Digital Logic Design

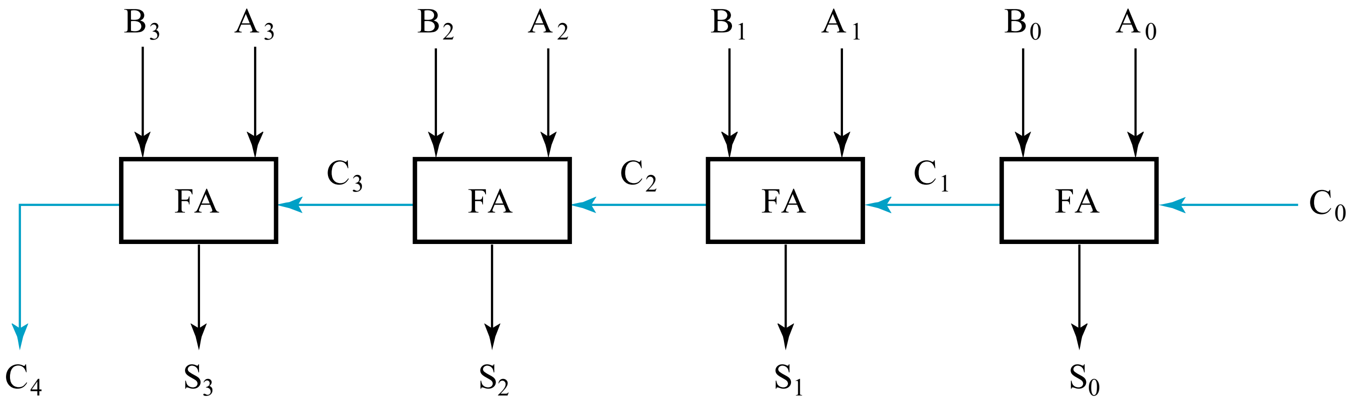
Hajar Falahati

Department of Computer Engineering
IRAN University of Science and Technology

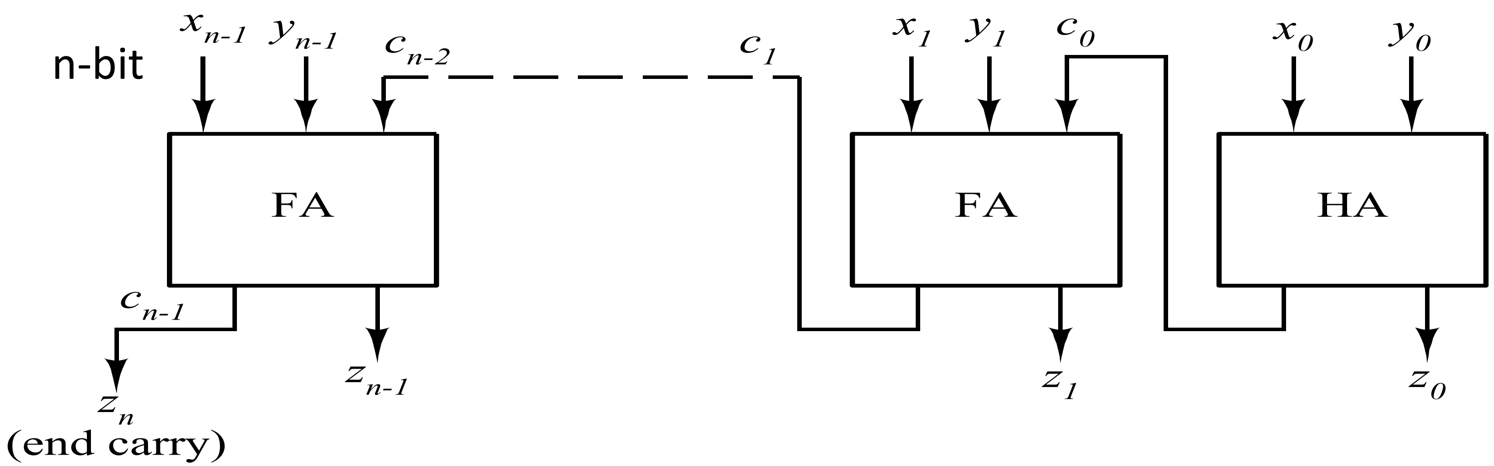
hfalahati@iust.ac.ir

Ripple-Carry-Adder (RCA)

4-bit



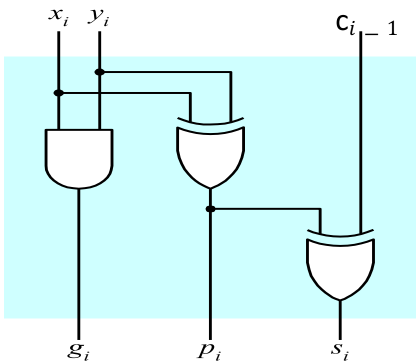
n-bit



Carry Look Ahead (CLA)

$s_i = pi \oplus c_{i-1}$

$c_i = g_i . + p_i \, . \, c_{i-1}$



$c_0 = g_0 + p_0 c_{in}$

$s_0 = p_0 \oplus c_{in}$

$c_1 = g_1 + p_1 c_0$

$\quad = g_1 + p_1 g_0$

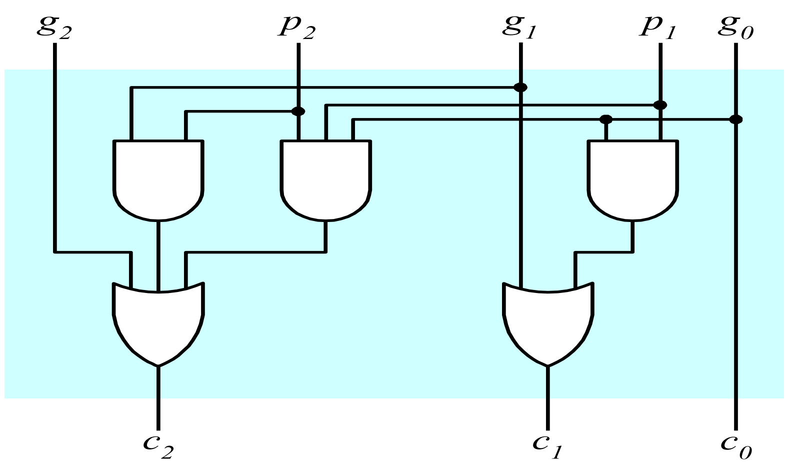
$s_1 = p_1 \oplus c_0$

$c_2 = g_2 + p_2 c_1$

$\quad = g_2 + p_2(g_1 + p_1 g_0)$

$\quad = g_2 + p_2 g_1 + p_2 p_1 g_0$

$s_2 = p_2 \oplus c_1$



Outline

- Subtractor
- Comparator

Subtractor

Recall: 2's Complement Subtraction

- $A = B - C$
  - $(A)_2 = (B)_2 + (-(C)_2) = (B)_2 + [C]_2 = (B)_2 + 2^n - (C)_2 = 2^n + (B - C)_2$
  - If $B \geq C$
    - $A \geq 2^n$ and the carry is discarded
    - $(A)_2 = (B)_2 + [C]|_{\text{carry discarded}}$
  - If $B < C$
    - $A = 2^n - (C - B)_2 = [C - B]_2$ or $A = -(C - B)_2$ (no carry in this case)
    - No overflow for Case 2.

