Binary Adders

- Addition is important function in computer system
  - Duh, but still.
- What does an adder have to do?
  - Add binary digits
  - Generate carry if necessary
  - Consider carry from previous digit

HA

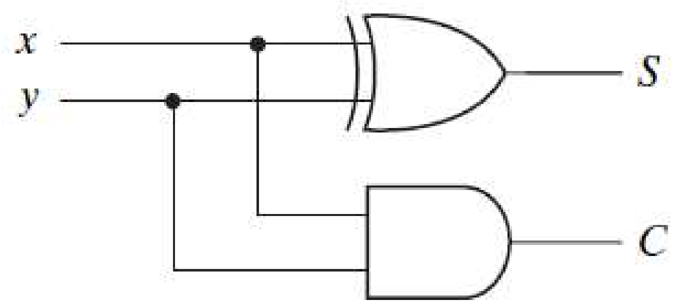
FA

- Binary adders operate “bit-wise”
  - A 16-bit adder uses 16 one-bit adders
- Binary adders come in two flavors
  - Half adder : adds two bits and generate result and carry
  - Full adder : also considers carry input
  - Two half adders (plus an OR gate) make one full adder

Binary Half Adder

- Specification:
  - Design a circuit that adds two bits and generates the sum and a carry
- Outputs:
  - Two inputs: *x, y*
  - Two output: *S* (sum), *C* (carry)
- Truth table:

$$\begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}$$



(b) $S = x \oplus y$
$C = xy$

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

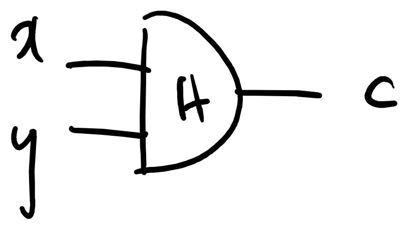
$C(x,y) = xy$

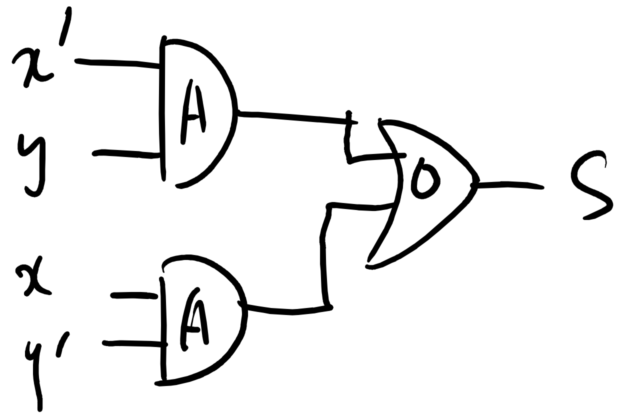
$S(x,y) = \Sigma(1,2) = \underline{x'y + xy'} = \underline{x \oplus y}$

Binary Half Adder - Circuit

- Circuit implementation:

$C(x,y) = xy$ $\quad S(x,y) = x'y + xy'$





- Works for single bit, but not for multi-bit numbers
  - <u>*Need to consider input carry from prior stage*</u>

x	*y*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

$$\begin{array}{rcc} & 0 & \\ & 0 & 0 \\ + & 0 & 1 \\ \hline & 0 & 1 \end{array} \qquad \begin{array}{rcc} & 1 & \\ & 0 & 1 \\ + & 0 & 1 \\ \hline & 1 & 0 \end{array}$$

- Specification:
  - A circuit that adds three bits and generates the sum and a carry
- Inputs:
  - Three inputs: *x,y,z*
  - Two outputs: *S, C*
- Truth table:

