School of Engineering and Computer Science

Mathematics Applied to Digital Engineering-2

(Combinational Logic)

Kamel ATTAR
kamel.attar@epita.fr

Week **#6** — **7** ✦ Mars/2024 ✦

Overflow Detection Circuit for 2's Complement Addition
Overflow Detection Circuit for Adder/Subtractor Control

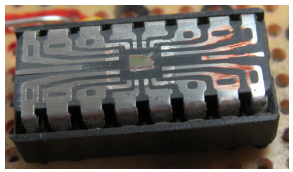
5 Binary Comparators

Equality Comparators
Magnitude Comparators

★ Integrated Circuits ★

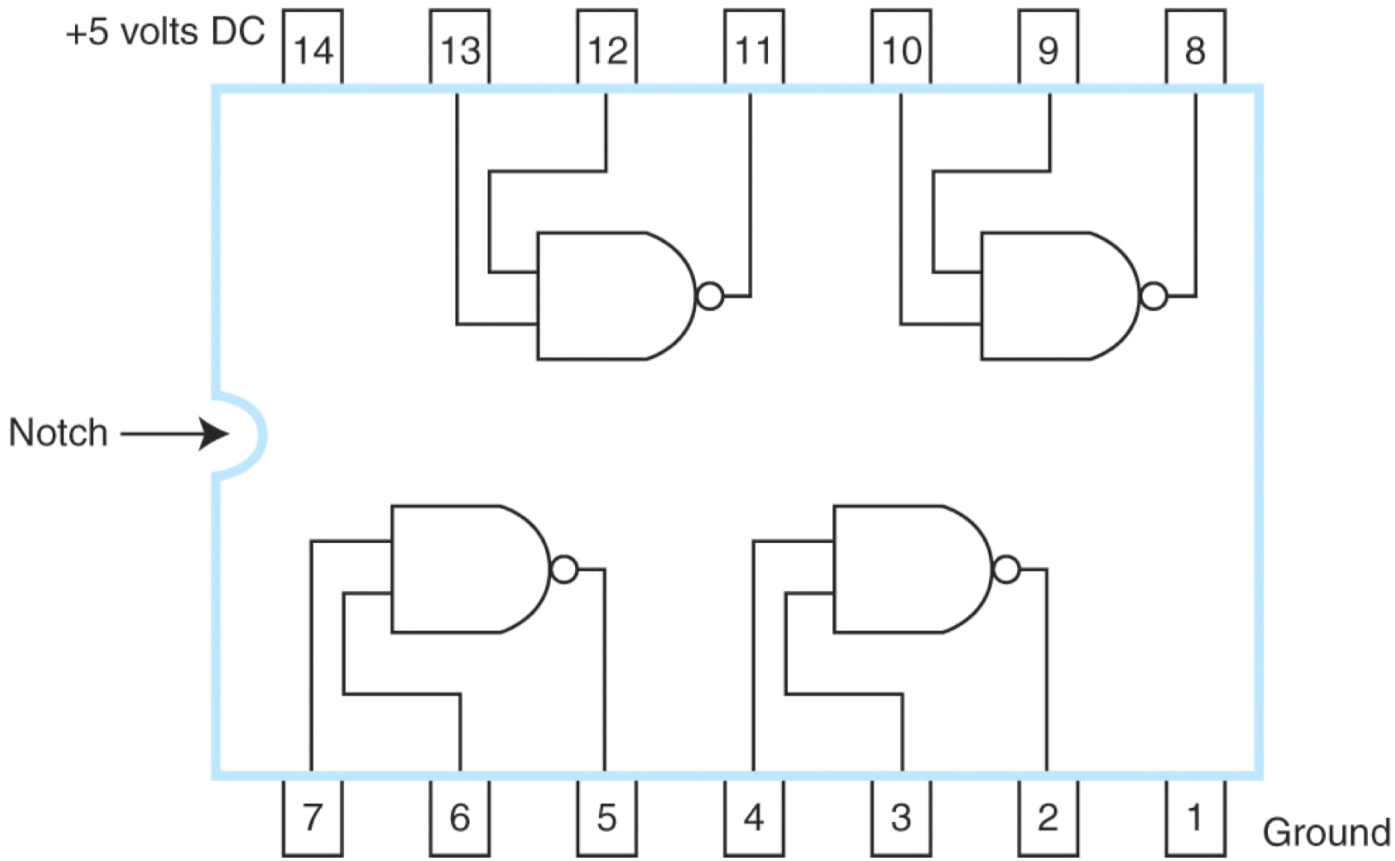
Integrated Circuits

- Every computer is built using collections of gates that are all connected by way of wires acting as signal gateway.
- Gates are not sold individually; they are sold in units called integrated circuits (ICs).
- A chip (a small silicon semiconductor crystal) is a small electronic device consisting of the necessary electronic components (transistors, resistors, and capacitors) to implement various gates.
- The first IC were called SSI chips and contained up to 100 electronic components per chip.
- We now have ULSI (ultra large-scale integration) with more than 1 million electronic components per chip.