- $(A)_2 = (B)_2 + (-(C)_2) = (B)_2 + [C]_2 = (B)_2 + (C)'_2 + 1$

Recall: Sample 1

$$
\begin{array}{r}
01001101 \\
-\ 00111010 \\
\hline
\end{array}
\qquad\qquad
\begin{array}{r}
01001101 \\
+\ 11000110 \\
\hline
10\,0010011
\end{array}
$$

Carry = 1=> Result is OK

Recall: Sample 2

$$
\begin{array}{r}
\;0\,1\,0\,0\,0\,0\,1\,1 \\
-\;0\,1\,0\,1\,0\,1\,0\,0 \\
\hline
\end{array}
\qquad\qquad
\begin{array}{r}
\;0\,1\,0\,0\,0\,0\,1\,1 \\
+\;1\,0\,1\,0\,1\,1\,0\,0 \\
\hline
1\,1\,1\,0\,1\,1\,1\,1
\end{array}
$$

Carry = 0 => Result is not OK

$[1\ 1\ 1\ 0\ 1\ 1\ 1\ 1]_2 = 00010001$

Result = - (00010001)

2’s Complement Subtractor

- Inputs
  - 2 numbers
  - x_i, y_i

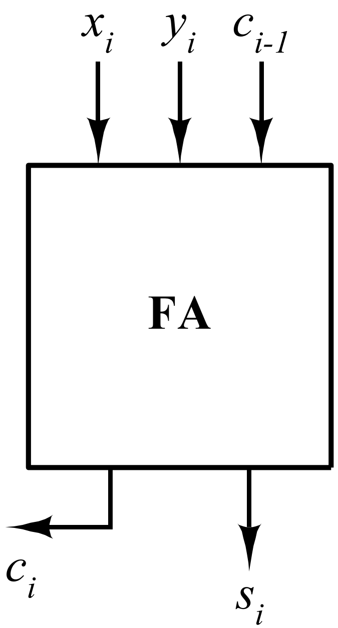
$$\begin{array}{r} x_i \\ -\ \ y_i \\ \hline c_i\ s_i \end{array}$$

- Output
  - 1 number
  - 1 borrow

- Functionality
  - Subtract two numbers

2’s Complement Subtractor Vs. Full Adder

- $C_{i-1} = 0$
  - Input: x_i , y_i
  - $S_i = x_i + y_i$
- $C_{i-1} = 1$
  - Input: x_i , $(y_i)'$
  - $S_i = x_i - y_i$
- Functionality
  - Subtract two numbers