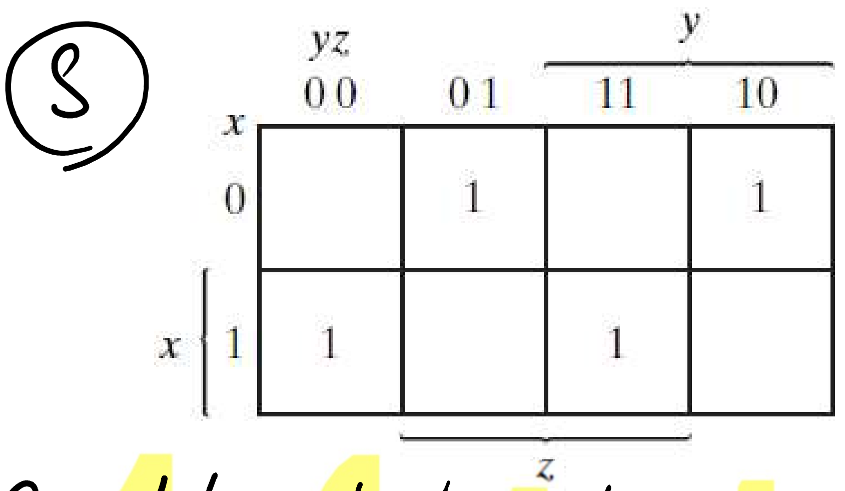
$C(x,y,z)=\underline{\Sigma(3,5,6,7)}$

$S(x,y,z)=\underline{\Sigma(1,2,4,7)}$

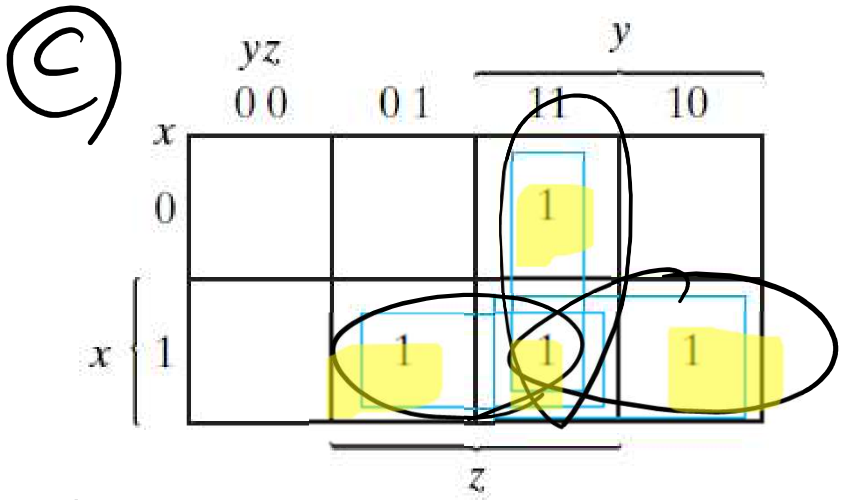
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder - Minimization

- Minimization of Boolean functions for *S* and *C*:



x \ yz	00	01	11	10
0		1		1
1	1		1	



x \ yz	00	01	11	10
0			1	
1		1	1	1

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$= x'(y'z + yz') + x(y'z' + yz)$$

$$= x'(y \oplus z) + x(y \oplus z)'$$

$$* \quad y \oplus z = A$$

$$= x'A + xA'$$

$$= x \oplus A = x \oplus (y \oplus z)$$

$$= x \oplus y \oplus z$$

$$C = xz + yz + xy$$

$$= xz(y + y') + yz(x + x') + xy(z + z')$$

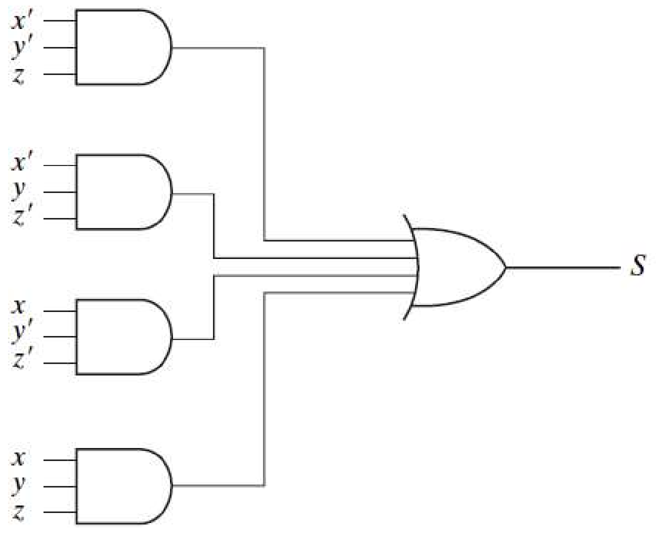
$$= xyz + xy'z + \cancel{xyz} + x'yz + \cancel{xyz} + xyz'$$

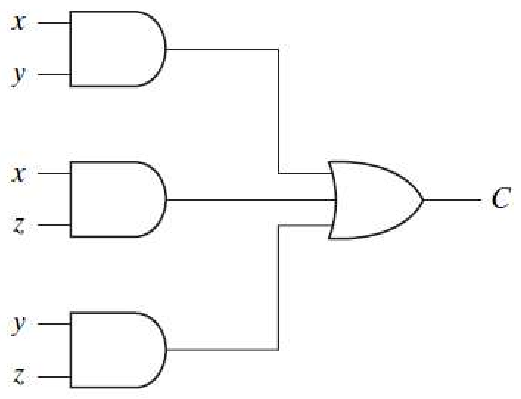
$$= xy\cancel{(z + z')} + z(\underline{xy' + x'y})$$

