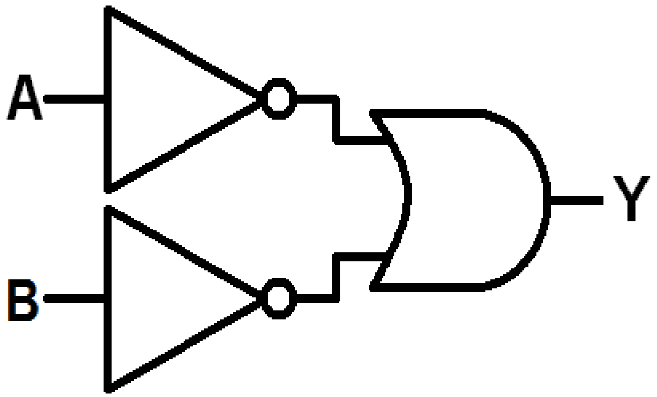
- Building Blocks
- 1 NAND Gate

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

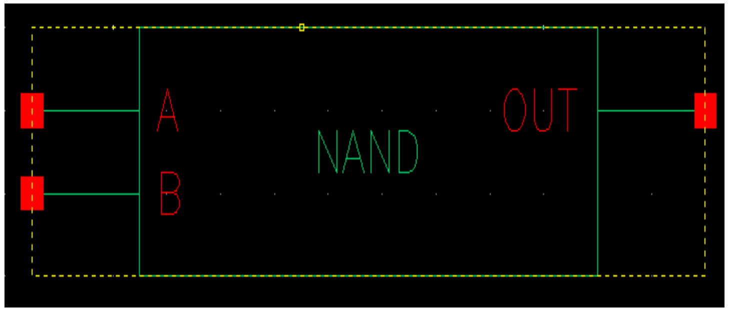
NAND Truth Table

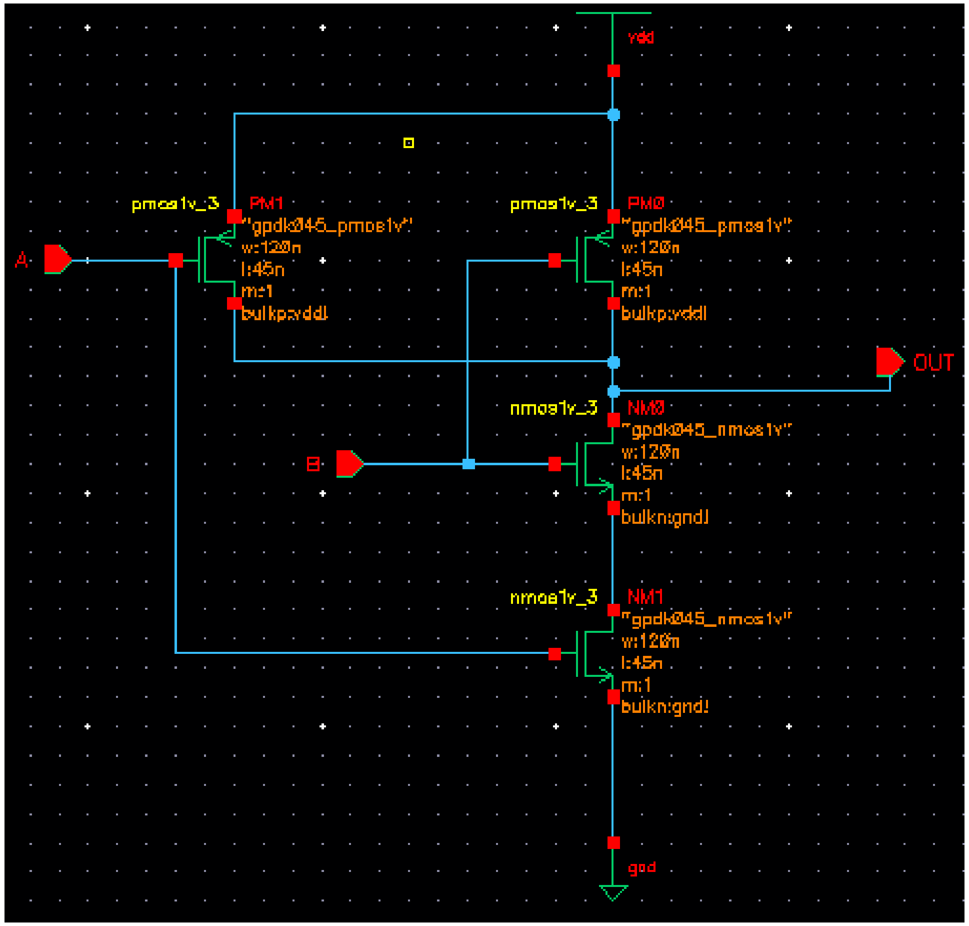
B \ A	0	1
0	1	1
1	1	

NAND Logic Expression: $Y = \overline{A} + \overline{B}$



- Building Blocks
- 1 NAND Gate





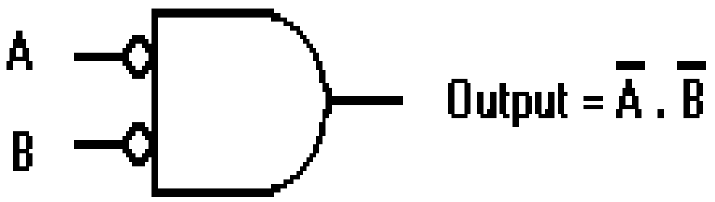
- Building Blocks
- ❷ NOR Gate

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOR Truth Table

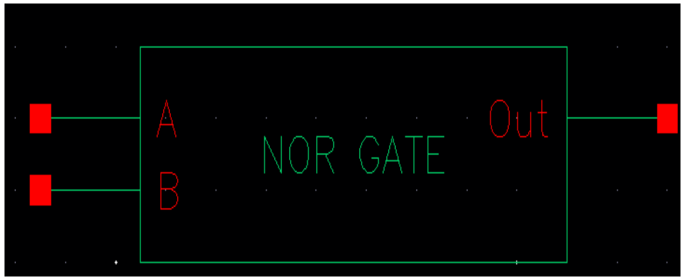
B \ A	0	1
0	1	
1		

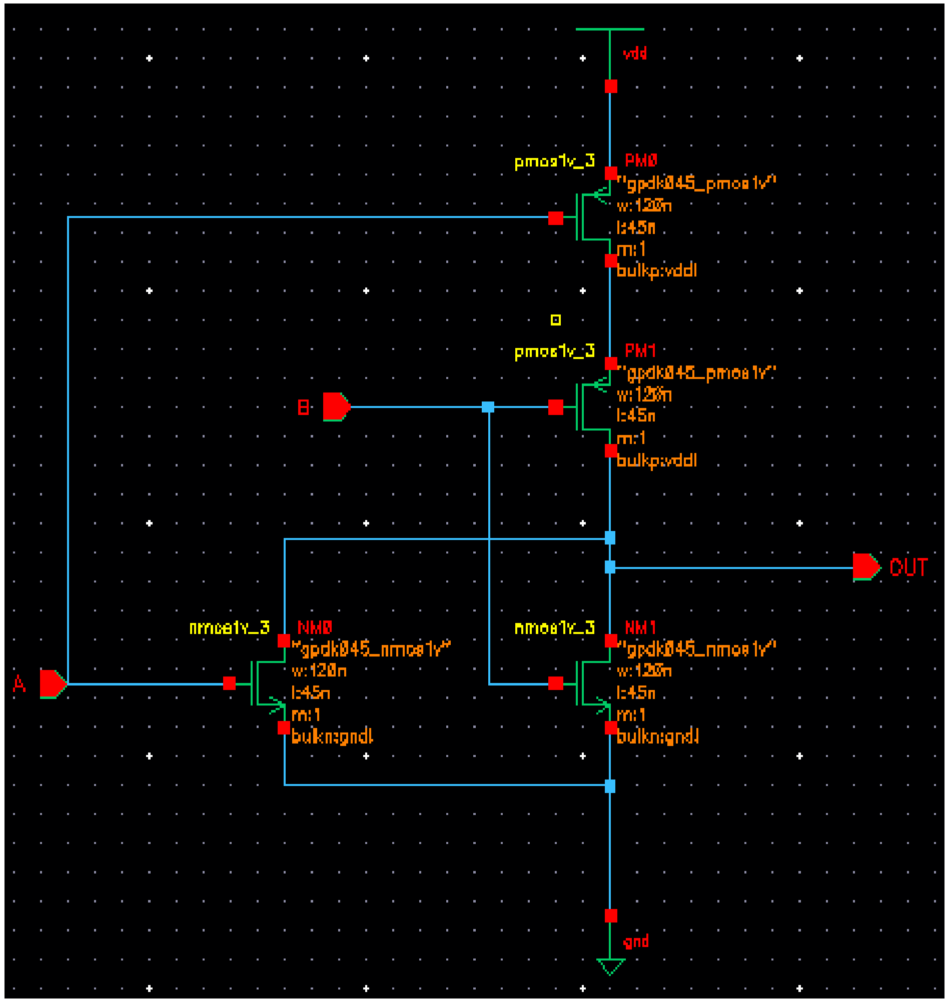
NOR Logic Expression: $Y = \overline{A}\,\overline{B}$



NOR Logic Circuit

- Building Blocks
- 2 NOR Gate





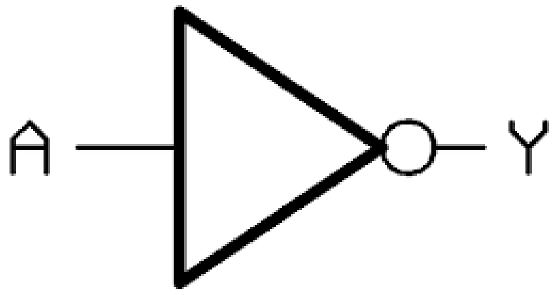
- Building Blocks
- ❸ NOT Gate

A	Y
0	1
1	0

NOT Truth Table

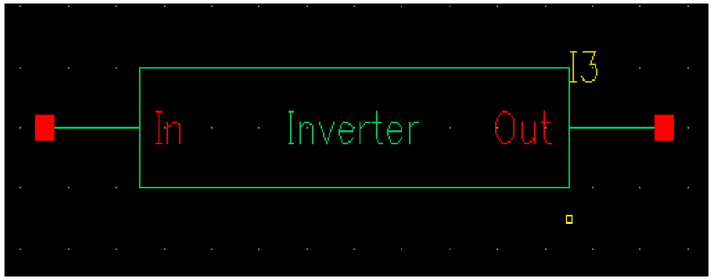
K-MAP Not Needed For NOT Gate

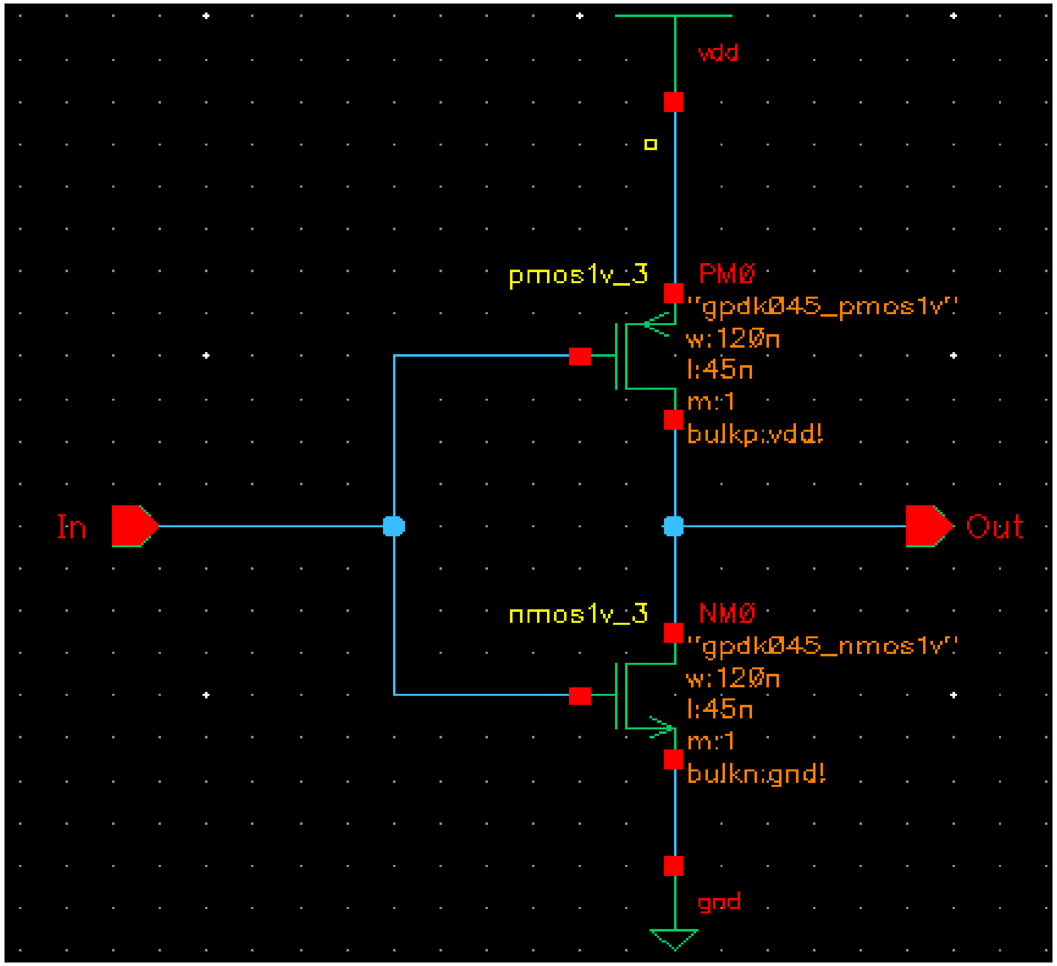
NOT Logic Expression:



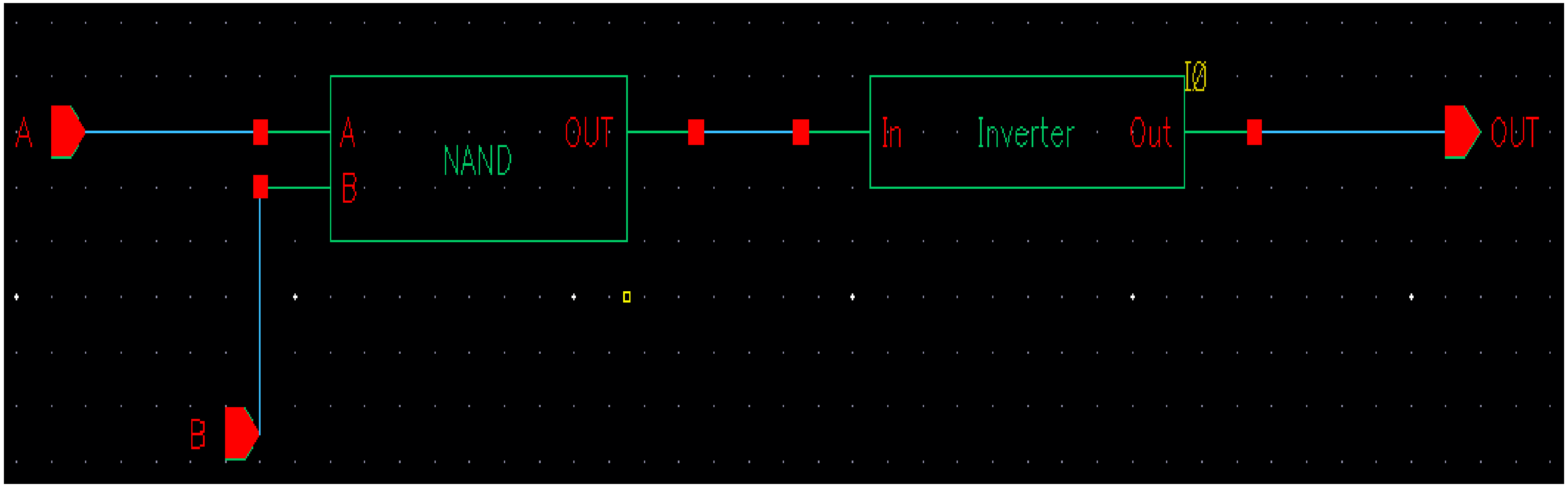
NOT Logic Circuit

- Building Blocks
- ❸ NOT Gate





- Blocks
- ❶ AND Gate



Adder

- Blocks
- ❷ XOR Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

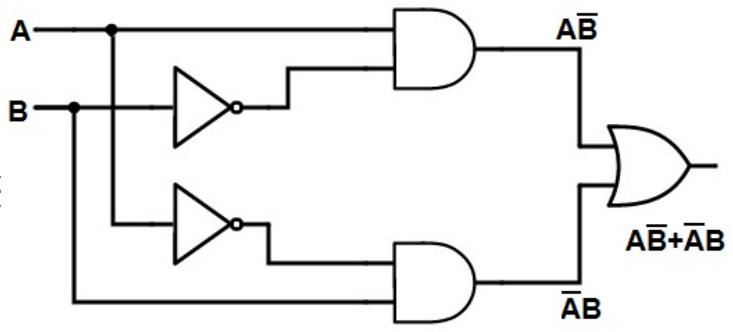
XOR Truth Table

B \ A	0	1
0		1
1	1	

XOR K-Map

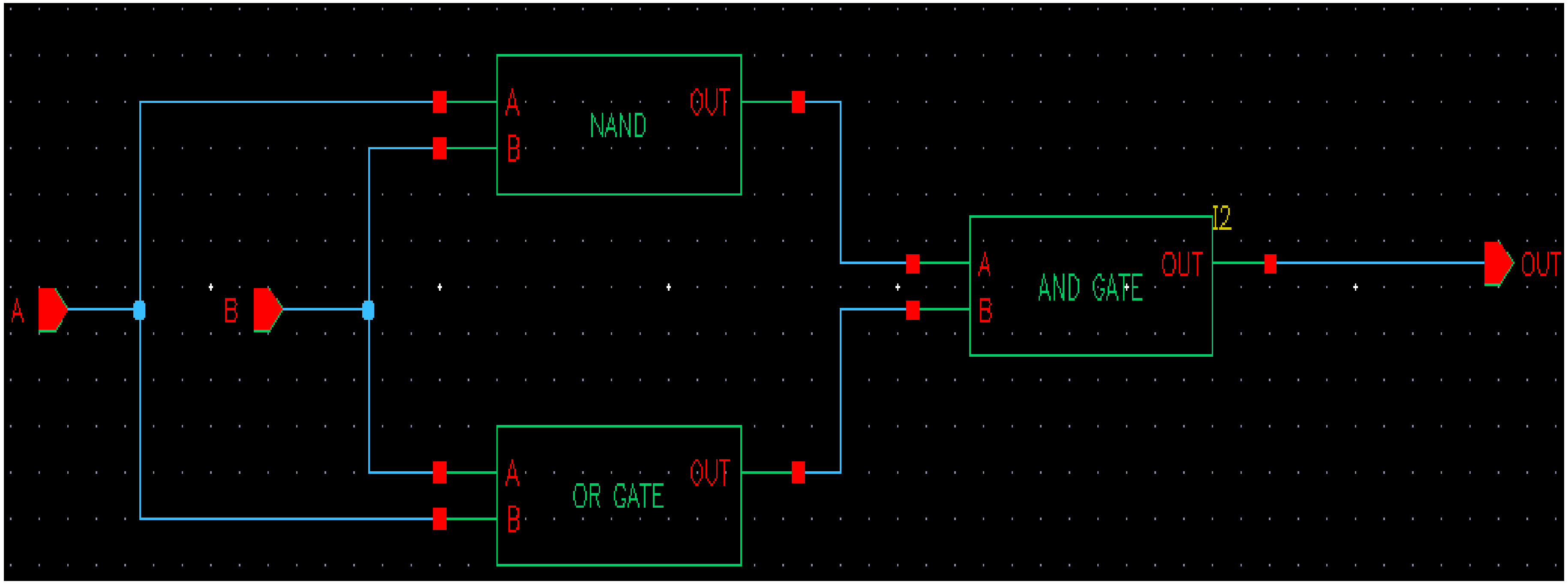
XOR

Logic Expression: Y = A’B + AB’ or Y = AE



XOR Logic Circuit

- Building Blocks
- 1 XOR Gate



- Blocks
- 3 Full Adder

A	B	Cin	S	Cout
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

Full Adder Truth Table

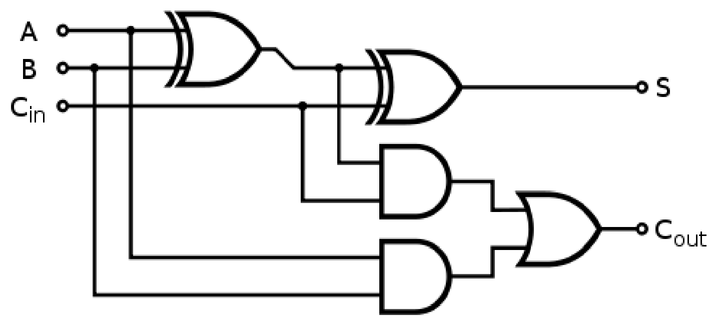
AB / Cin	00	01	11	10
0		1		1
1	1		1	

Sum k-map

AB / Cin	00	01	11	10
0			1	
1		1	1	1

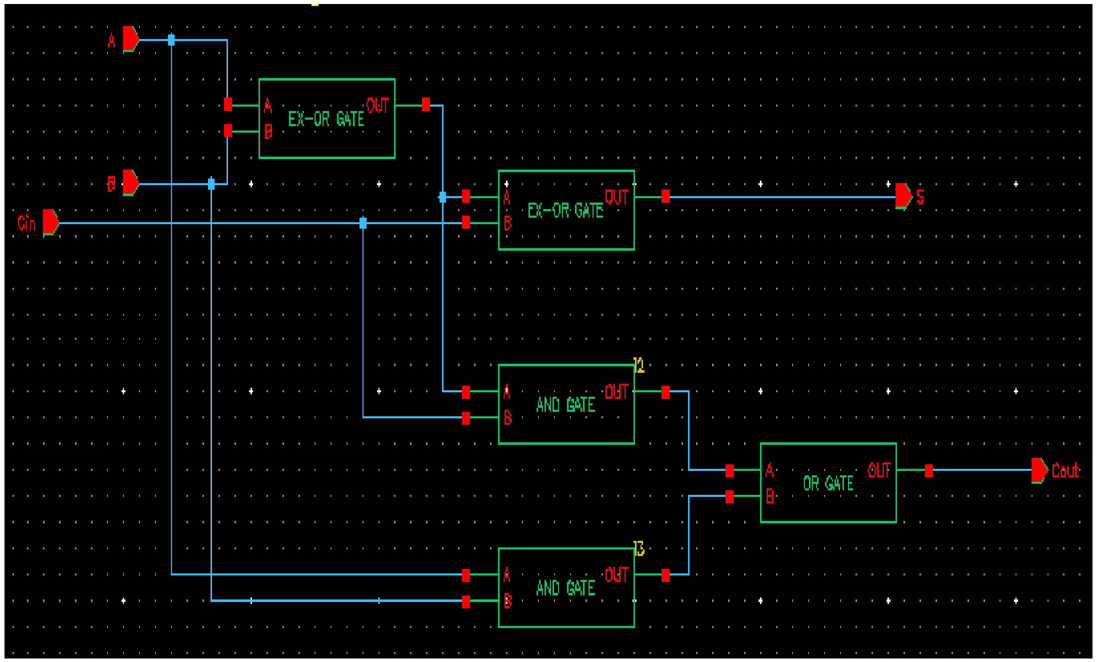
Carry K-map

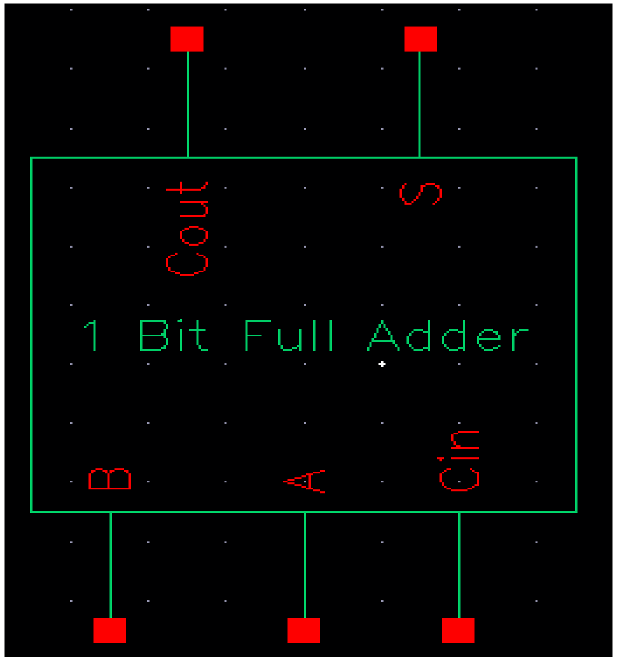
$$Sum = A'B'Cin + A'BCin' + ABCin + AB'CinS$$
$$= A \oplus \mathbf{B} \oplus \mathbf{C} \quad (1)$$
$$\mathbf{Cout}: Cout = AB + ACin + BCinCout =$$
$$([A \oplus \mathbf{B}]\mathbf{Cin}) + (AB) \quad (2)$$