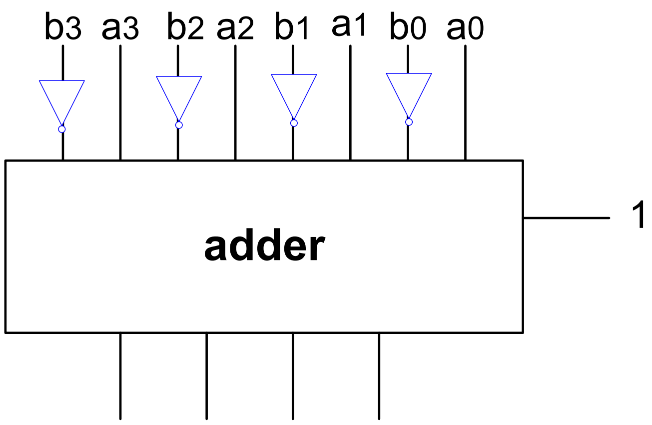
4-bit
2's Complement Subtractor

- Inputs
  - 2 4-bit numbers
  - a, b
- Output
  - a-b

4-bit Subtractor: Realization

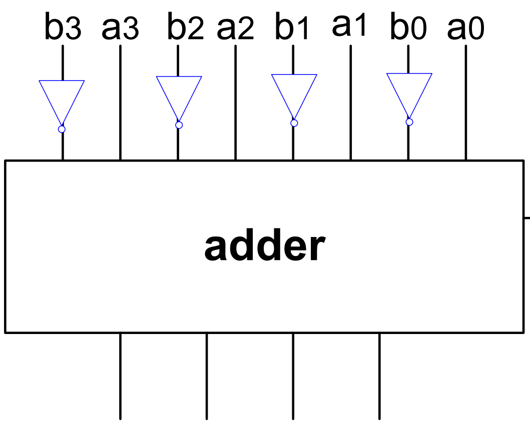
- Inputs
  - 2 4-bit numbers
  - a, b

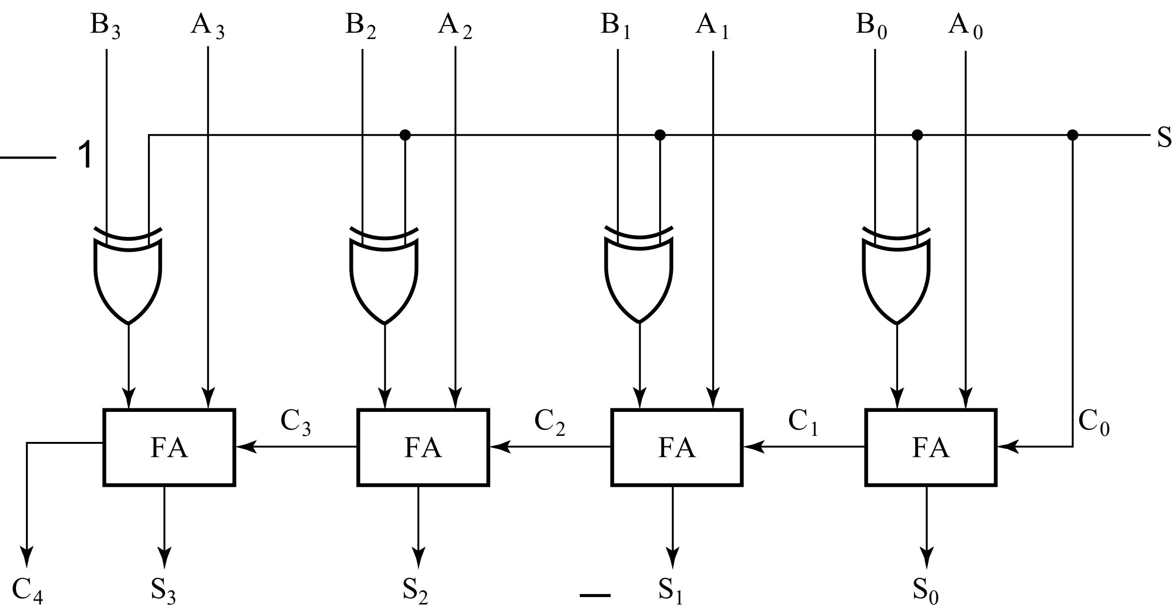
- Output
  - a-b



$$a - b = a + (-b) = a + (\bar{b} + 1)$$

4-bit Subtractor: Detailed Realization





$$a - b = a + (-b) = a + (\bar{b} + 1)$$

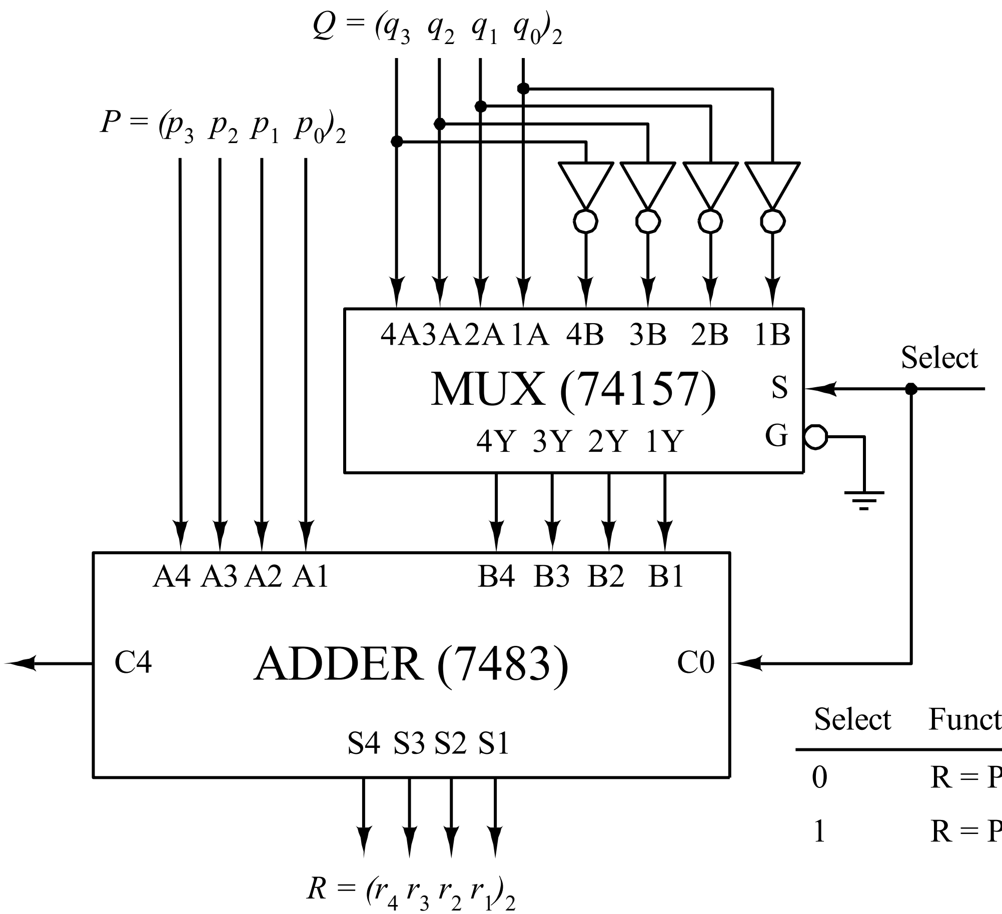
2's Complement Adder/ Subtractor

- Inputs
  - 2 4-bit numbers
  - **Select**

- Output
  - **Select** ==0 => Add
  - **Select** ==1 => Sub

Adder/ Subtractor: Realization

- Inputs
  - 2 4-bit numbers
  - Select
- Output
  - Select ==0 => Add
  - Select ==1 => Sub



Select	Function
0	$R = P + Q$
1	$R = P + \overline{Q} + 1$

Overflow

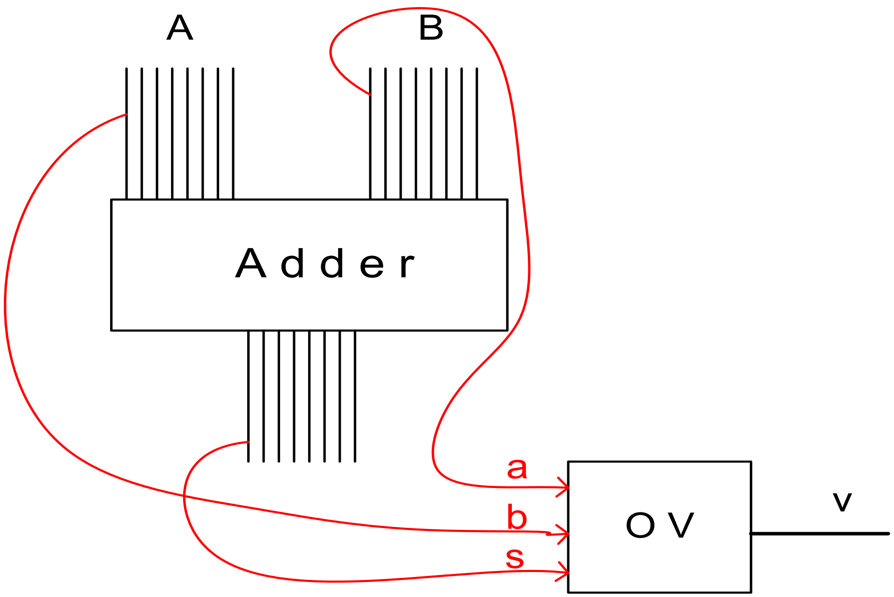
```
  1 1 0 0 0 0 1 1
+
  1 0 1 0 1 1 0 0
-----------------
1 0 1 1 0 1 1 1 1
```

```
  0 0 1 1 0 1 0 0
+
  0 1 1 0 1 0 0 0
-----------------
  1 0 0 1 1 1 0 0
```