$$= xy + z(x \oplus y)$$

Full Adder - Circuit

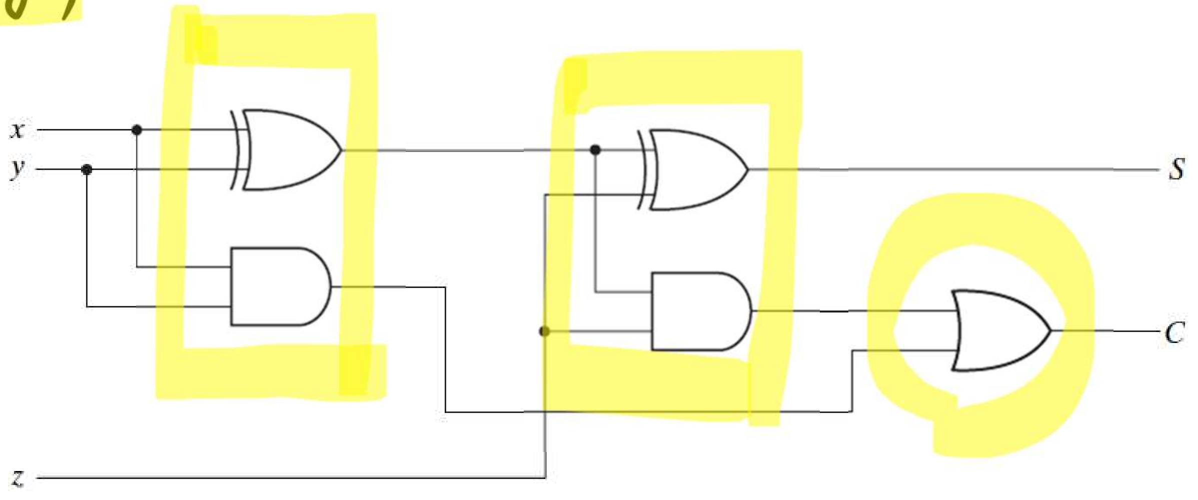
- Possible circuit implementations:





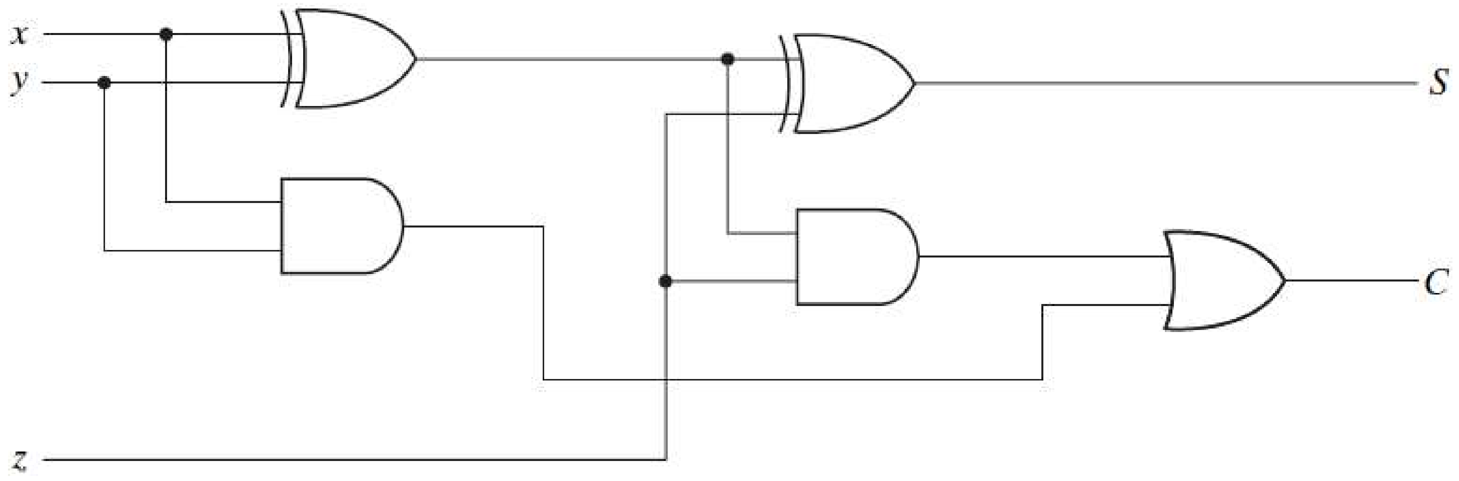
$S = (x \oplus y) \oplus z$

$C = xy + z(x \oplus y)$



Full Adder from Half Adders

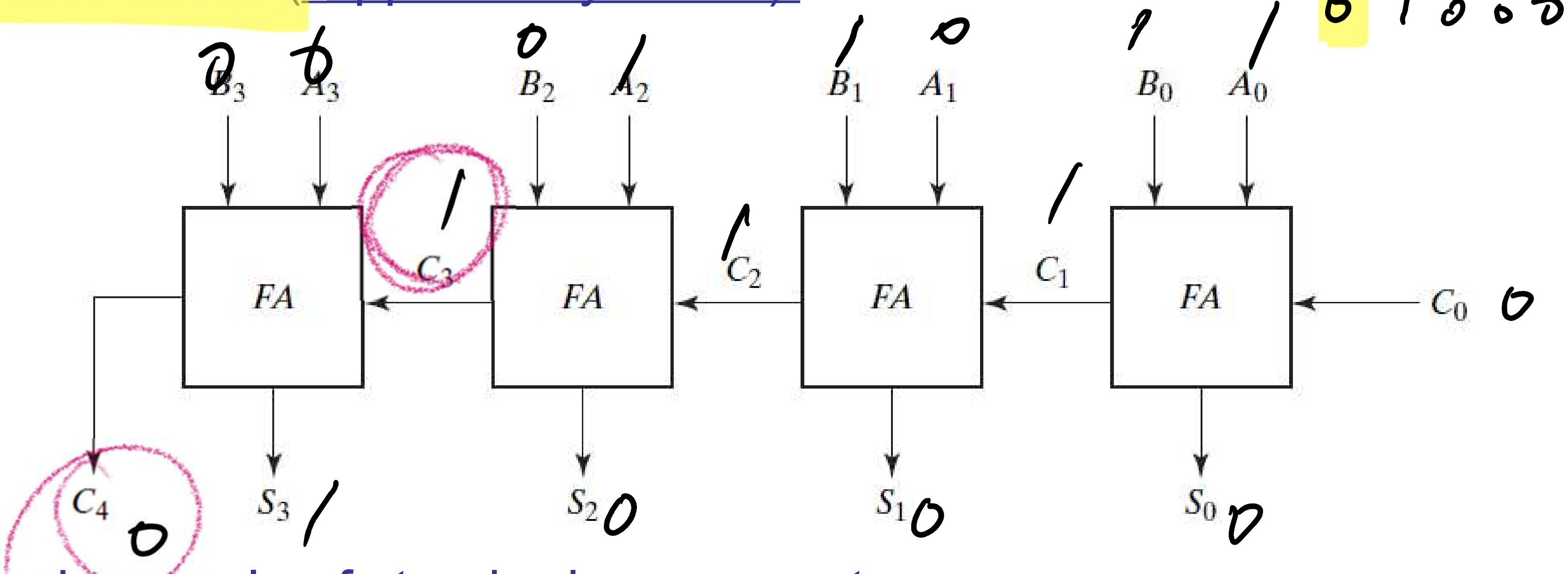
- How can two half adders make a full adder?
- Observations:
  - Three inputs x, y, z can be added in two steps
    - » $x+y+z = (x+y) + z$
  - What about the carry?
    - » Carry can occur when adding $x+y$ and when adding z
- Full adder: $S = x \oplus y \oplus z$, $C = x\,y + (x \oplus y)\,z$



Binary n-bit Adder

- How can we build an *n*-bit adder from full adders?
  - One adder for each bit (*n* total)
  - Connect carry to next adder's input
  - Output: sequence of sums and a final carry
- 4-bit adder circuit (Ripple Carry Adder):

$$
\begin{array}{rl}
 & 1\ 1\ 1\ 0 \\
A = A_3A_2A_1A_0 = & 0\ 1\ 0\ 1 \\
+\ B = B_3B_2B_1B_0 = & 0\ 0\ 1\ 1 \\
\hline
6 & 1\ 0\ 0\ 0
\end{array}
$$



- Classical example of standard components
  - Would require truth table with 2^9 entries!

Overflow

- *n*-bit addition can generate (*n+1*)-bit number
  - Called “overflow”
  - Needs to be detected by computer system
- How can we detect overflow in addition?
  - End carry
- Also necessary for signed numbers or subtraction
  - Most significant bit indicates sign
- If carry *into* sign position and *out* of sign position differ, then overflow
  - Result would be correct with extra position
  - Can be detected by XOR gate
  - Can be used as input carry for next adder circuit

c_3	c_4	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Overflow Conditions

- Overflow conditions
  - There is no overflow if signs are different (*pos* + *neg*, or *neg* + *pos*)
  - Overflow can happen only when both numbers have <u>same sign</u>, and
  - If carry *into* sign position and *out* of sign position differ
  - Example: 2's complement signed numbers wih *n* = 4 bits

```
             1  1  0                         0  0  0
  +6      0 110                  -6       1 010
  +7      0 111                  -7       1 001
 ------------------          ------------------
 +13    0 1 101             -13     1 0 011
```

- Result would be correct with extra position
- Detected by XOR gate (output =1 when inputs differ)
- Can be used as input carry for next adder circuit