Full Adder Logic Circuit

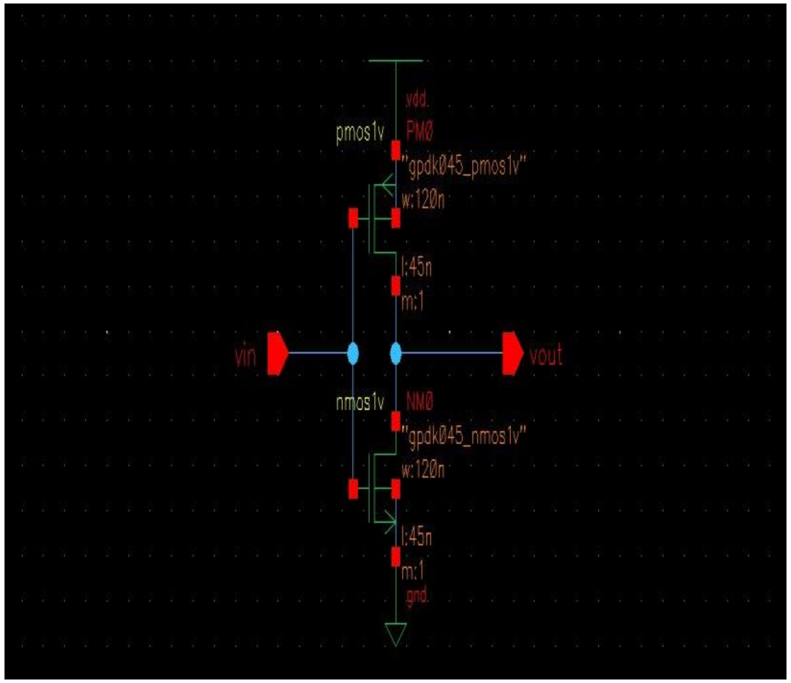
- Building Blocks
- ❸ Full Adder



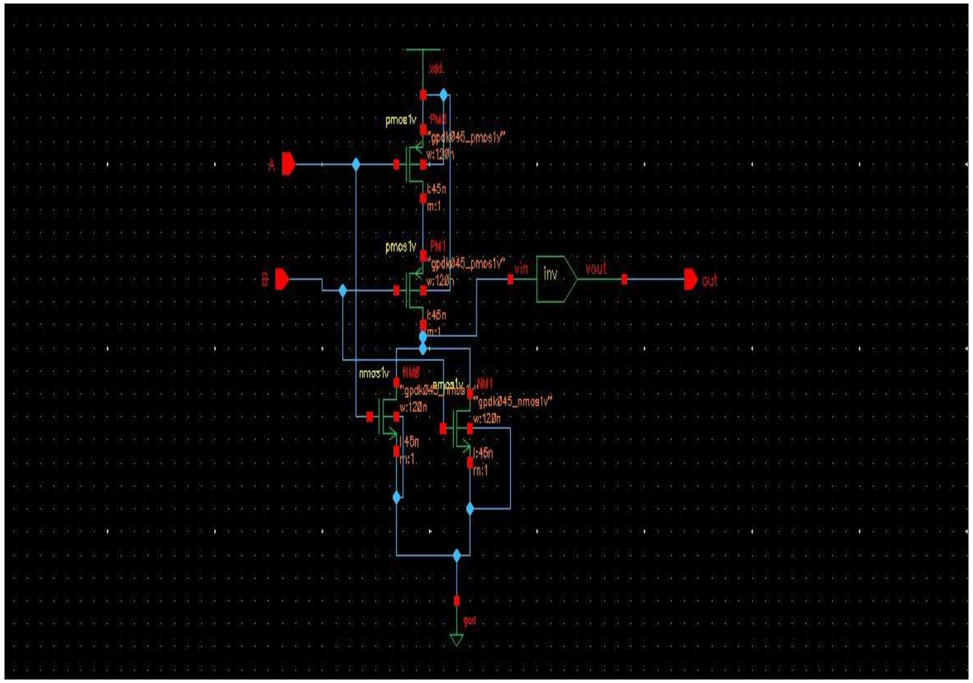


- Brute Force Implementation of Full Adder
- Inverter, OR, AND, XOR,

Inverter Schematic

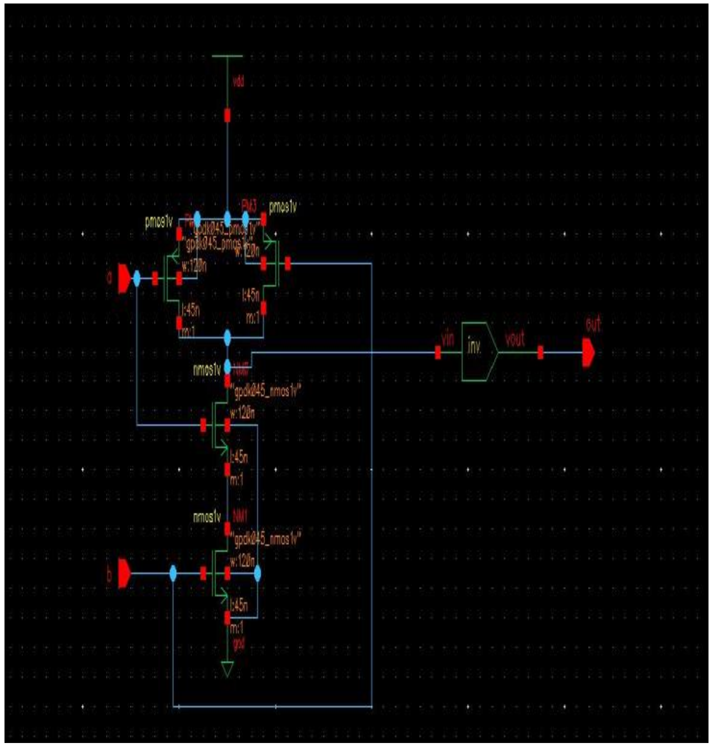


OR Schematic

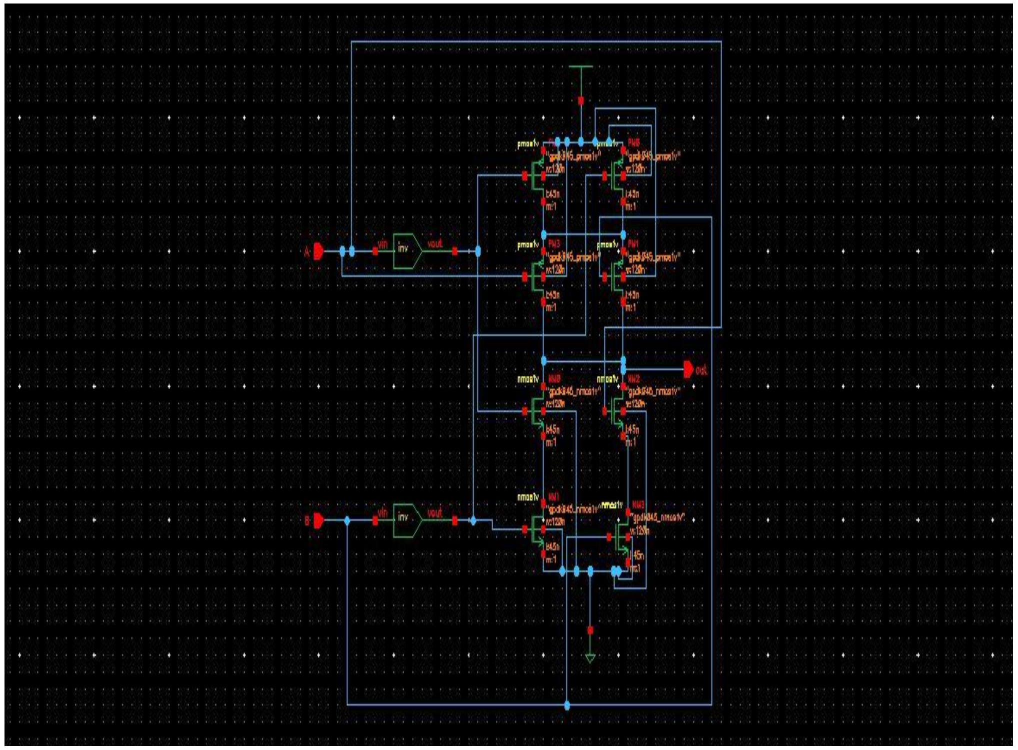


- Brute Force Implementation of Full Adder
- Inverter, OR, AND, XOR,

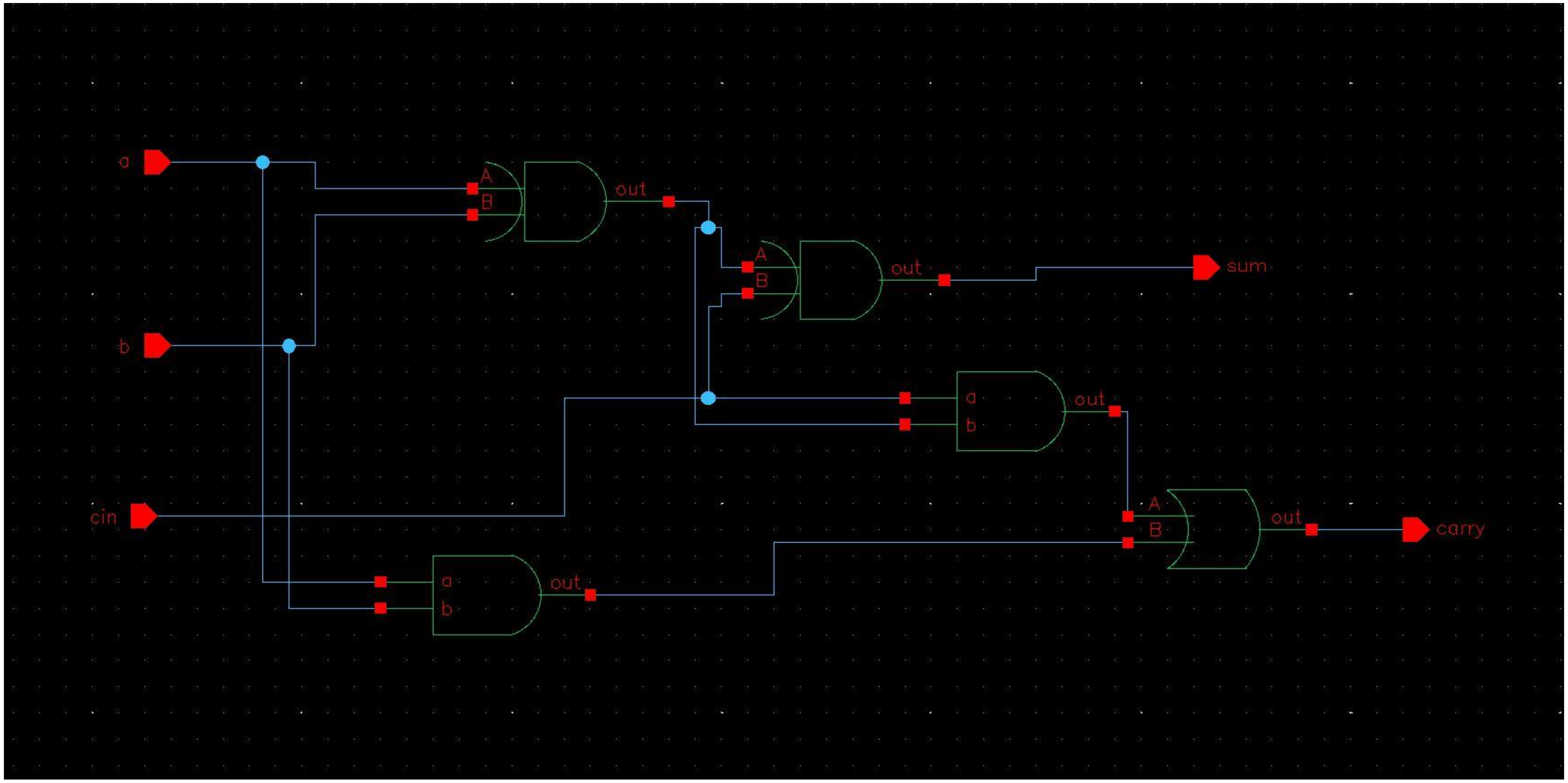
AND Schematic



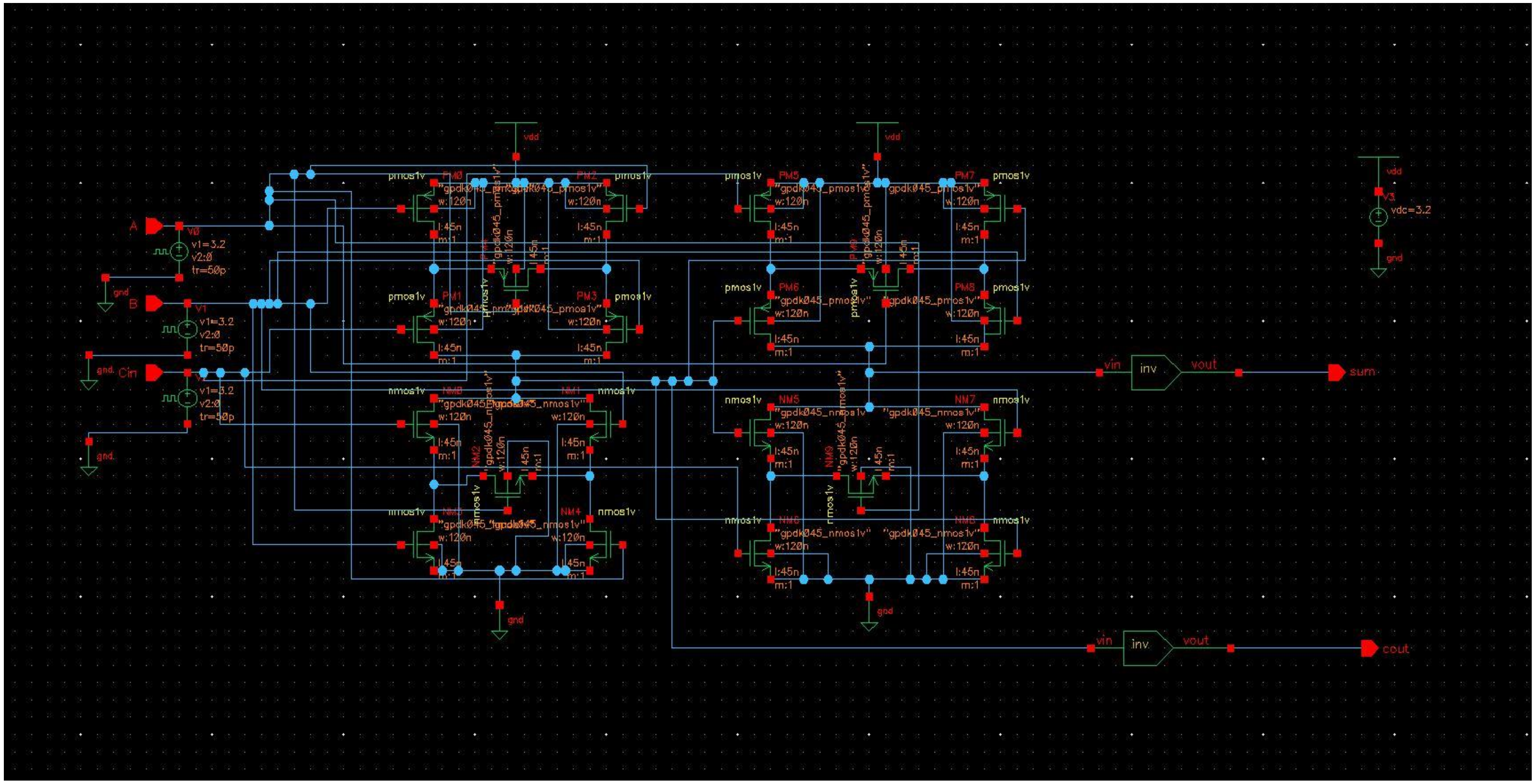
XOR Schematic



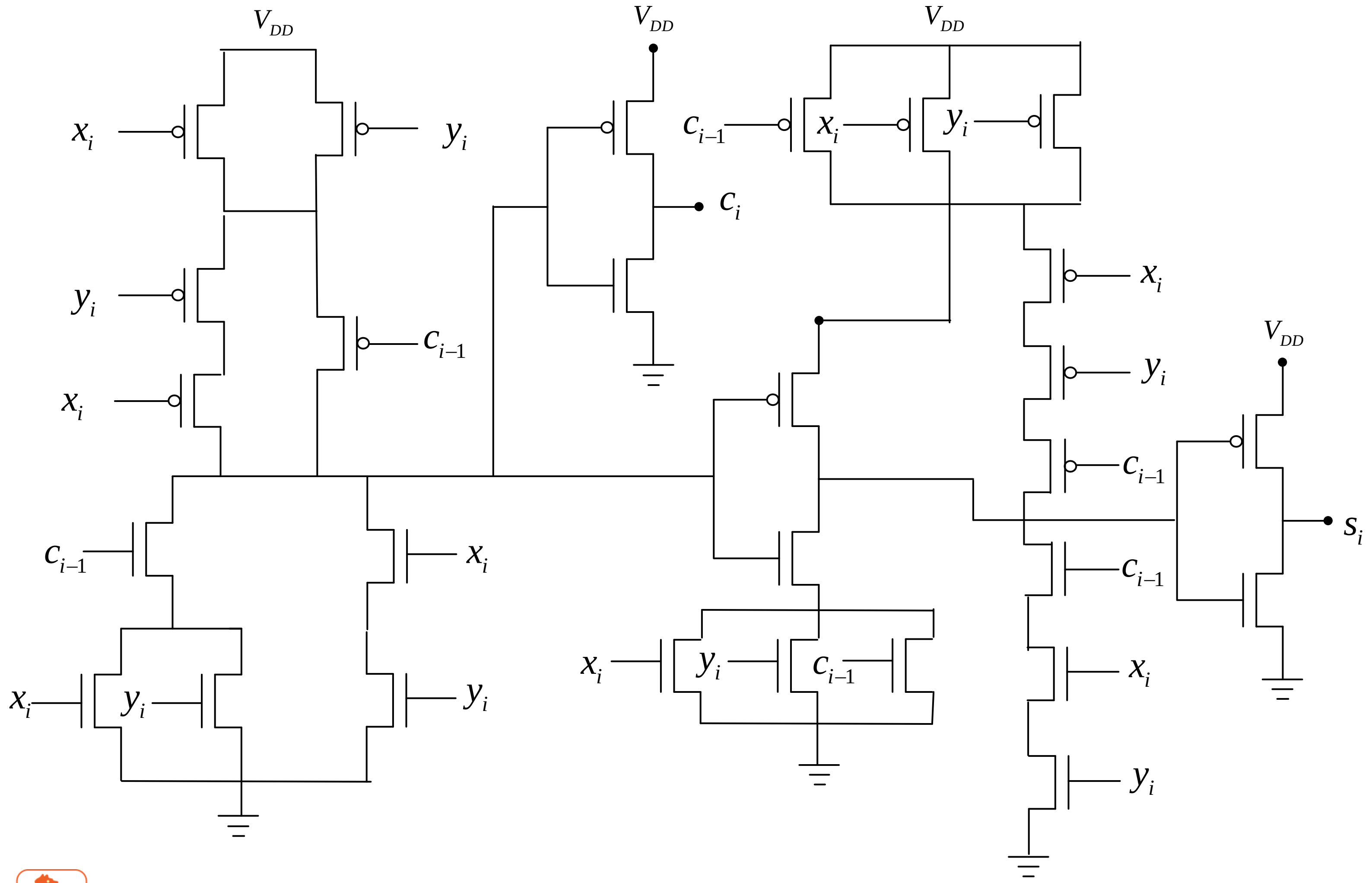
- Brute Force Implementation of Full Adder



- Brute Force Implementation of Full Adder (Pass Transistor Logic)

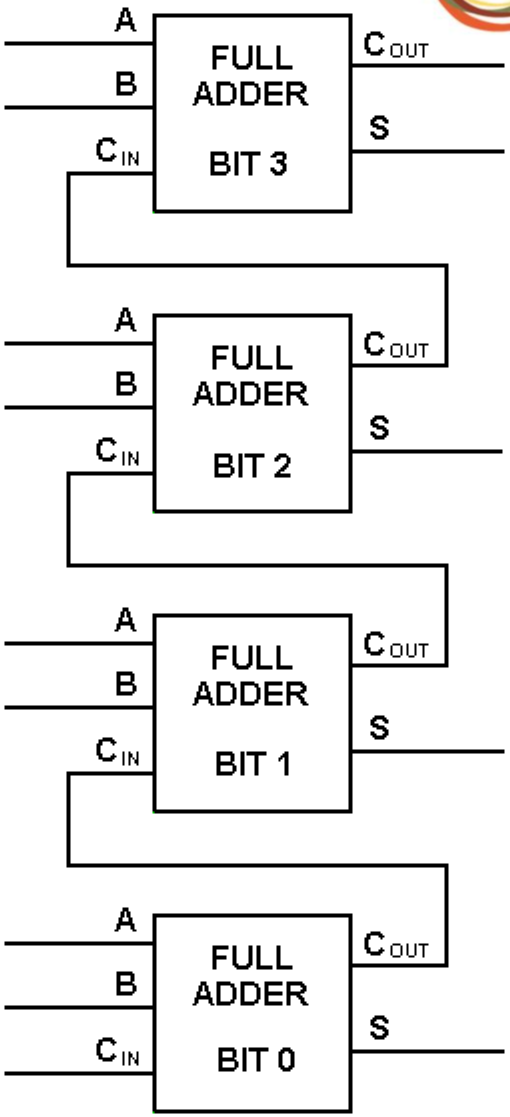


- CMOS Full Adder



- Full Adder
- If we have carryout propagating

from bit 0 to bit 3, then the delay will be increased. The more bits we design the more critical delay it may have.