Case	Carry	Sign Bit	Condition	Overflow ?
$B + C$	0 0	0 1	$B + C \leq 2^{n-1} - 1$ $B + C > 2^{n-1} - 1$	No Yes
$B - C$	1 0	0 1	$B \leq C$ $B > C$	No No
$-B - C$	1 1	1 0	$-(B + C) \geq -2^{n-1}$ $-(B + C) < -2^{n-1}$	No Yes

Overflow Detection

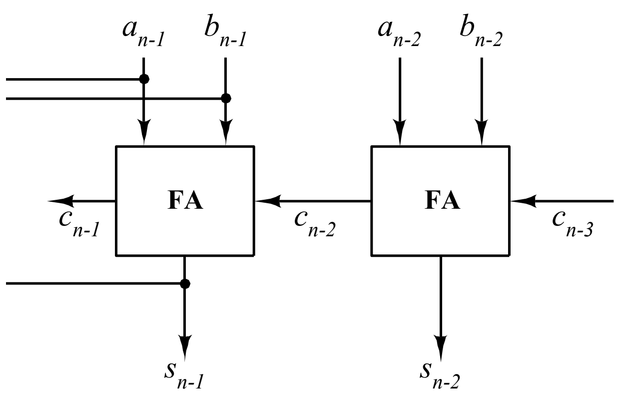
- Design a circuit to detect overflow



Overflow Detection: TT

Case	Carry	Sign Bit	Condition	Overflow ?
B + C	0 0	0 1	$B + C \le 2^{n-1} - 1$ $B + C > 2^{n-1} - 1$	No Yes
B - C	1 0	0 1	$B \le C$ $B > C$	No No
-B - C	1 1	1 0	$-(B + C) \ge -2^{n-1}$ $-(B + C) < -2^{n-1}$	No Yes

a_{n-1}	b_{n-1}	c_{n-2}	c_{n-1}	s_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0



Overflow Detection: Logic Equation

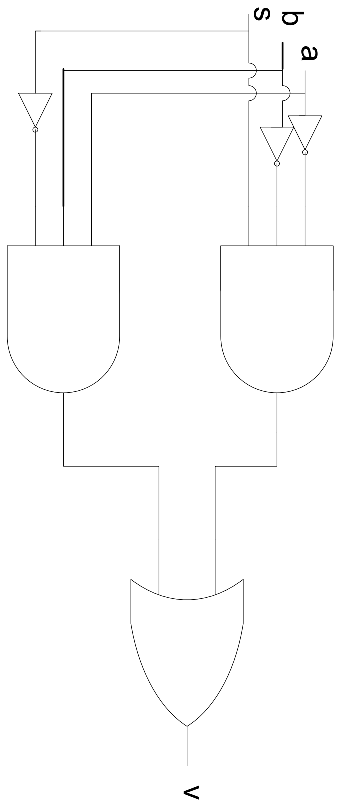
Case	Carry	Sign Bit	Condition	Overflow ?
B + C	0 0	0 1	$B + C \leq 2^{n-1} - 1$ $B + C > 2^{n-1} - 1$	No Yes
B - C	1 0	0 1	$B \leq C$ $B > C$	No No
-B - C	1 1	1 0	$-(B + C) \geq -2^{n-1}$ $-(B + C) < -2^{n-1}$	No Yes

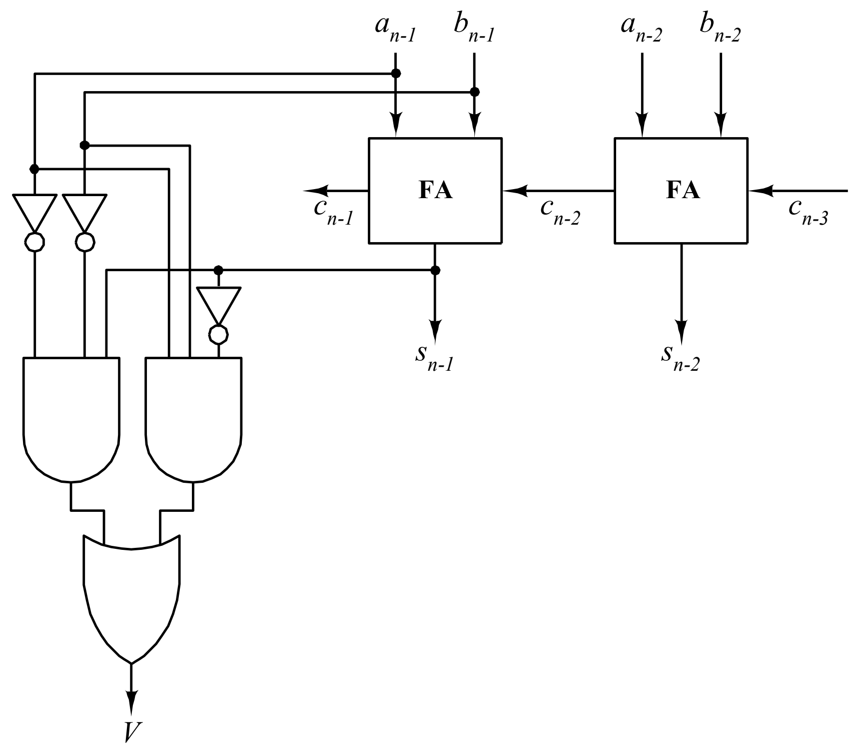
a_{n-1}	b_{n-1}	c_{n-2}	c_{n-1}	s_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

$V = a_{n-1} . b_{n-1} . (s_{n-1})' + (a_{n-1})' . (b_{n-1})' . s_{n-1}$

Overflow Detection: Realization

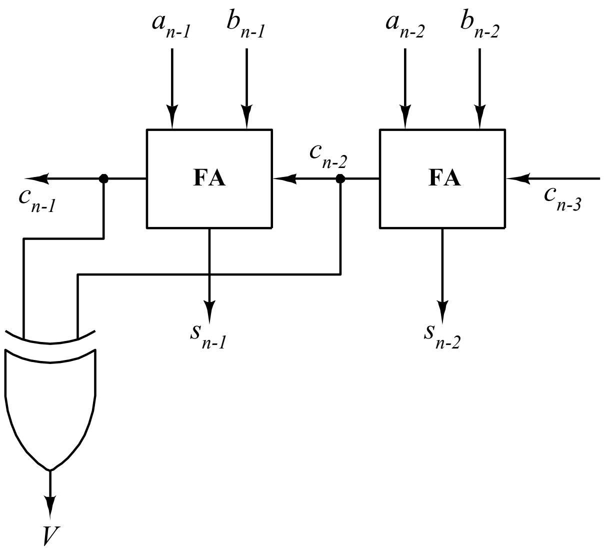
$V = a_{n-1} . b_{n-1} . (s_{n-1})' + (a_{n-1})' . (b_{n-1})' . s_{n-1}$