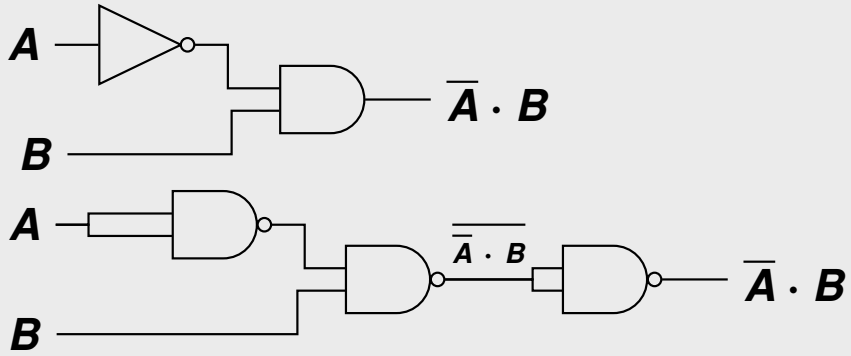
Integrated Circuits

A simple SSI Integrated Circuits



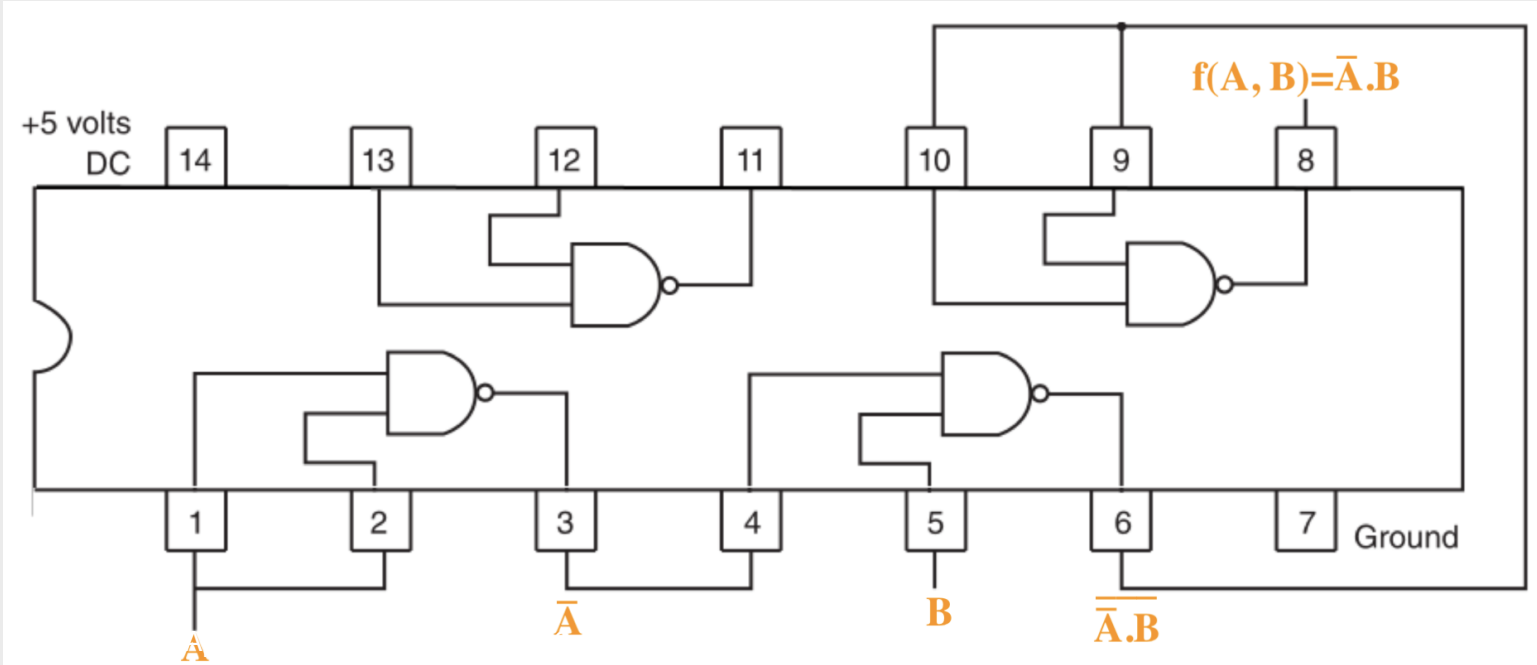
Example

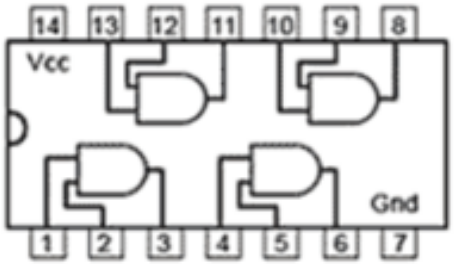
The boolean circuit $F(A, B) = \overline{A}B$



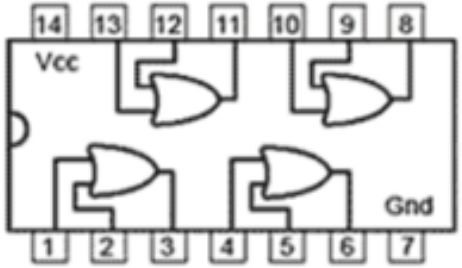
Can be rendered using only **NAND** gates as:

So we can wire the pre-packaged circuit to implement our function:

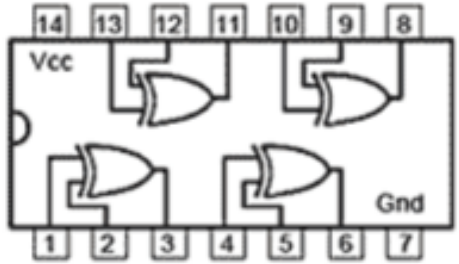




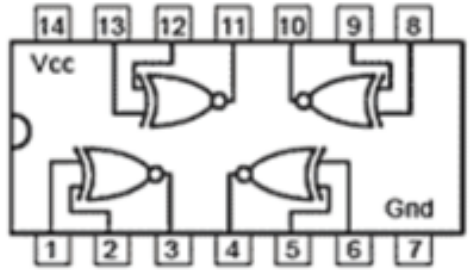
7408 Quad 2 input AND Gates