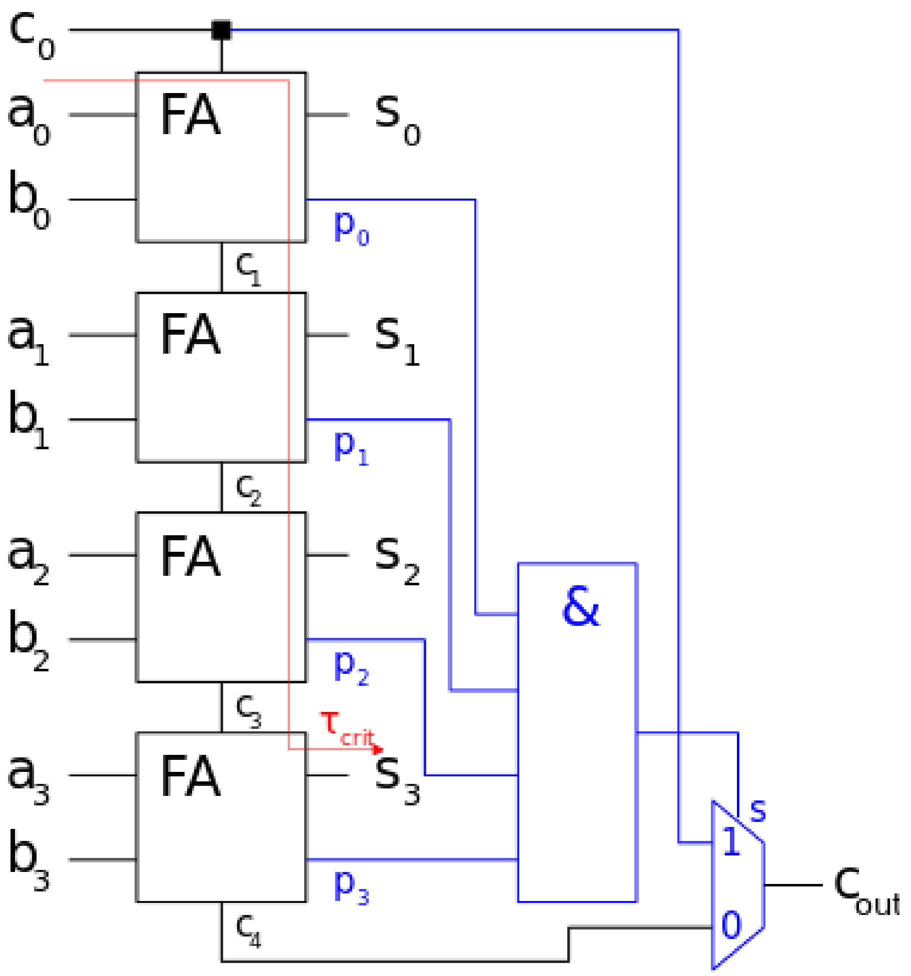


Full Adder

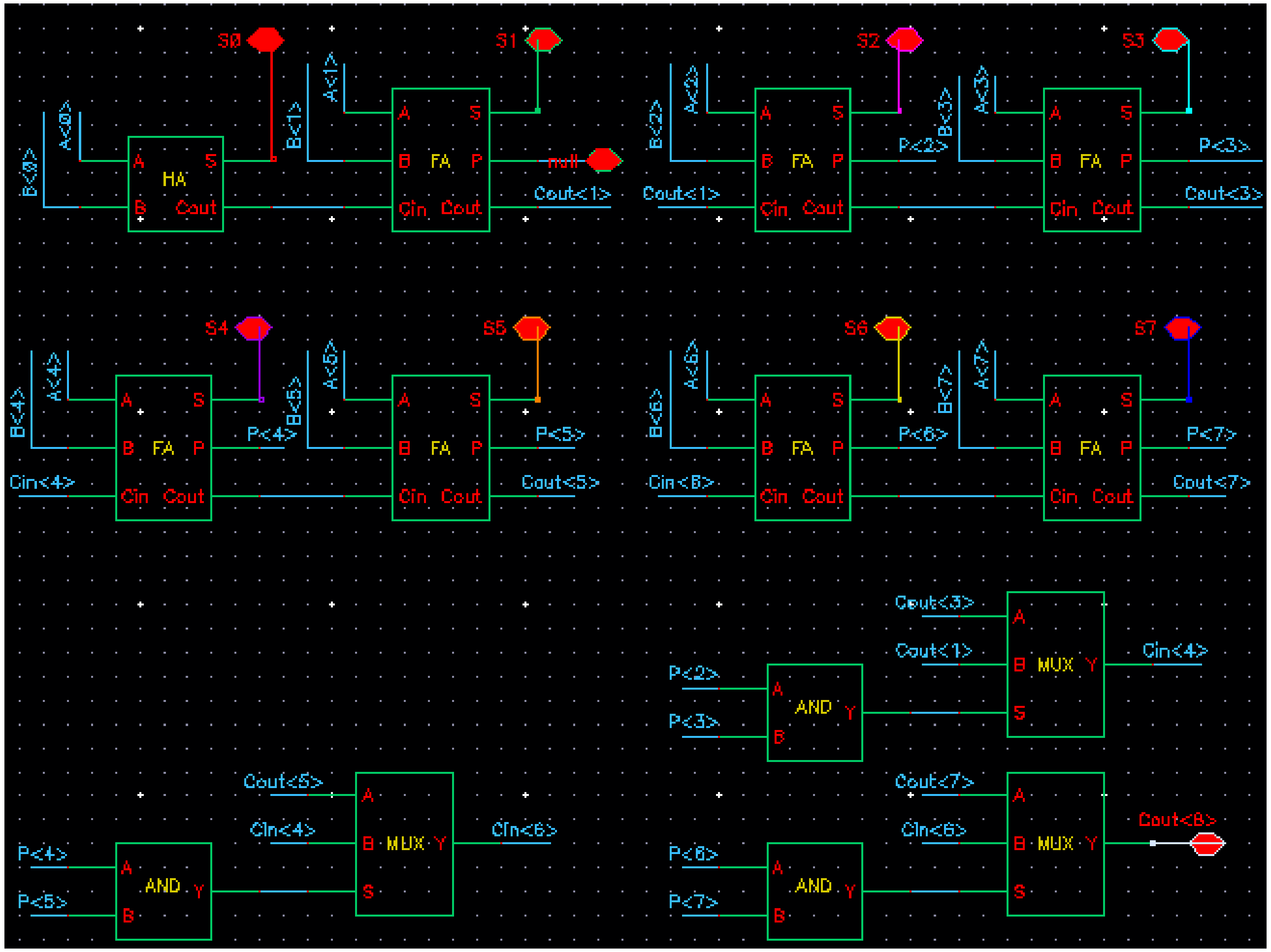
- Carry Skip Adder
- Calculate the “carry out” at the same time as the sum calculation
- Takes the “propagate” of each adder (a   b) into an AND gate as inputs
- If the AND gate output is “0” the output of the MUX is Cout of the final full adder
- If all propagates are “1” then the output of the AND gate is “1” and selector of the MUX chooses C0

- If

- Carry Skip Adder

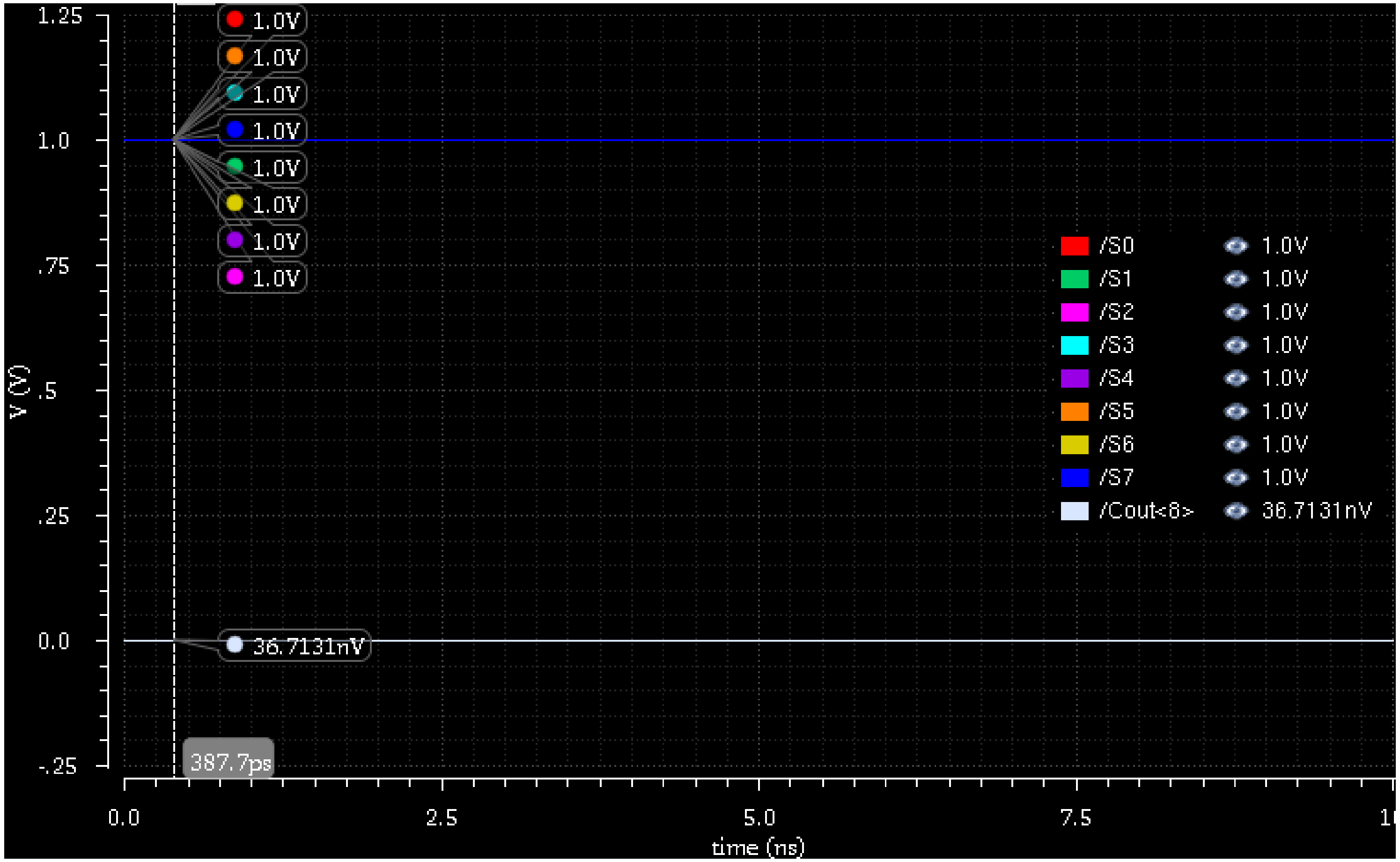


- 8-bit Carry Skip Adder



- 8-bit Carry Skip Adder

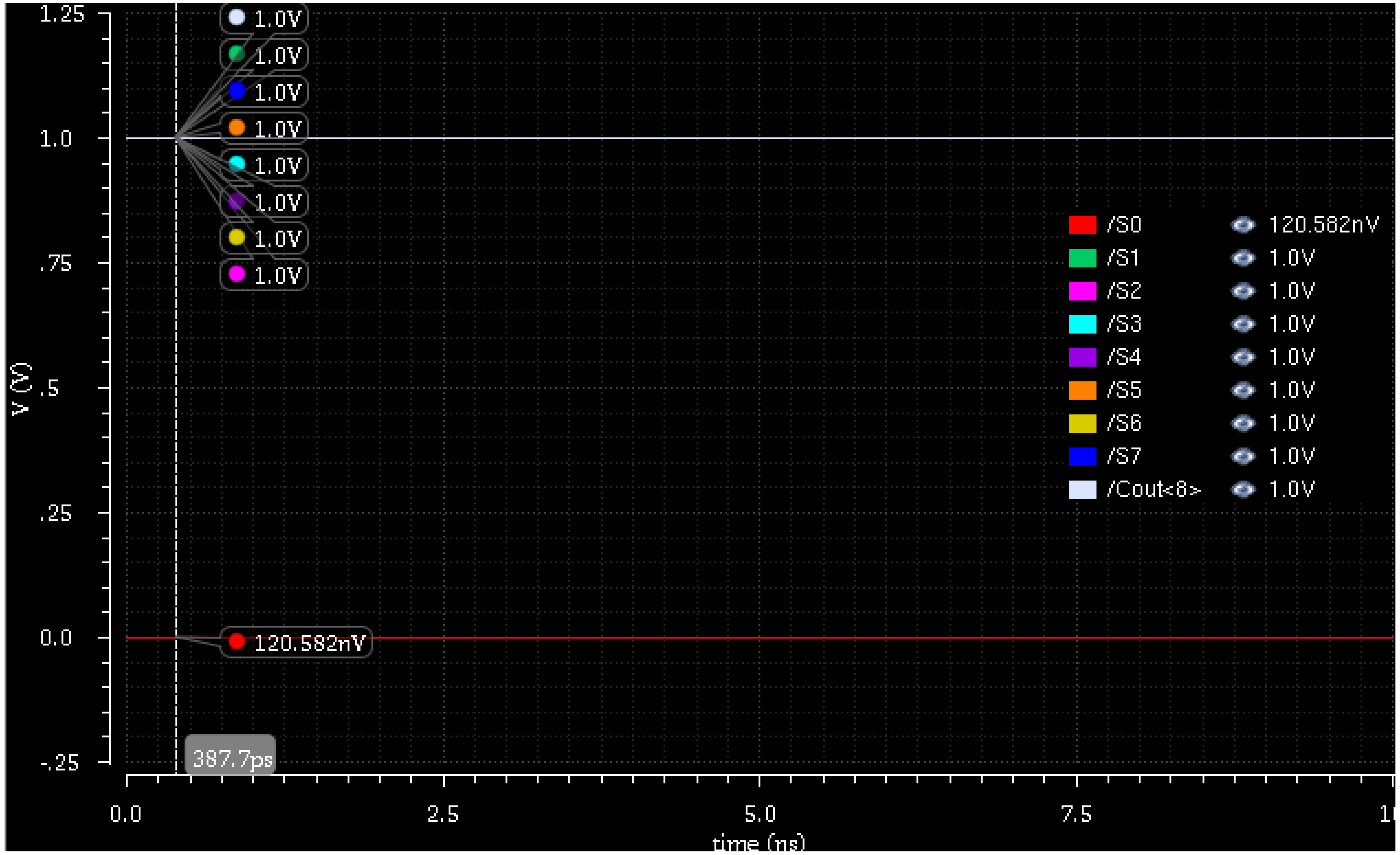
```
 00000000
+11111111
011111111
```



🟥	/S0	1.0V
🟩	/S1	1.0V
🟪	/S2	1.0V
🟦	/S3	1.0V
🟪	/S4	1.0V
🟧	/S5	1.0V
🟨	/S6	1.0V
🟦	/S7	1.0V
⬜	/Cout<8>	36.7131nV

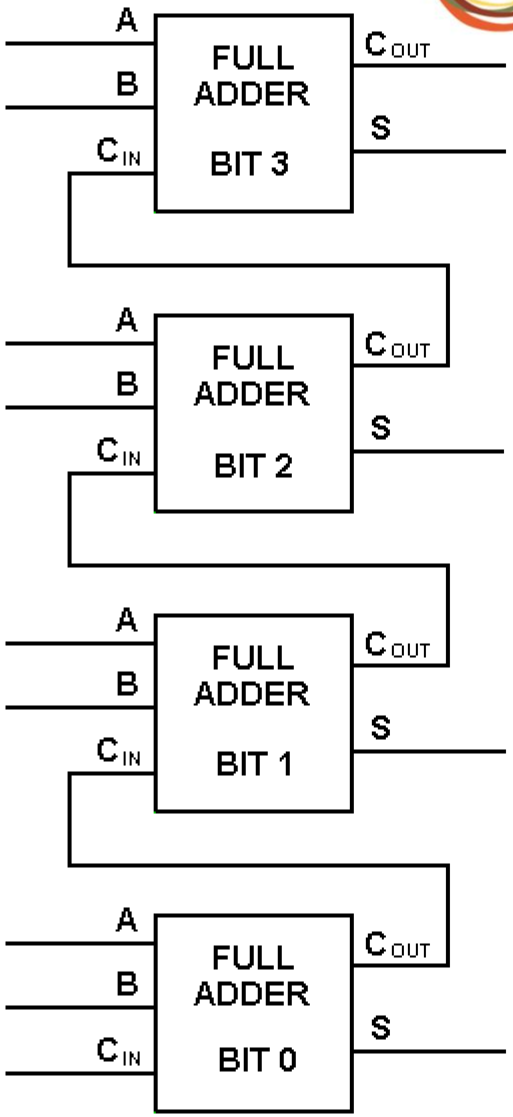
- 8-bit Carry Skip Adder

$$\begin{array}{r} 11111111 \\ \underline{+11111111} \\ 111111110 \end{array}$$



- Full Adder
- If we have carryout propagating

from bit 0 to bit 3, then the delay will be increased. The more bits we design the more critical delay it may have.



- Carry Lookahead Adder (CLA)
- Advantage
- Fast adder
- Good for high speed digital design.

- Disadvantage
- Area consumption
- Extra gates for the functions of ‘Propagate’ and ‘Generate’.

- To resolve the delay issue on the Ripple-Carry Adder, they add some more logics to make the adder become CLA. The CLA's calculation does no longer depend on the previous carry bit; all the SUMs' calculations are depending on the first Cin(0) and the input values only. Therefore, the delay is greatly improve if you have a huge input # design.

- Defining 3 new variables that only depends on x_i and y_i
- ❶ Generate (g): $C_i = 1$ independent of C_{i-1}

  $$g_i = x_i y_i$$

- ❷ Propagate (p): $C_{i-1} = C_i$

  $$p_i = x_i \oplus y_i$$

- ❸ Kill, Delete(d): $C_i = 0$ independent of C_{i-1}

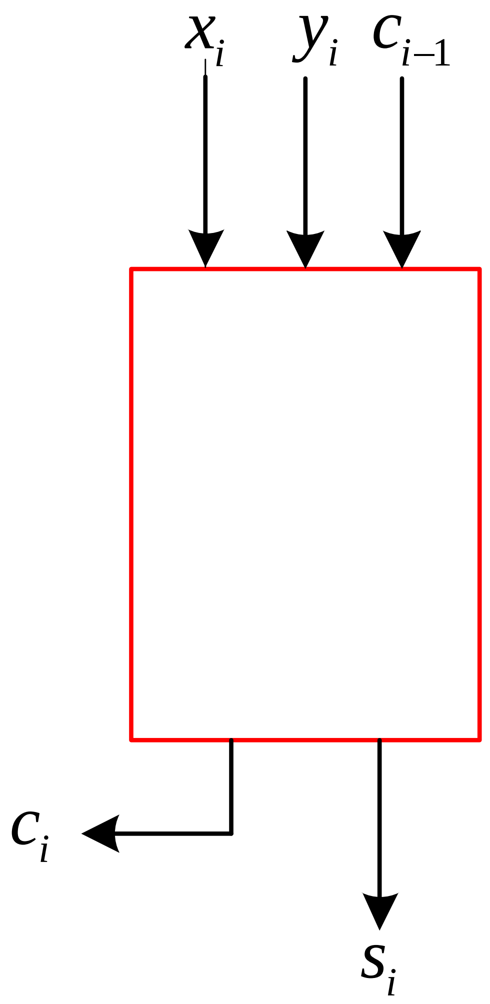
  $$d = \overline{x_i}\,\overline{y_i}$$

- Full-Adder

$$s_i = x_i \oplus y_i \oplus c_{i-1}$$

$$c_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$$

c_{i-1}	x_i	y_i	s_i	c_i	
Inputs			Outputs		
0	0	0	0	0	Delete
0	0	1	1	0	Propagate
0	1	0	1	0	Propagate
0	1	1	1	1	Generate
1	0	0	1	0	Delete
1	0	1	0	1	Propagate
1	1	0	0	1	Propagate
1	1	1	1	1	Generate



- Expressing sum and carry in terms of generate, propagte, and delete.