Overflow Detection: Realization 2

a_{n-1}	b_{n-1}	c_{n-2}	c_{n-1}	s_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0



$V = a_{n-1} . b_{n-1} . (s_{n-1})' + (a_{n-1})' . (b_{n-1})' . s_{n-1}$

$V = c_{n-2} \oplus c_{n-1}$

Comparator

Comparator

- Inputs
  - 2 numbers
  - x_i, y_i

- Output
  - 3 1-bit
  - Equal
  - Greater than
  - Lower than

Comparator: Check Equality

- Design a circuit to check the equality of two numbers

Comparator: Check Inequality

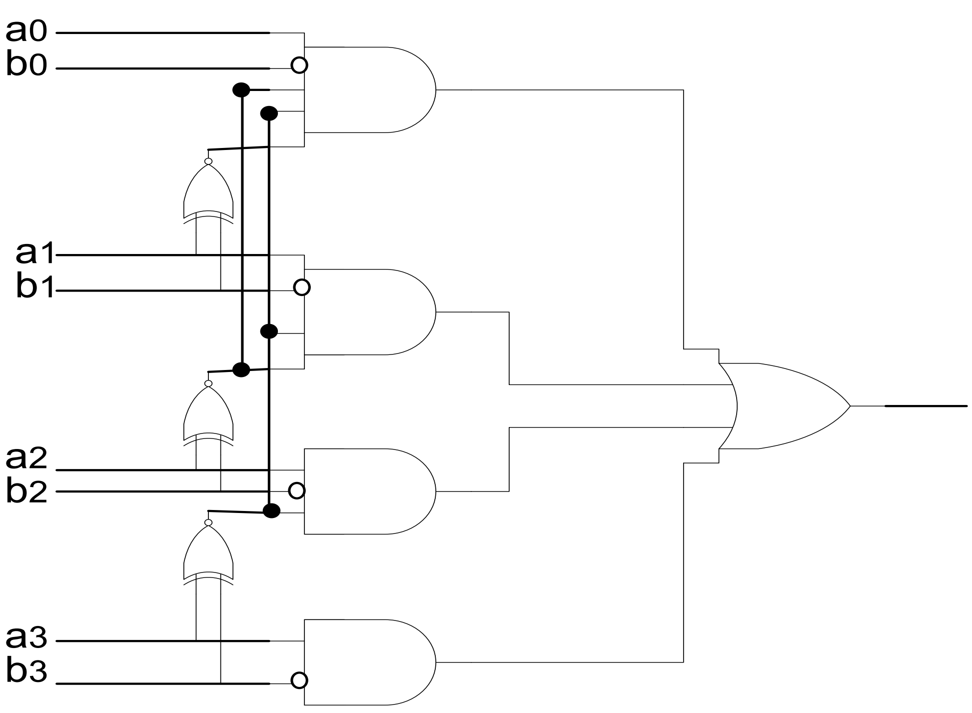
- Design a circuit to check a>b

Check Inequality: Realization

- Check MSB of a
  - If a3 = 1 and b3 = 0 => a > b
  - If a3 = 0 and b3 = 1=> a < b

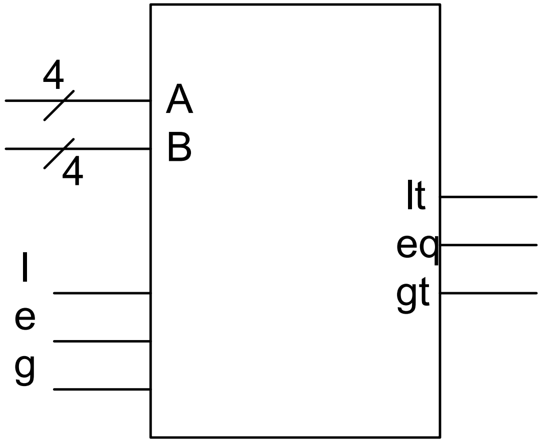
  - Else If a2 = 1 and b2 = 0 => a>b
  - Else If a2 = 0 and b2 = 1 => a < b
  - (a3 = b3)

  - Else If a1 = 1 and b1 = 0 => a>b
  - Else If a1 = 0 and b1 = 1 => a < b
  - (a3 = b3 , a2 = b2)



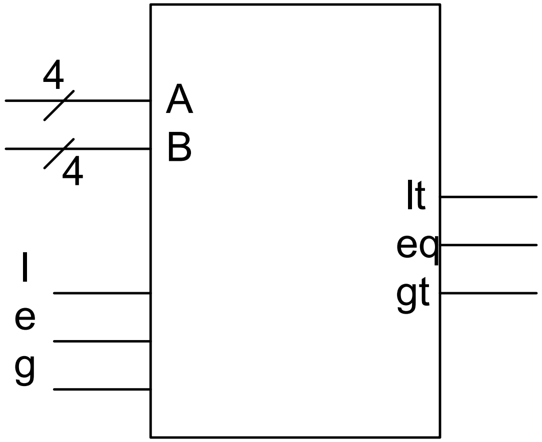
Comparator Circuit

- Design a comparator circuit (7485)
- Inputs
  - 2 4-bit numbers
  - l, e and g (cascading)
- Outputs
  - If (A > B) => lt=0, eq=0, gt=1
  - if (A < B) => lt=1, eq=0, gt=0
  - if (A = B) => lt=l, eq=e, gt=g



Comparator: 16 bit

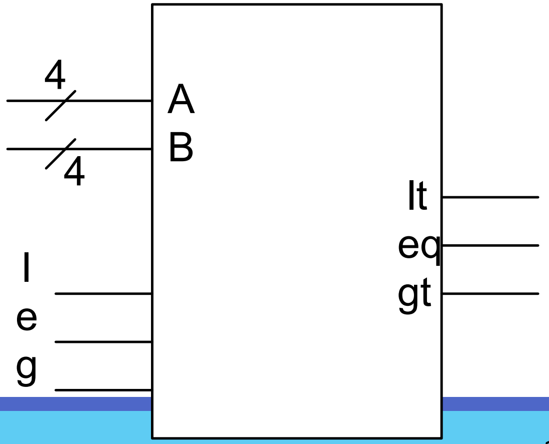
- Compare two 16-bit numbers



16-bit Comparator: Steps

1. Compare the **4 most significant bits** of the two inputs
   - **'A'** is **less** than **'B'** if its 4 MSB are **smaller** => STOP
   - **'A'** is **greater** than **'B'** if its 4 MSB are **larger** => STOP
   - If these 4 MSB are the **last ones**, two numbers are equal => STOP
   - If they are **equal** the **result** can be found in **this first stage** => GO Step2