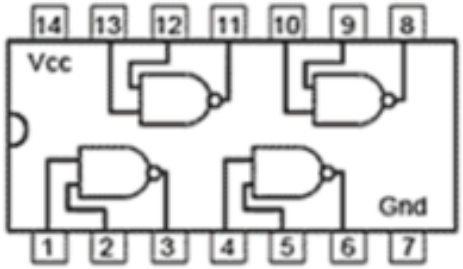
7432 Quad 2 input OR Gates



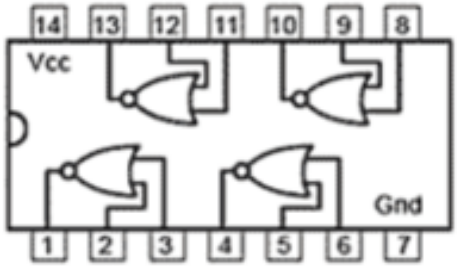
7486 Quad 2 input XOR Gates



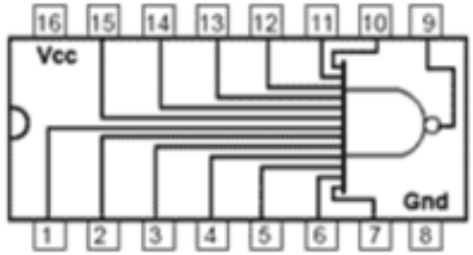
747266 Quad 2 input XNOR Gates



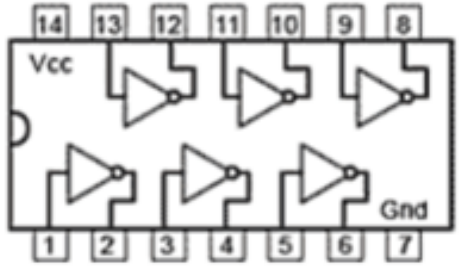
7400 Quad 2 input NAND Gates



7402 Quad 2 input NOR Gates



74133 Single 13 input NAND Gate



7404 Hex NOT Gates (Inverters)

- In digital electronics an adder is a logic circuit that implements addition of numbers. So we will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions.