$$c_0 = g_0$$

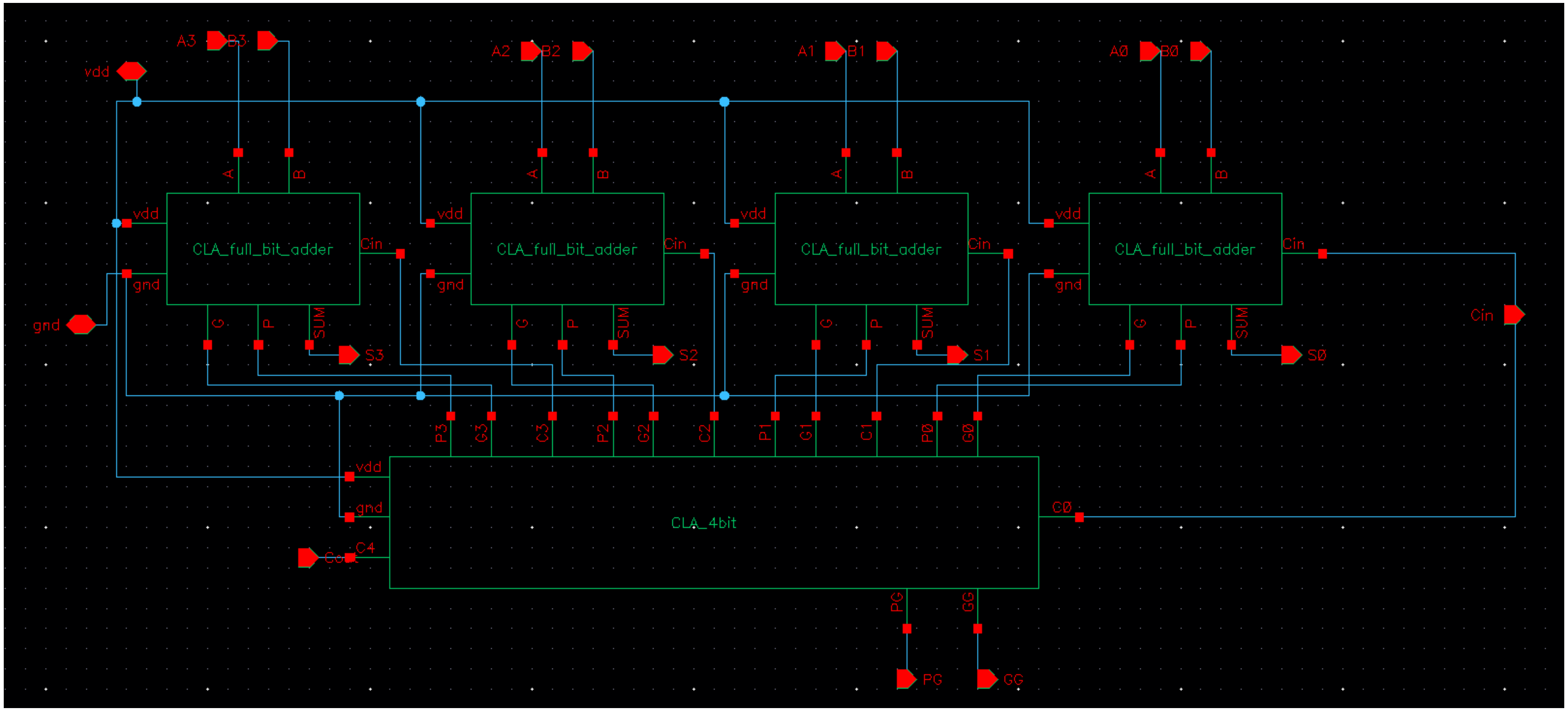
$$c_1 = g_1 + p_1 c_0 = g_1 + p_1 g_0$$

$$c_2 = g_2 + p_2 c_1 = g_2 + p_2 g_1 + p_2 p_1 g_0$$

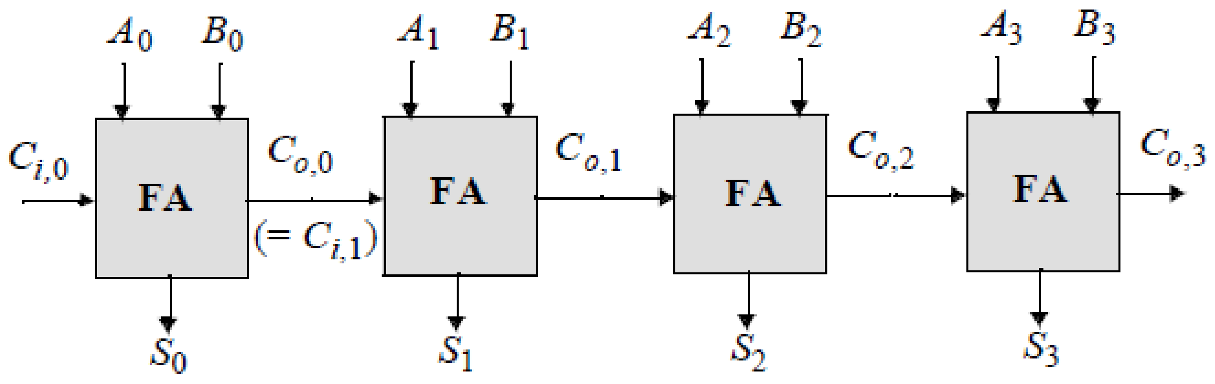
$$s_i = x_i \oplus y_i \oplus c_{i-1} = p_i \oplus c_{i-1}$$

$$c_i(g, p) = g + p c_{i-1}$$

- Carry Lookahead Adder (CLA)



- The Ripple-Carry Adder



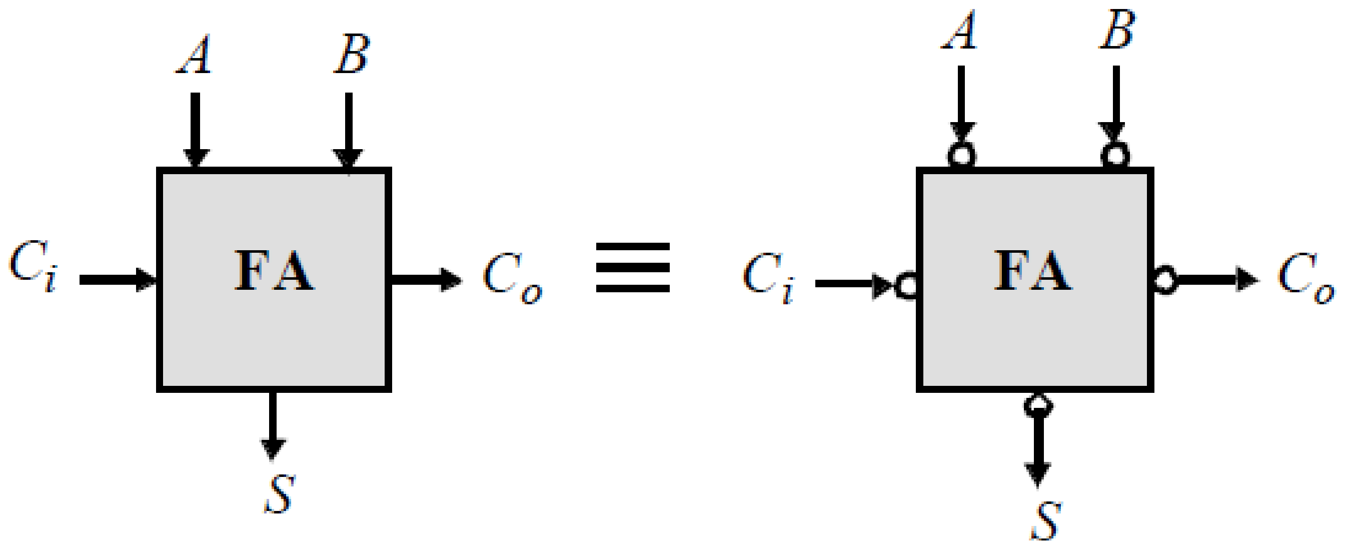
Worst case delay linear with the number of bits

$t_d = O(N)$

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

Goal: Make the fastest possible carry path circuit

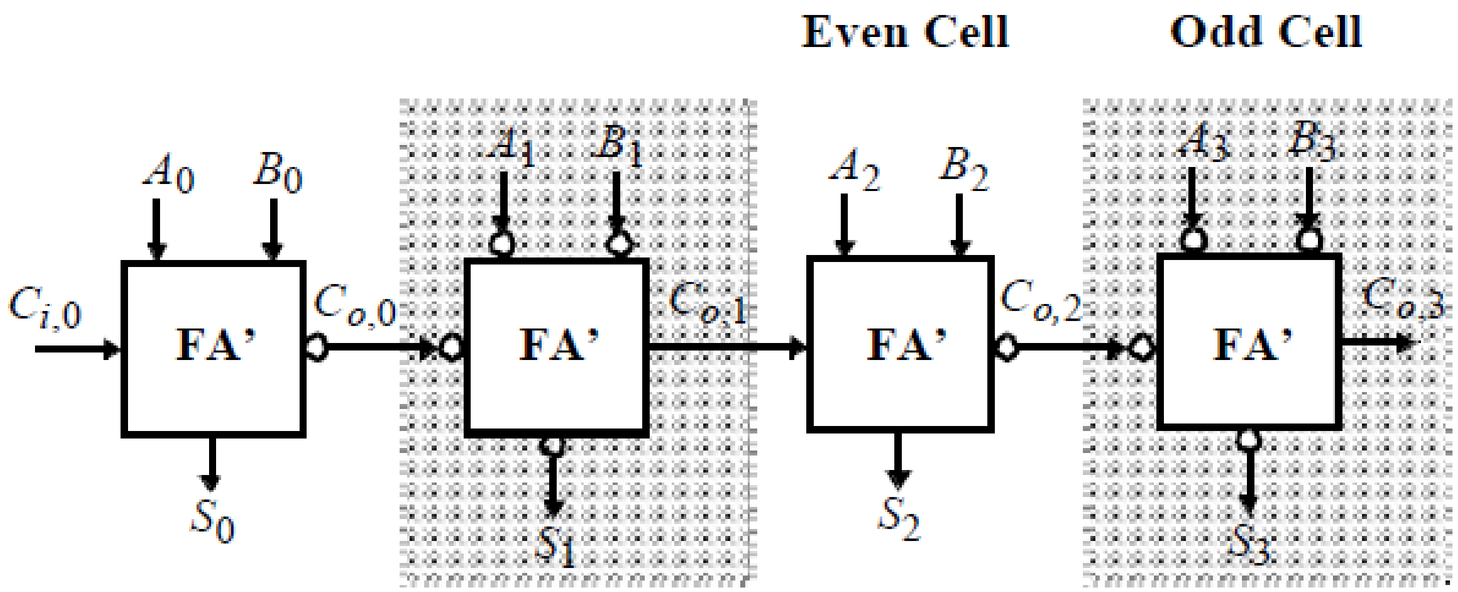
- Inversion Properties



$$\overline{S}(A,B,C_i) = S(\overline{A},\overline{B},\overline{C_i})$$

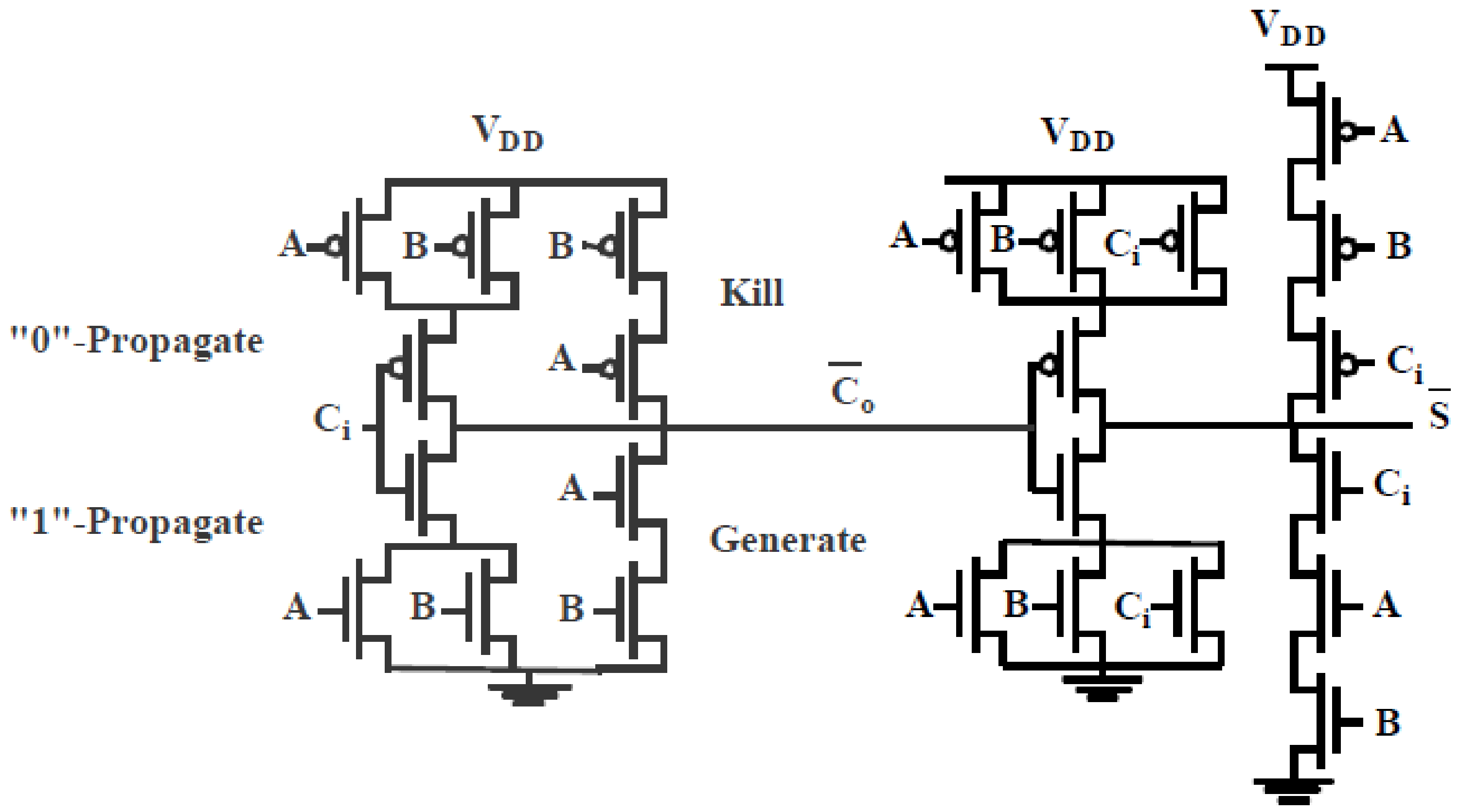
$$\overline{C_o}(A,B,C_i) = C_o(\overline{A},\overline{B},\overline{C_i})$$