2. Comparator will **recursively** do the same comparison on **less significant bits four by four**



Comparator: Signed numbers

- Design a circuit to compare two signed numbers

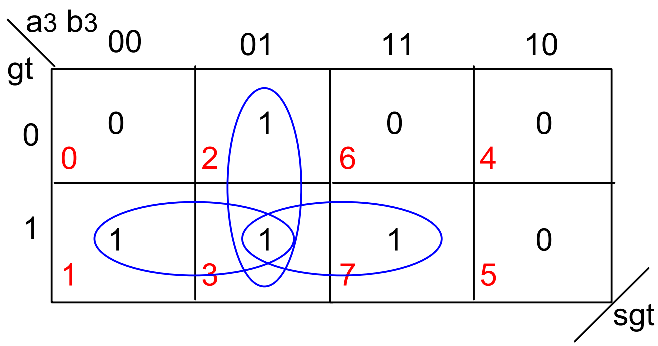
Signed Numbers (A > B)

- A > B
  - If **A** is **positive** and **B** is **negative => A > B**
  - If **Both** are **positive** or **negative => consider gt**

- Example: compare +3 and +2 by a 4 bit unsigned comparator
  - 3 = $(0010)_2$
  - 2 = $(0011)_2$
  - In unsigned comparator => 3 > 2

- Example: compare -2 and -3 by a 4 bit unsigned comparator
  - $(-2)_{2s} = 2^n - 2 \Rightarrow (1110)_{2s}$
  - $(-3)_{2s} = 2^n - 3 \Rightarrow (1101)_{2s}$
  - In unsigned comparator => $2^n - 2 > 2^n - 3$

Signed Numbers (A > B): K-map



$$sgt = \overline{a_3}.b_3 + \overline{a_3}.gt + b_3.gt$$

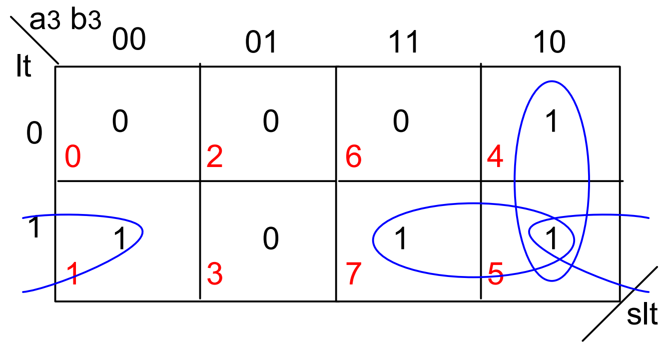
Signed Numbers (A < B)

- A < B
  - If **A** is **negative** and **B** is **positive => A < B**
  - If **Both** are **positive** or **negative => consider It**

- Example: compare +2 and +3 by a 4 bit unsigned comparator
  - 2 = $(0010)_2$
  - 3 = $(0011)_2$
  - In unsigned comparator => 2 < 3

- Example: compare -3 and -2 by a 4 bit unsigned comparator
  - $(-3)_{2s} = 2^n - 3 \Rightarrow (1101)_{2s}$
  - $(-2)_{2s} = 2^n - 2 \Rightarrow (1110)_{2s}$
  - In unsigned comparator => $2^n - 3 < 2^n - 2$

Signed Numbers (A < B): K-map



$$slt = a_3.\overline{b_3} + lt.a_3 + lt.\overline{b_3}$$

Sample Logic1: BCD Adder

- Design a circuit to add two BCD numbers

BCD Adder: Truth Table

- Compare **BCD sum** with **binary sum**

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder Vs. Binary Adder

- When are they different?

Binary Sum K	Z_8	Z_4	Z_2	Z_1	BCD Sum C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder Vs. Binary Adder: Difference

- When are they different?
  - $F = K + Z_8 Z_4 + Z_8 Z_2$

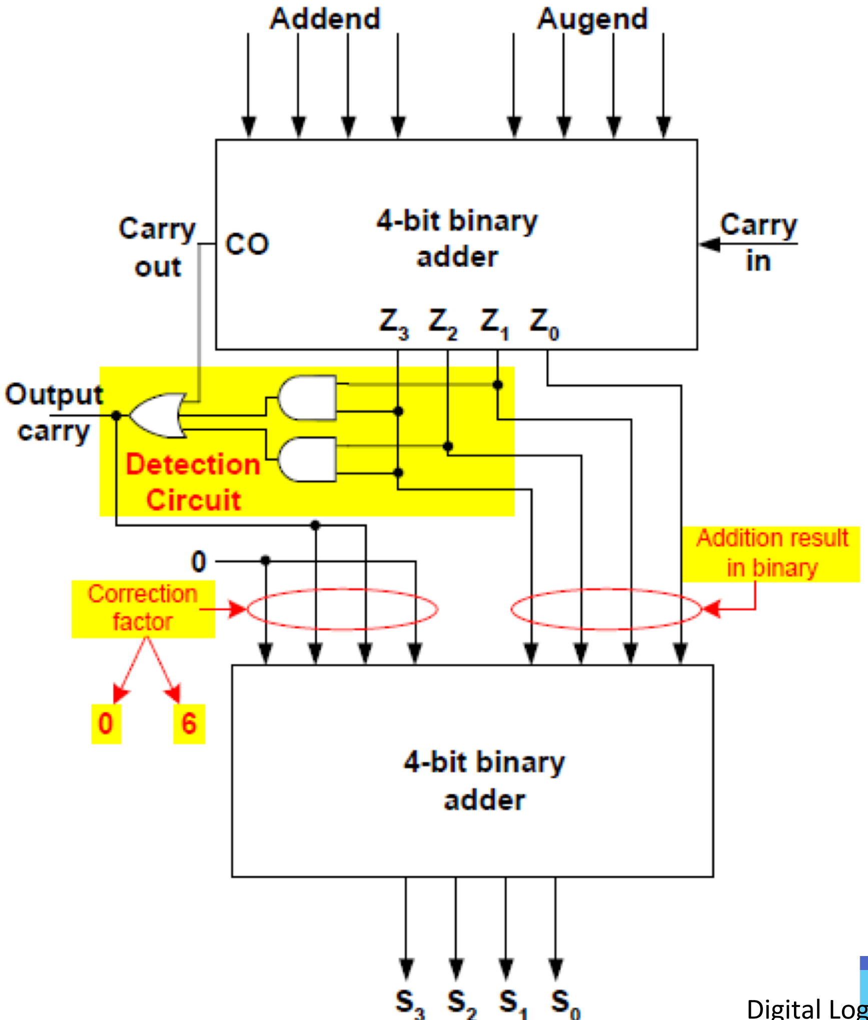
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder: How to correct?