- In digital electronics an adder is a logic circuit that implements addition of numbers. So we will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions.

- We will build up the circuitry to do this addition from some component circuits.

- In digital electronics an adder is a logic circuit that implements addition of numbers. So we will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions.

- We will build up the circuitry to do this addition from some component circuits.

- In this part, we will build a circuit that can be used to find $x + y$, where x and y are two bits.
  - The input to our circuit will be x and y, because these each have the value **0** or the value **1**.
  - The output will consist of two bits, namely, s and c, where s is the sum bit and c is the carry bit.

- In digital electronics an adder is a logic circuit that implements addition of numbers. So we will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions.

- We will build up the circuitry to do this addition from some component circuits.

- In this part, we will build a circuit that can be used to find $x + y$, where x and y are two bits.
  - The input to our circuit will be x and y, because these each have the value **0** or the value **1**.
  - The output will consist of two bits, namely, s and c, where s is the sum bit and c is the carry bit.
- This circuit is called a multiple output circuit because it has more than one output. The circuit that we are designing is called the **half adder**, because it adds two bits, without considering a carry from a previous addition.

- In digital electronics an adder is a logic circuit that implements addition of numbers. So we will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions.

- We will build up the circuitry to do this addition from some component circuits.

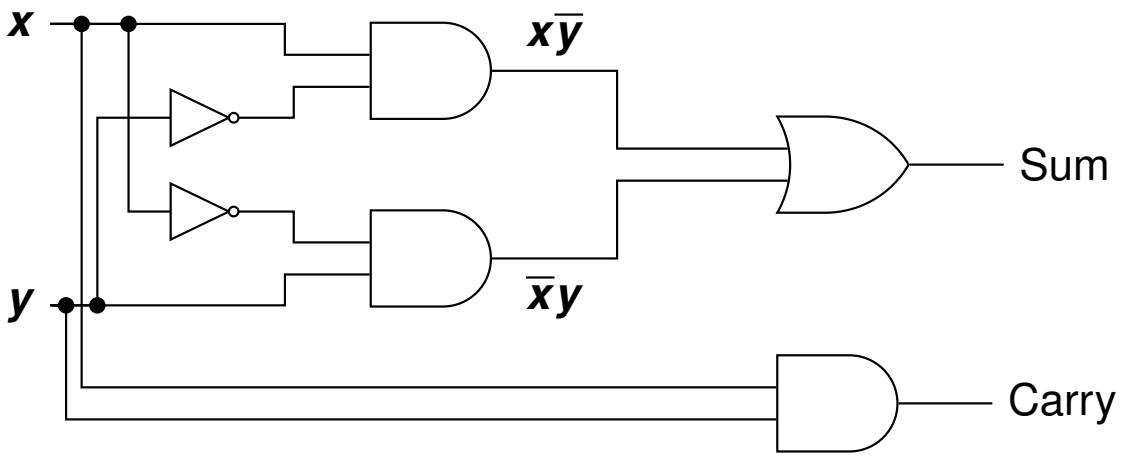
- In this part, we will build a circuit that can be used to find $x + y$, where x and y are two bits.
  - The input to our circuit will be x and y, because these each have the value **0** or the value **1**.
  - The output will consist of two bits, namely, s and c, where s is the sum bit and c is the carry bit.
- This circuit is called a multiple output circuit because it has more than one output. The circuit that we are designing is called the **half adder**, because it adds two bits, without considering a carry from a previous addition.