- Adder



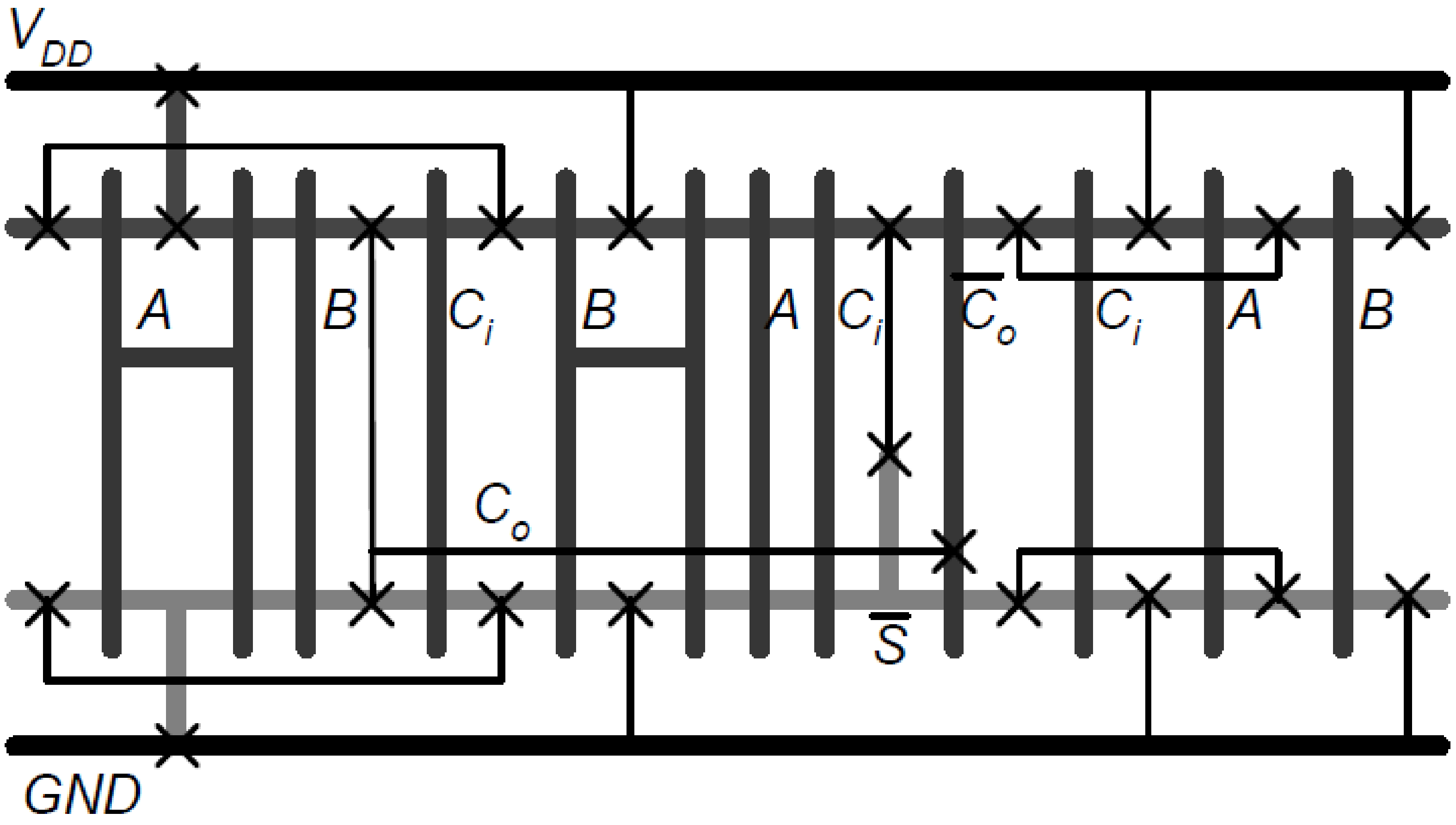
Exploit Inversion Property

- The Mirror Adder

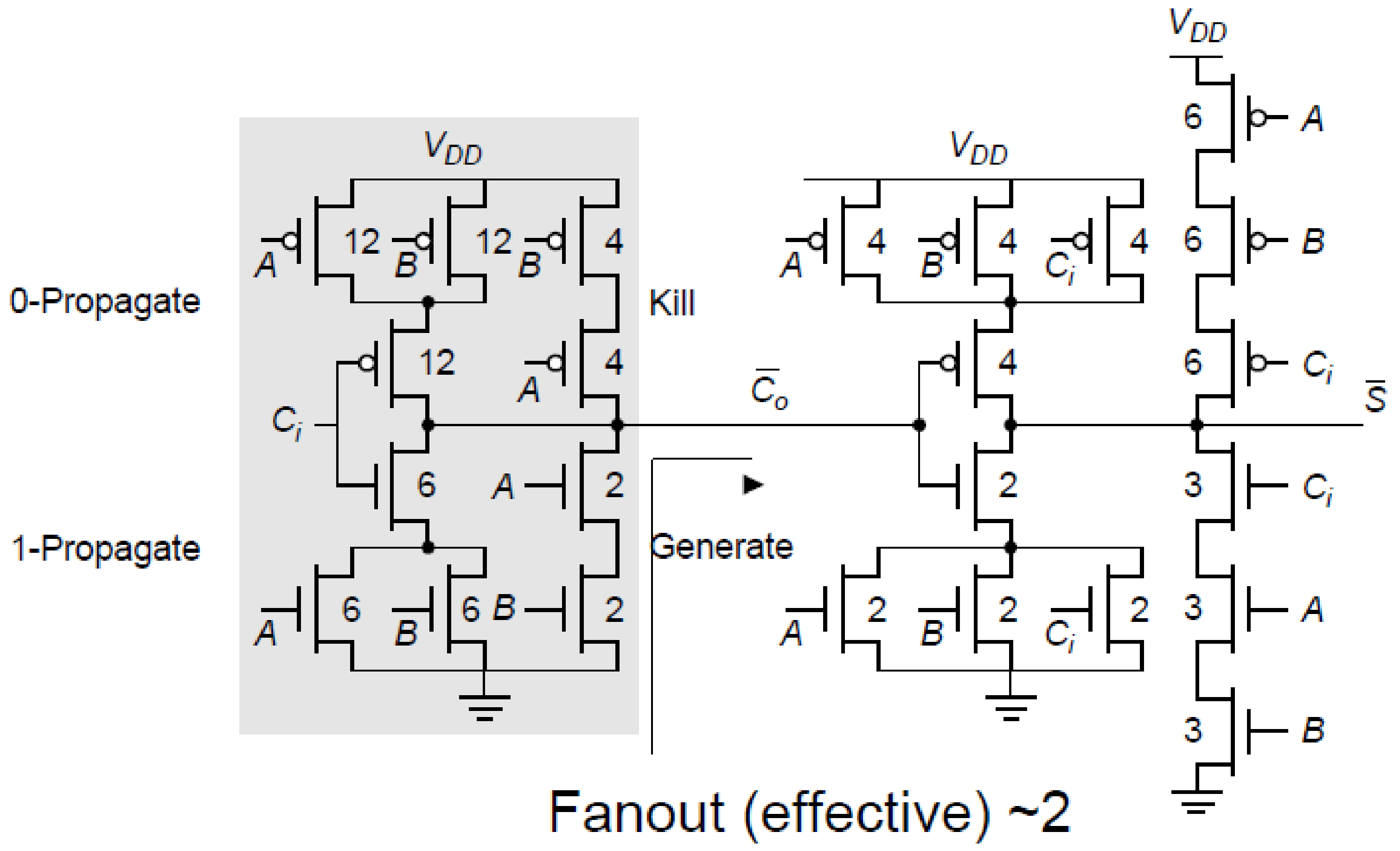


24 transistors

- The Mirror Adder Cell

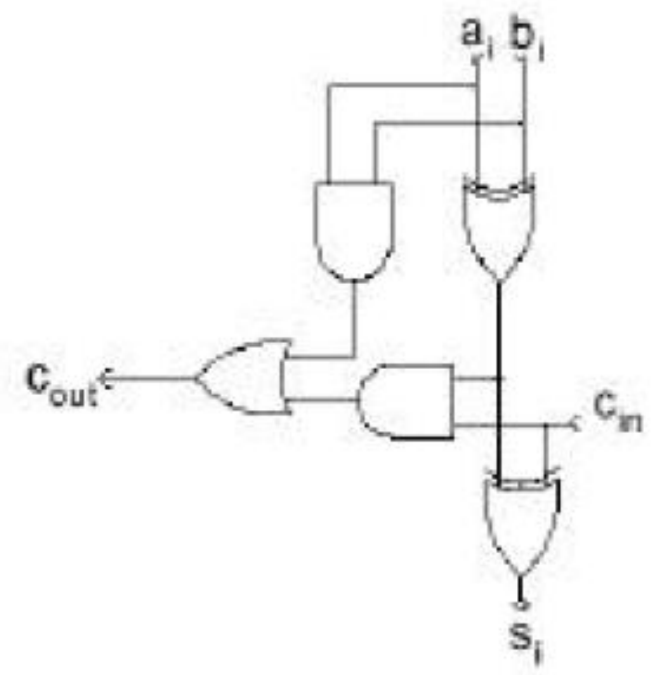


- The Mirror Adder

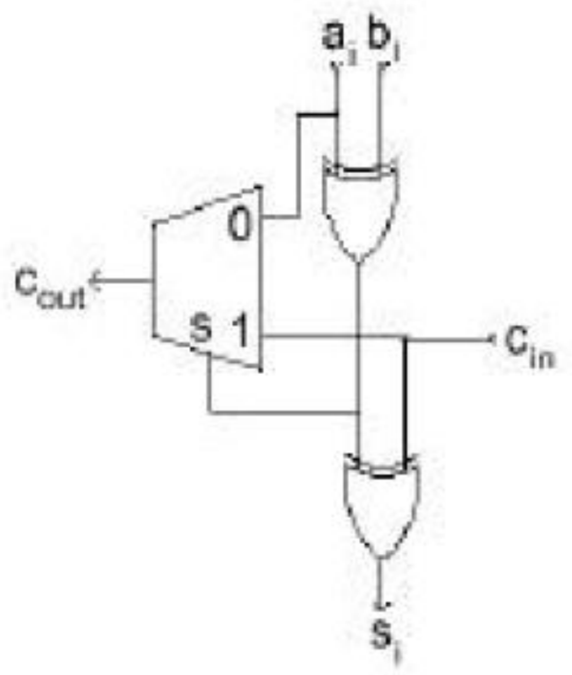


Fanout (effective) ~2

- Full Adder Implementation

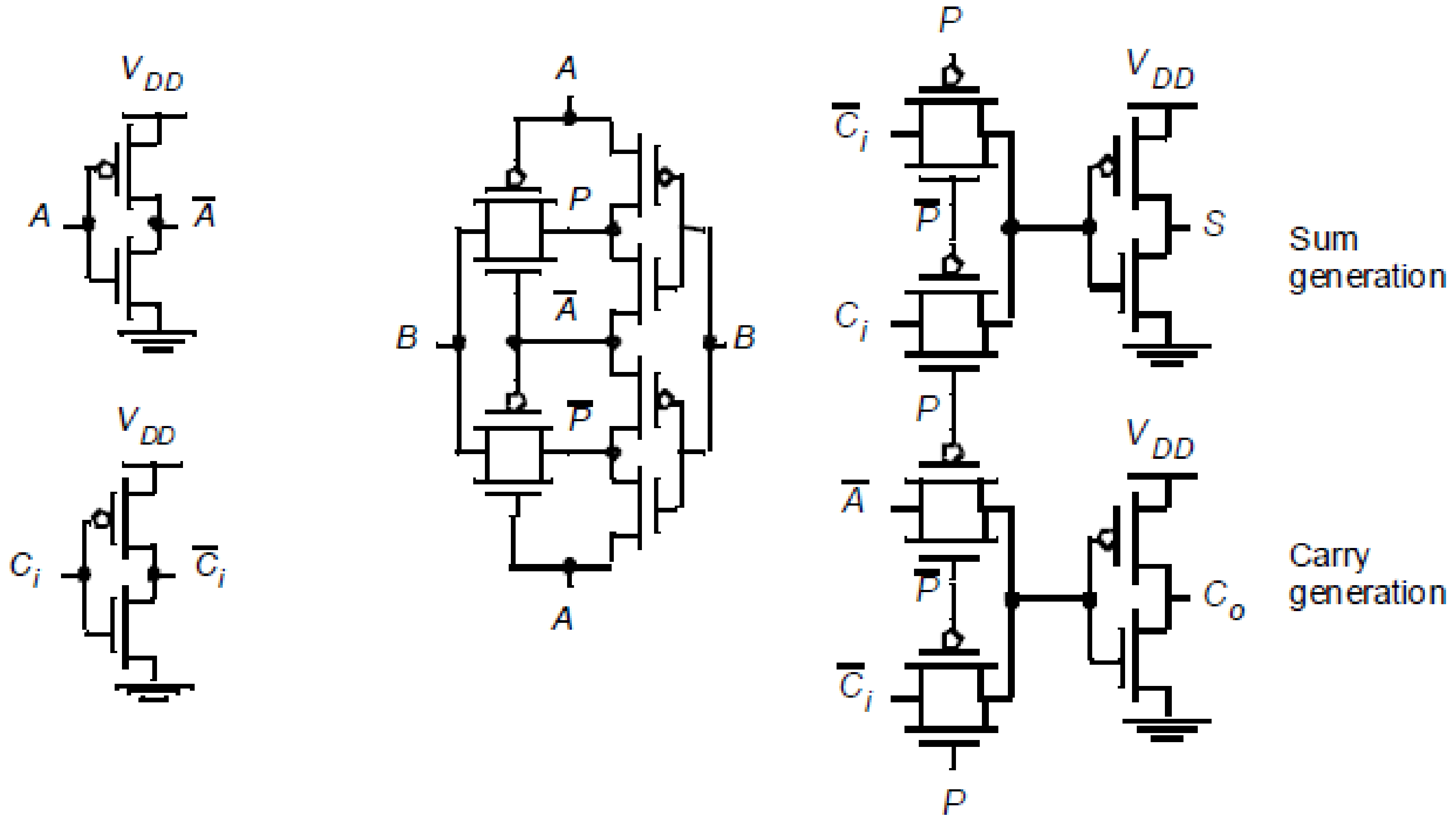


Standard CMOS

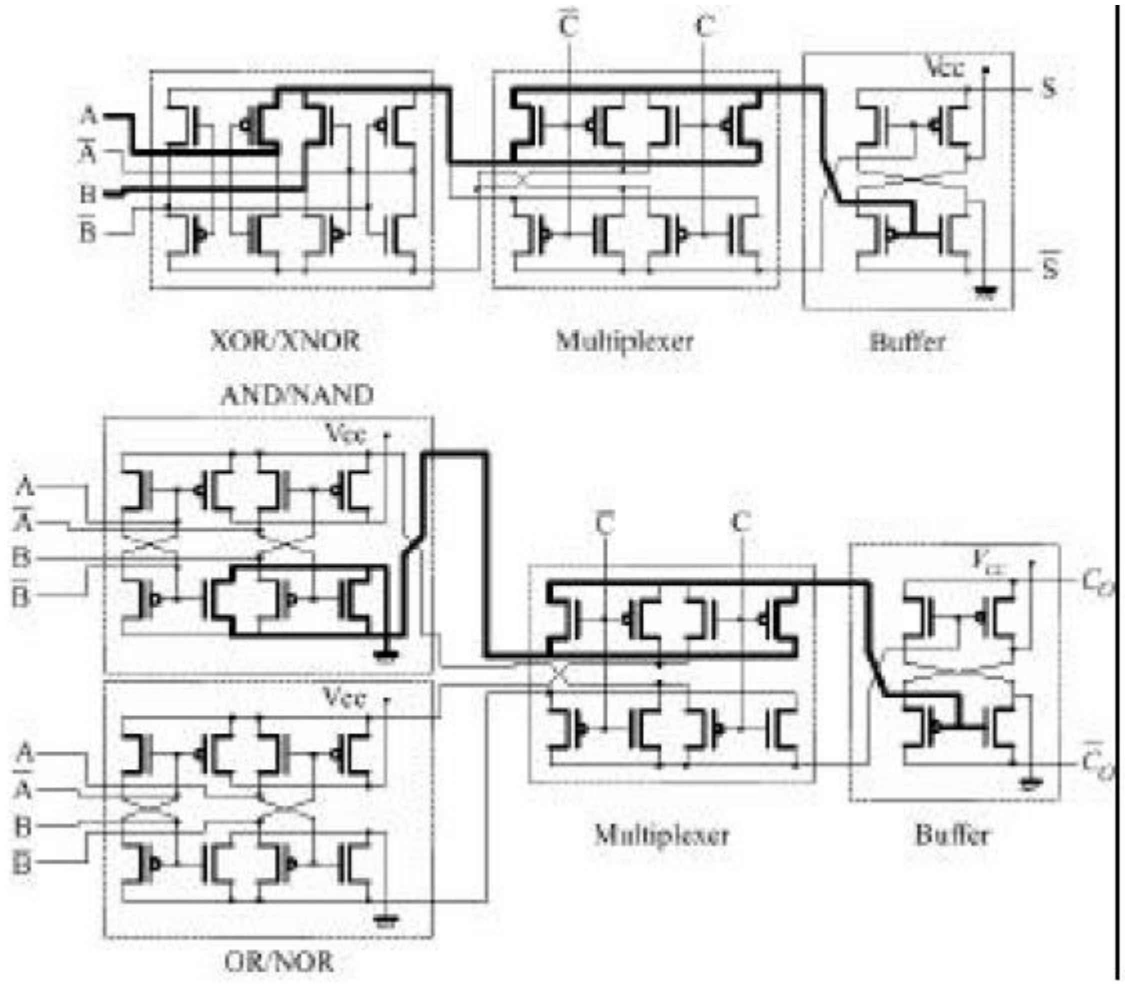


Multiplexer-based

- TG-based Full adder

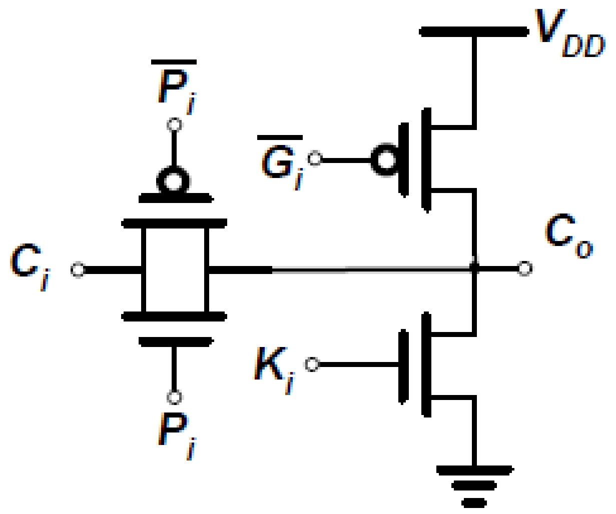


- Full Adder in DPL

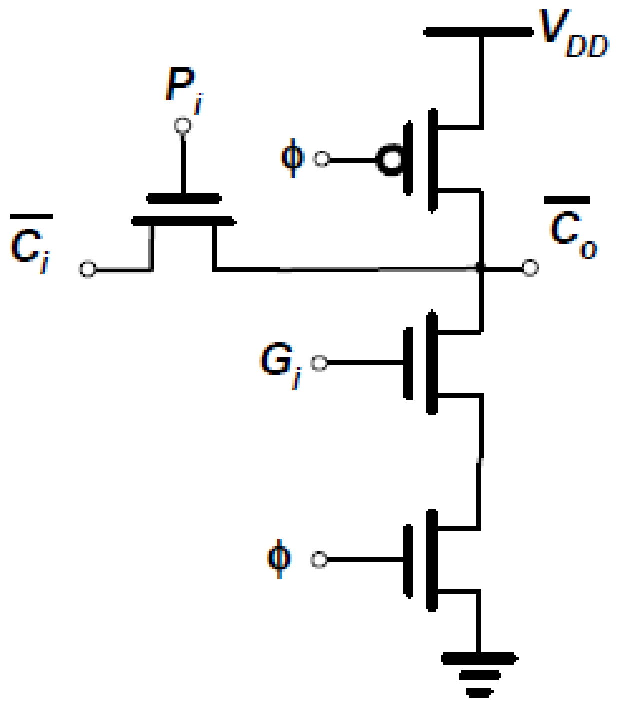


- Manchester Carry Chain

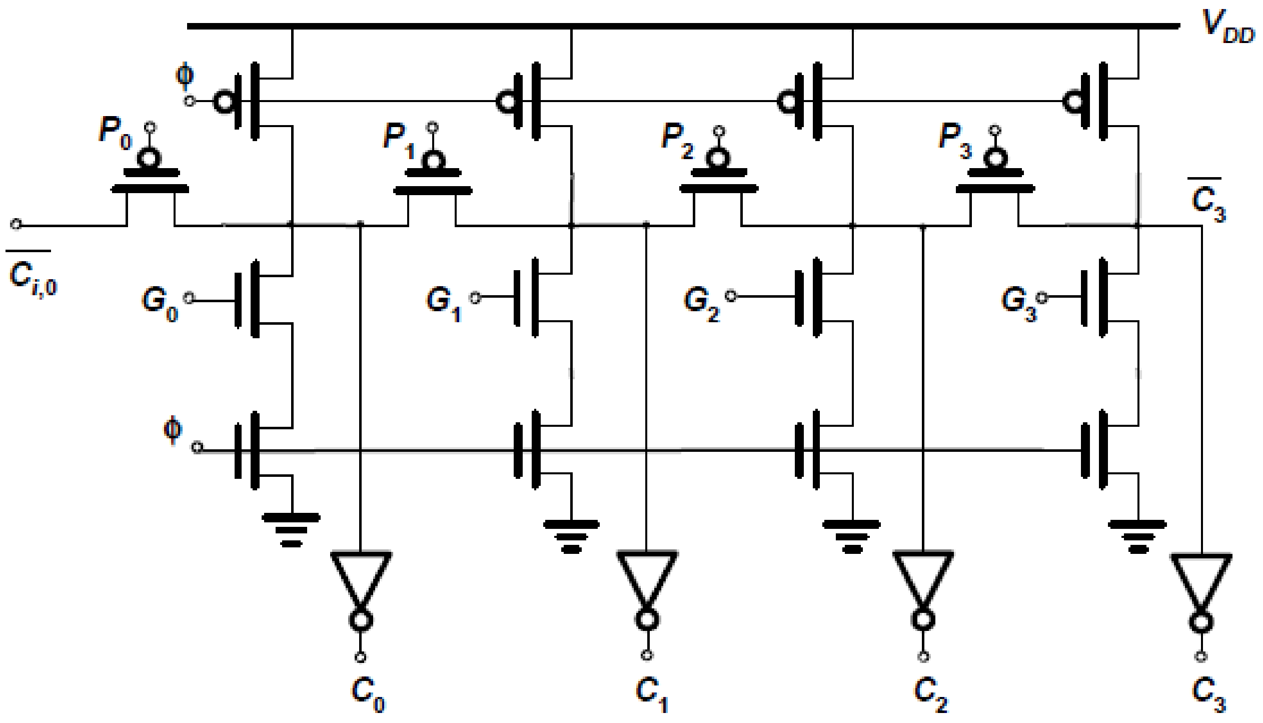
Static



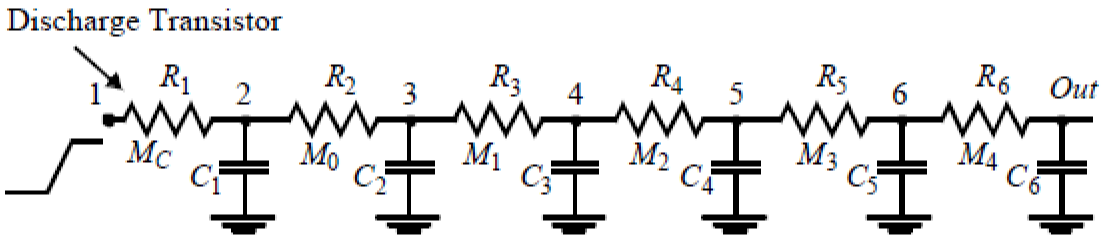
Dynamic



- Manchester Carry Chain
  - Implement P with pass-transistors
  - Implement G with pull-up, kill (delete) with pull-down
  - Use dynamic logic to reduce the complexity and speed up

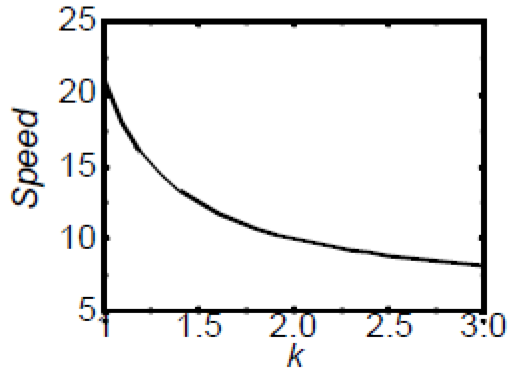


- Sizing Manchester Carry Chain

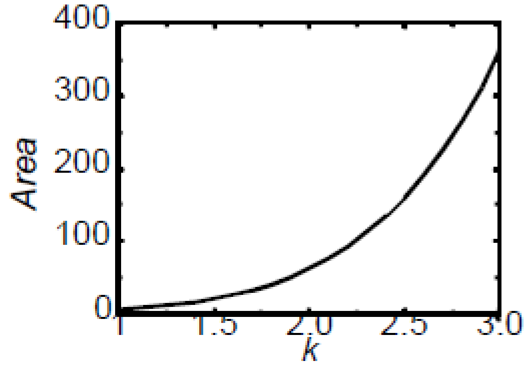


$$t_p = 0.69 \sum_{i=1}^{N} C_i \left(\sum_{j=1}^{i} R_j \right)$$