- When are they different?
  - $F = K + Z_8 Z_4 + Z_8 Z_2$
- How to correct it?
  - Add by 6 (0110)

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

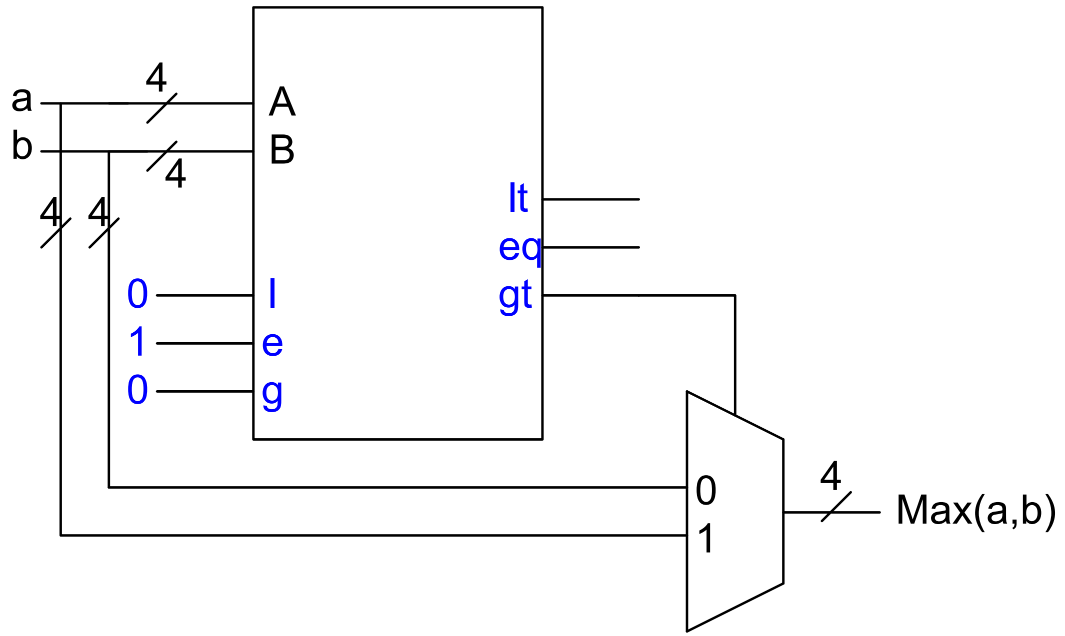
BCD Adder: Realization



Sample Logic2: Max Logic

- Design a circuit to find the maximum number

Max Logic



Sample Logic3: Multiplier

- Design a circuit to multiply two 2-bit numbers

$$X = \sum_{i=0}^{M-1} X_i \, 2^i$$

Multiplicand

$$Y = \sum_{i=0}^{N-1} Y_i \, 2^i$$

Multiplier

Product $\quad Z = X * Y$

Multiplier: Recall

$$\begin{array}{rcl} 1100 & : & 12_{10} \\ \underline{0101} & : & 5_{10} \end{array}$$

$$\begin{array}{rcl} 1100 & : & 12_{10} \\ \underline{0101} & : & 5_{10} \\ 1100 & & \end{array}$$

$$\begin{array}{rcl} 1100 & : & 12_{10} \\ \underline{0101} & : & 5_{10} \\ 1100 & & \\ 0000 & & \end{array}$$

$$\begin{array}{rcl} 1100 & : & 12_{10} \\ \underline{0101} & : & 5_{10} \\ 1100 & & \\ 0000 & & \\ 1100 & & \end{array}$$

$$\begin{array}{rcl} 1100 & : & 12_{10} \\ \underline{0101} & : & 5_{10} \\ 1100 & & \\ 0000 & & \\ 1100 & & \\ \underline{0000} & & \end{array}$$

$$\begin{array}{rcl} 1100 & : & 12_{10} \\ \underline{0101} & : & 5_{10} \\ 1100 & & \\ 0000 & & \\ 1100 & & \\ \underline{0000} & & \\ 00111100 & : & 60_{10} \end{array}$$

Multiplier: Partial Products

$$X = \sum_{i=0}^{M-1} X_i \, 2^i$$

Multiplicand

$$Y = \sum_{i=0}^{N-1} Y_i \, 2^i$$

Multiplier

$$Z = X * Y$$

$$= \sum_{i=0}^{N-1} \left(\sum_{j=0}^{M-1} X_i Y_j 2^{i+j} \right)$$

Partial products

```
     1100  :  12₁₀     multiplicand
     0101  :  5₁₀      multiplier
     ----
     1100             ┐
    0000              │ partial
   1100               │ products
  0000                ┘
 --------
 00111100  :  60₁₀     product
```

Multiplier: Binary Representation

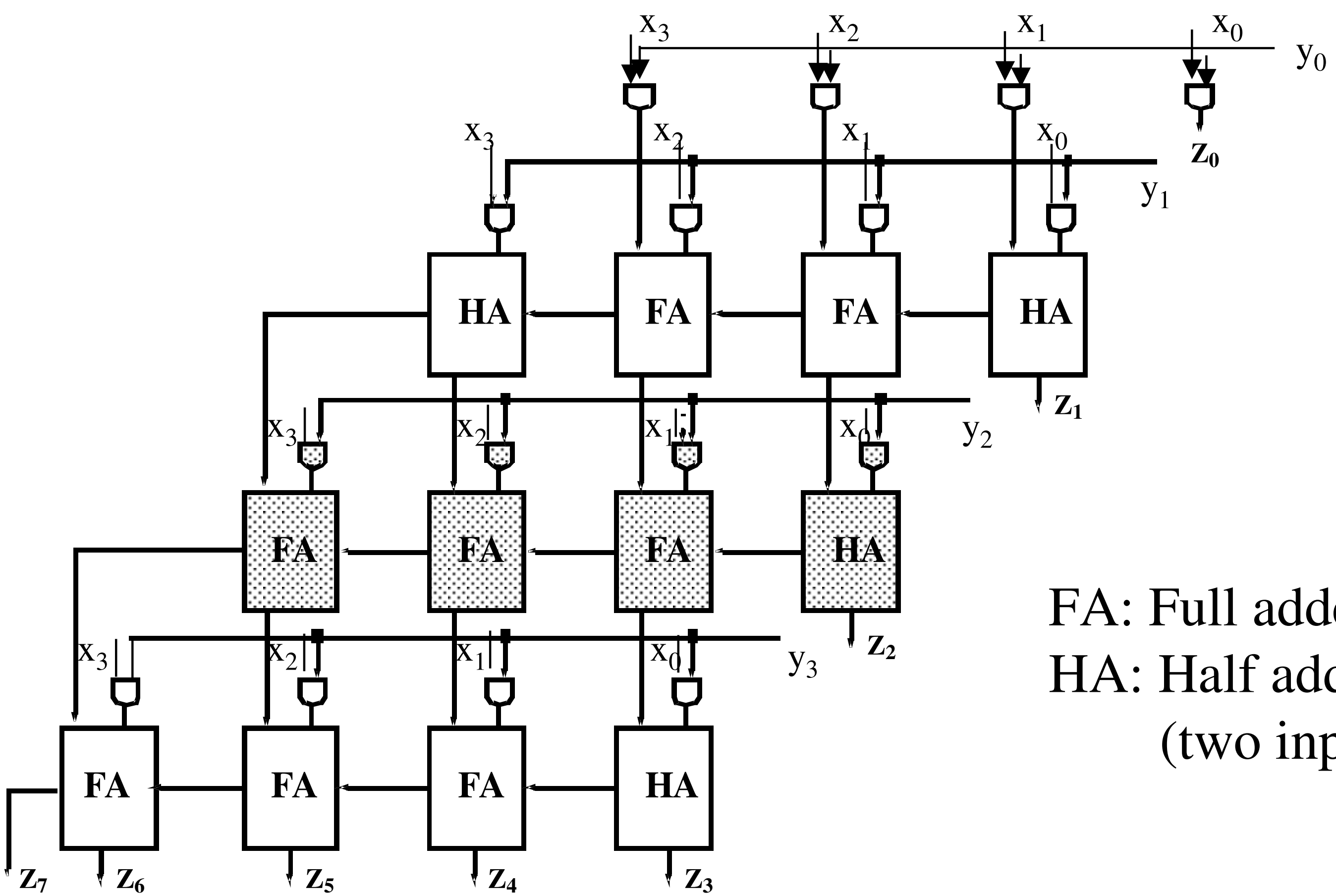
Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$

Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$

Product: $$P = \left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$

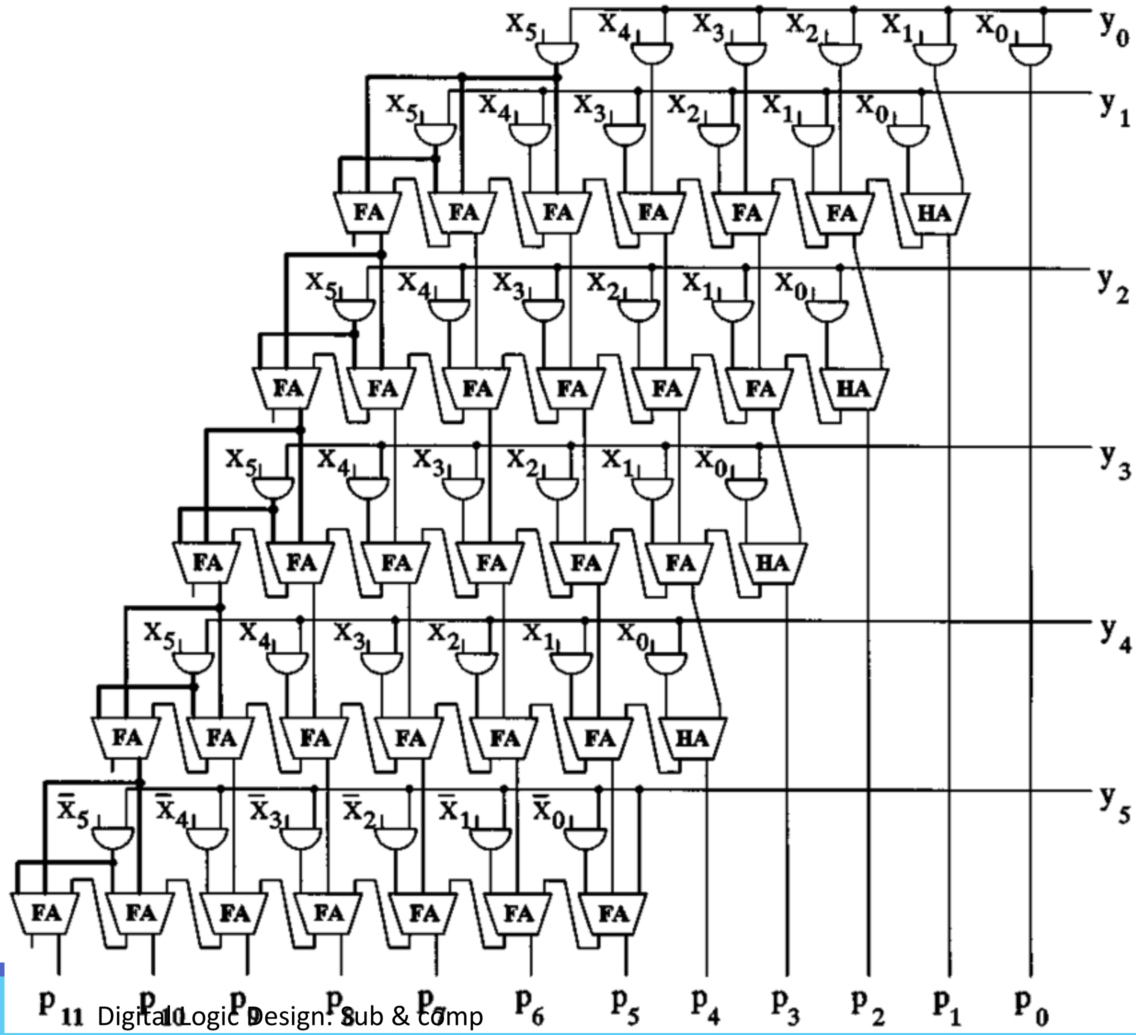
						y_5	y_4	y_3	y_2	y_1	y_0	multiplicand
						x_5	x_4	x_3	x_2	x_1	x_0	multiplier
						x_0y_5	x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0	
					x_1y_5	x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0		
				x_2y_5	x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0			partial products
			x_3y_5	x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0				
		x_4y_5	x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0					
	x_5y_5	x_5y_4	x_5y_3	x_5y_2	x_5y_1	x_5y_0						
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	product

4-bit Multiplier: Realization



FA: Full adder
HA: Half adder
(two inputs)

6-bit Multiplier: Realization



Thank You