Tapering?



Speed (normalized by 0.69*RC*)



Area (in minimum size devices)

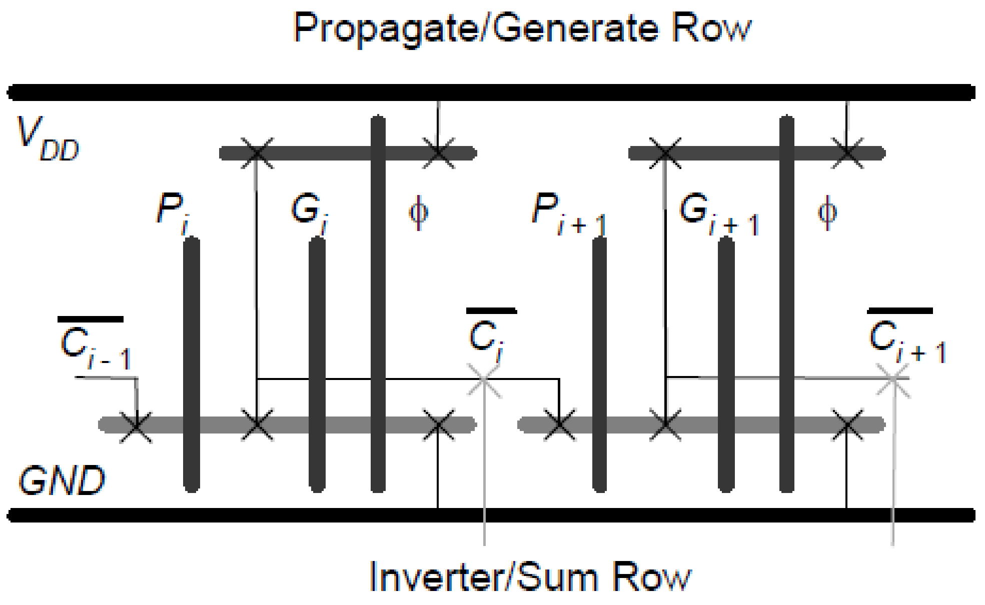
- Sizing Manchester Carry Chain

- Delay equation

$$t_p = 0.69 \sum_{i=1}^{N} C_i \left(\sum_{j=1}^{i} R_j \right) = 0.69 \frac{N(N+1)}{2} RC$$

- Delay is quadratic with N
  - » Progressive sizing should help?

- Sizing Manchester Carry Chain
  - Stick Diagram



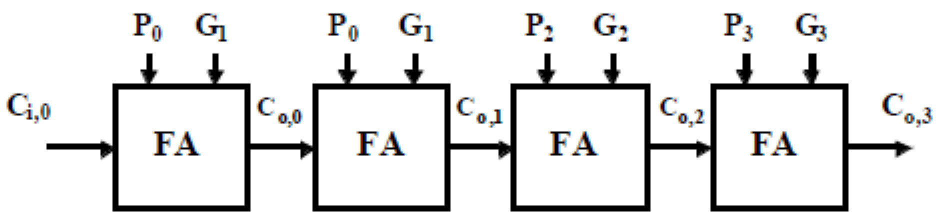
C_{fix} – fixed capacitance at the node (pull-down, pull-up diffusions, metal, + inverter ~15fF
C ~ 2fF/μm
R ~ 10kΩ μm
When $CW > C_{fix}$
small improvements with sizing,
Loading of the input stage

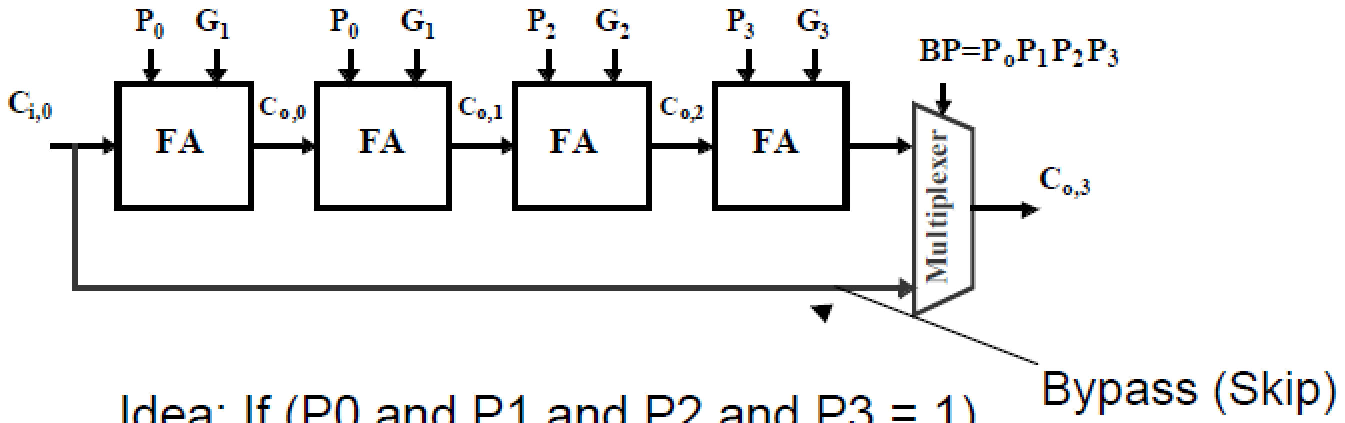
$$t_p = 0.69\frac{N(N+1)}{2}RC = 0.69\frac{N(N+1)}{2}\frac{R}{W}\left(C_{fix} + C \cdot W\right)$$

Adder

- Manchester Carry Chain
- Length of chain is limited to *k* = 4-8
- Standard solution – add inverters
- The overall *N*-bit adder delay is a sum of *N*/*k* segments (linear)

- Carry-skip Adder

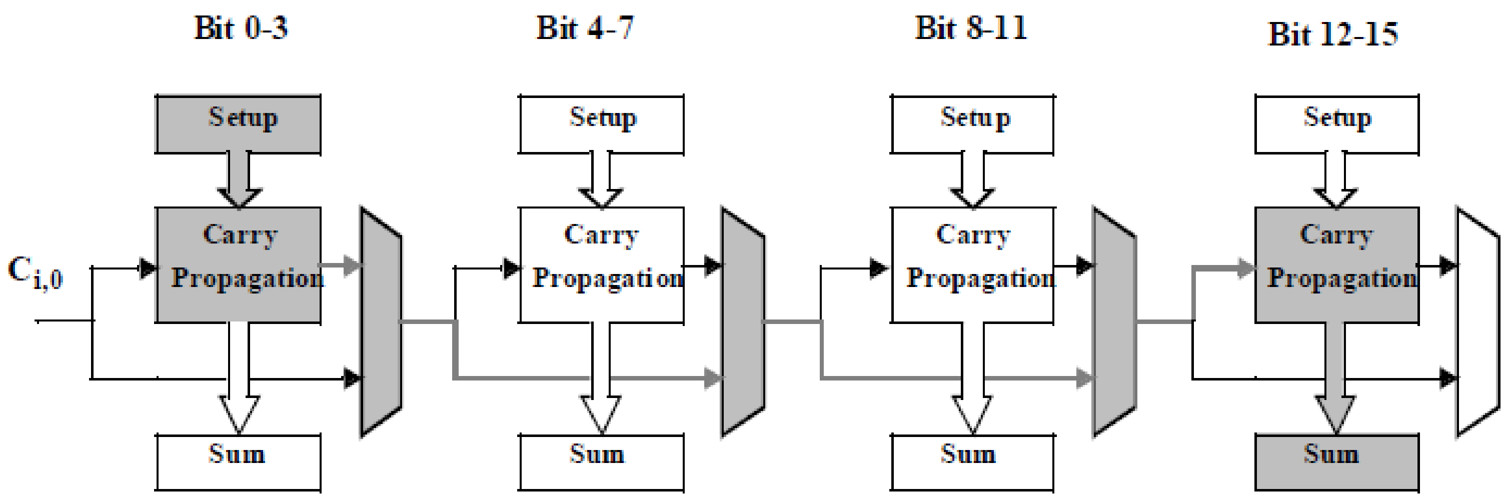




Idea: If (P0 and P1 and P2 and P3 = 1)
then $C_{o3} = C_0$, else “kill” or “generate”.

MacSorley, Proc IRE 1/61
Lehman, Burla, IRE Trans on Comp, 12/61

- Carry-skip Adder

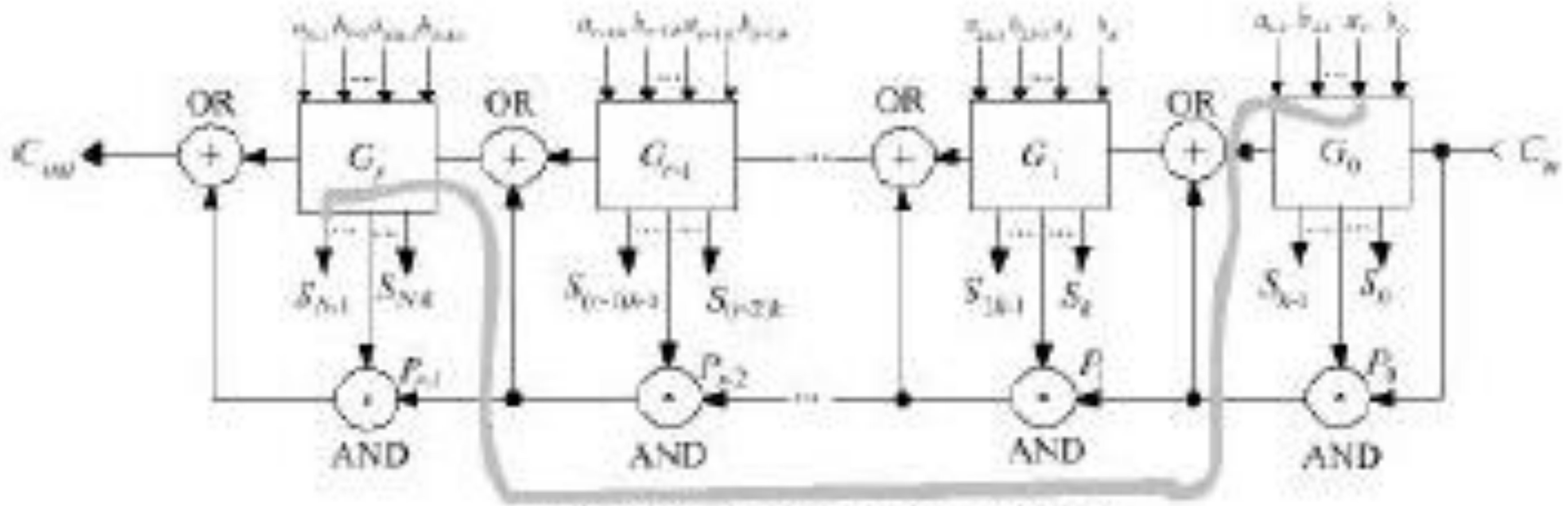


Critical Path

For N-bit adder with k-bit groups

$$t_d = (k-1)t_{RCA} + \left(\frac{N}{k} - 2\right)t_{SKIP} + (k-1)t_{RCA}$$

- Carry-skip Adder

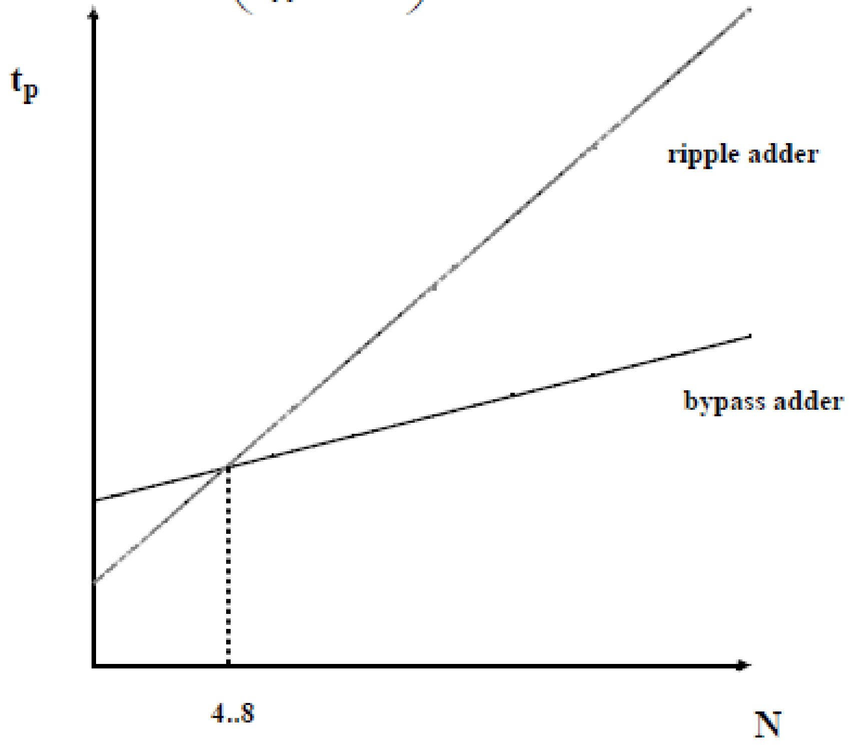


Critical path, delay Δ=2(k-1)+(N/k-2)

- Carry-skip Adder

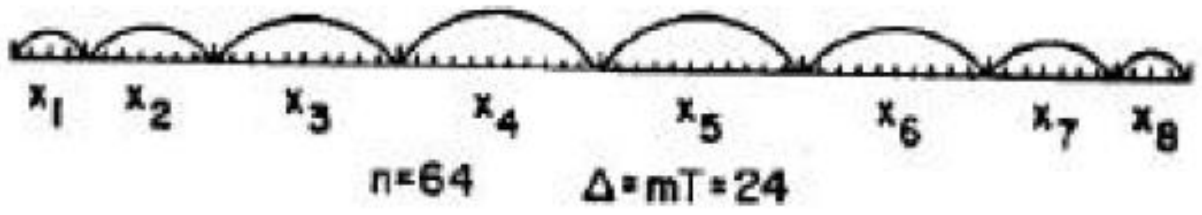
Critical path delay with constant groups

$$t_d = 2(k-1)t_{RCA} + \left(\frac{N}{k} - 2\right)t_{SKIP}$$



- Carry-skip Adder

Variable Group Length

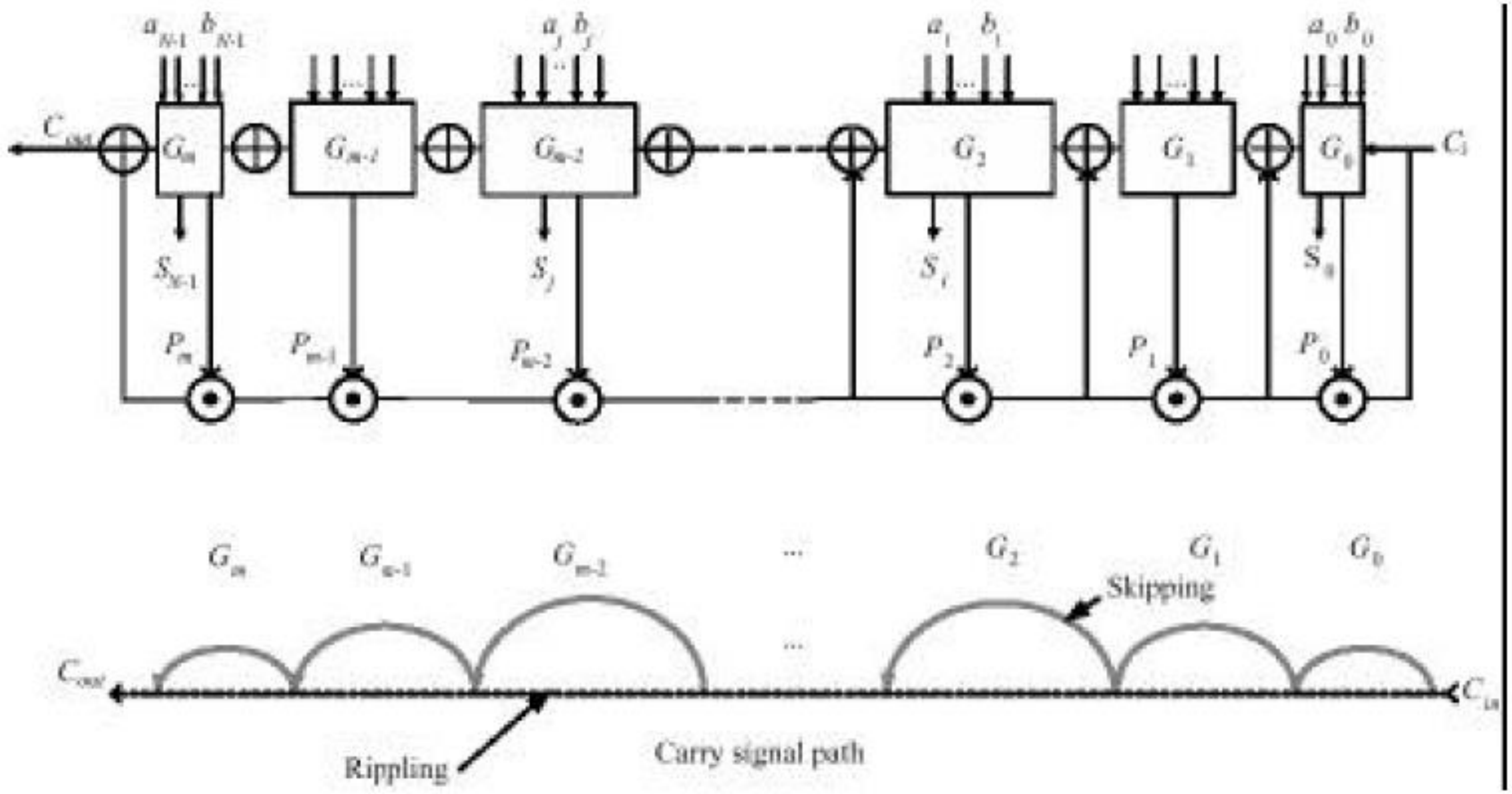


$x_1 = x_8 = 4,\ x_2 = x_7 = 7,\ x_3 = x_6 = 10,\ x_4 = x_5 = 11.$

$$t_d = c_1 + \sqrt{c_2 N + c_3}$$

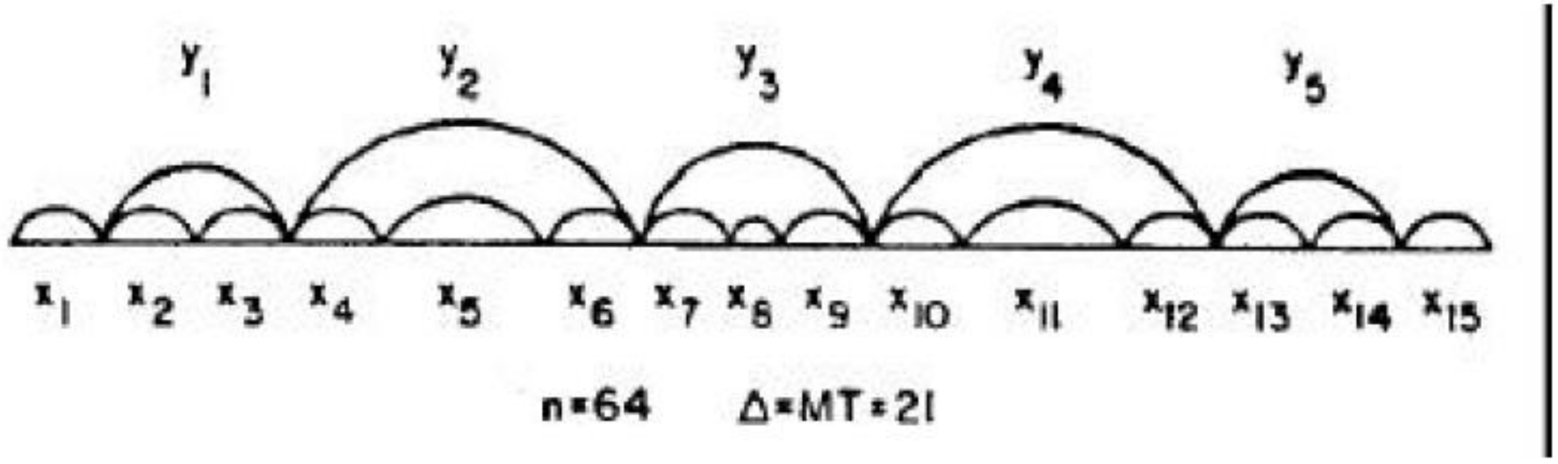
Oklobdzija, Barnes, Arith'85

- Carry-skip Adder



- Carry-skip Adder

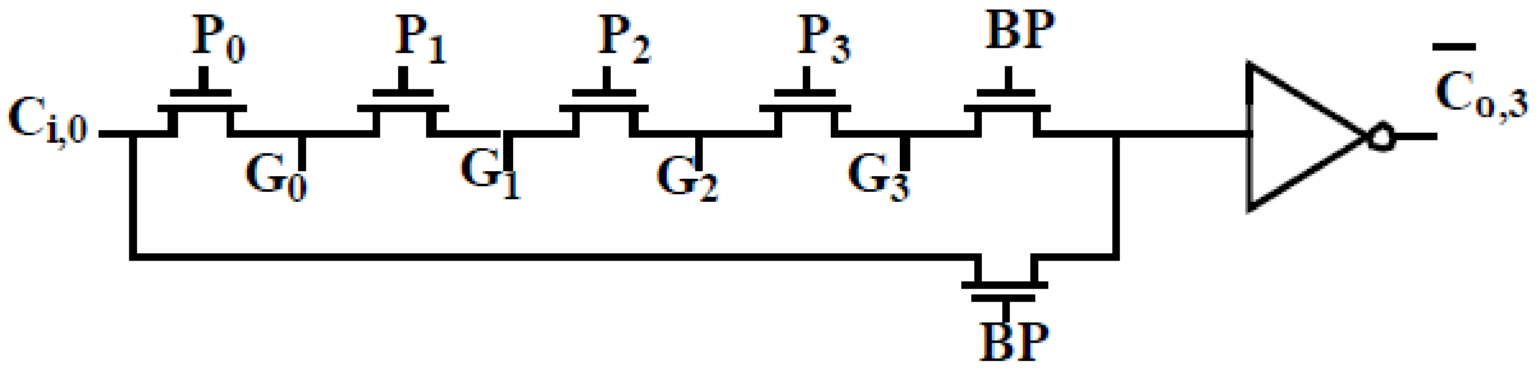
Variable Block Lengths



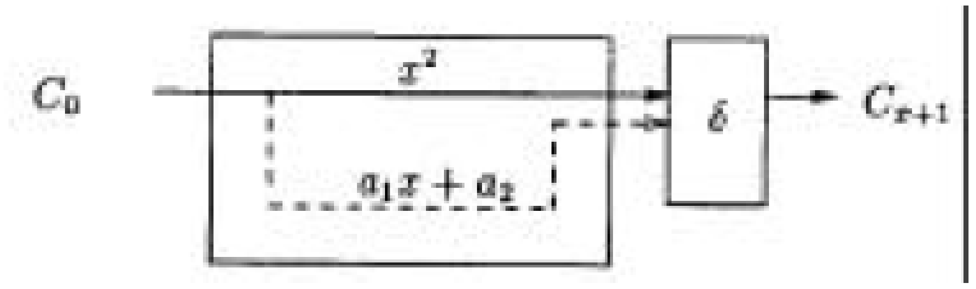
y_1
y_2
y_3
y_4
y_5
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
n=64
Δ=MT=21

Oklobdzija, Barnes, Arith'85

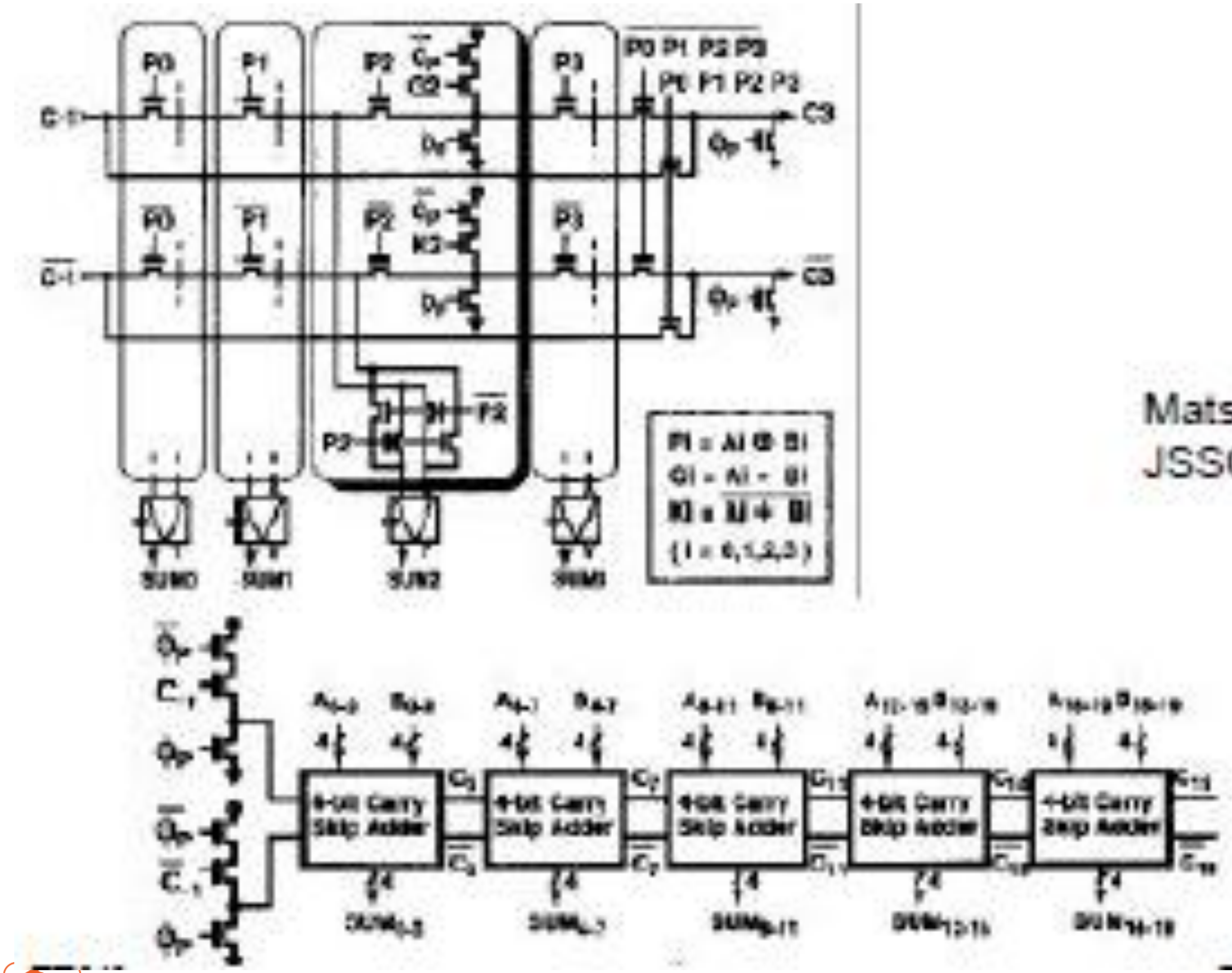
- Manchester Chain with Carry-skip Adder



Delay model:



- PTL with SA-F Implementation



Matsui,
JSSC 12/94