x	+	y	=	*Carry*	*Sum*
0	+	0	=	0	0
0	+	1	=	0	1
1	+	0	=	0	1
1	+	1	=	1	0

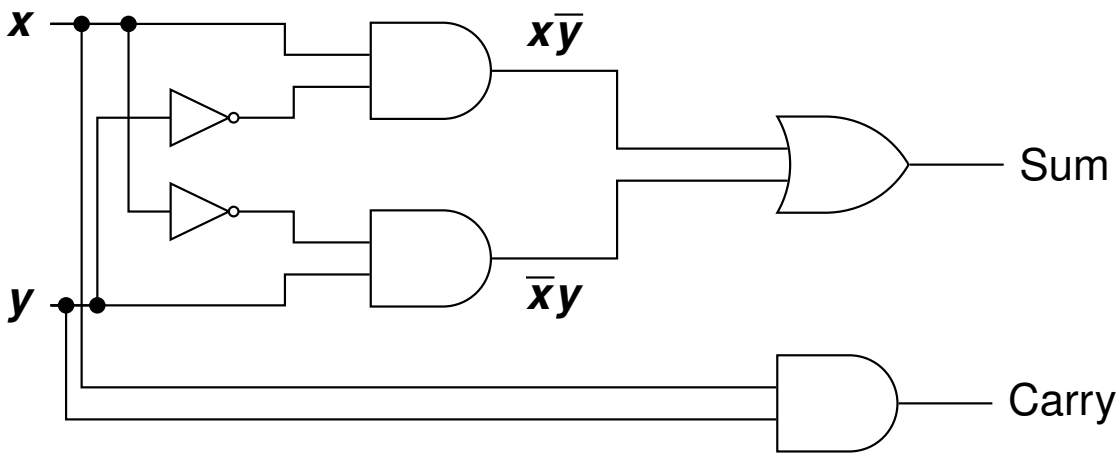
- From this table we see that

$$c = xy \quad \text{and that} \quad s = x\overline{y} + \overline{x}y = x \oplus y.$$

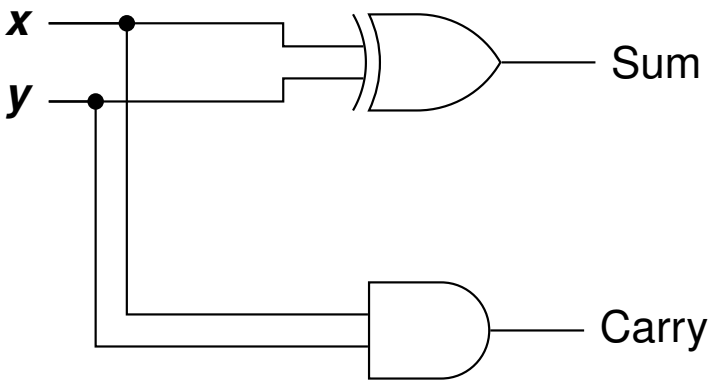
- Hence, the circuit shown below computes the sum bit ***s*** and the carry bit ***c*** from the bits ***x*** and ***y***.

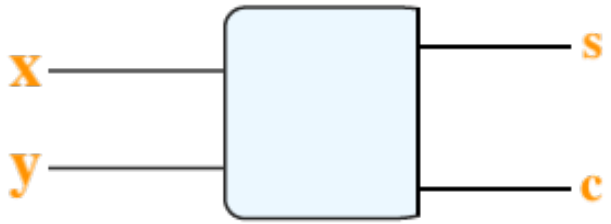


- Hence, the circuit shown below computes the sum bit ***s*** and the carry bit ***c*** from the bits ***x*** and ***y***.



- **XOR** is applied to both inputs to produce sum and AND gate is applied to both inputs to produce carry





- **Half adder** is used to make full adder as a **full adder** requires **3** inputs, the third input being an input carry i.e. we will be able to cascade the carry bit from one adder to the other

- We use the **full adder** to compute the sum bit and the carry bit when two bits and a carry are added.
- The full adder can take three **1**-bit inputs and them to create a **2**-bit value.

The truth table includes five columns with **three inputs** and **two outputs**.

1. input x
2. input y
3. input **carry IN** c_i for the current column
4. output resulting **carry OUT** (carry-over), generated for the next column
5. output the resulting SUM s.

Inputs			outputs	
x	y	c_i	c_{i+1}	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

From this table we see that

$$c_{i+1} = ?$$

$$s = ?$$

- **Half adder** is used to make full adder as a **full adder** requires **3** inputs, the third input being an input carry i.e. we will be able to cascade the carry bit from one adder to the other

- We use the **full adder** to compute the sum bit and the carry bit when two bits and a carry are added.
- The full adder can take three **1**-bit inputs and them to create a **2**-bit value.

The truth table includes five columns with **three inputs** and **two outputs**.

1. input x
2. input y
3. input **carry IN** c_i for the current column
4. output resulting **carry OUT** (carry-over), generated for the next column
5. output the resulting SUM s.

Inputs			outputs	
x	y	c_i	c_{i+1}	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

From this table we see that

$$c_{i+1} = \overline{x}yc_i + x\overline{y}c_i + xy\overline{c_i} + xyc_i$$

$$s = \overline{x}\,\overline{y}c_i + \overline{x}y\overline{c_i} + x\overline{y}\,\overline{c_i} + xyc_i$$

- **Half adder** is used to make full adder as a **full adder** requires **3** inputs, the third input being an input carry i.e. we will be able to cascade the carry bit from one adder to the other

- We use the **full adder** to compute the sum bit and the carry bit when two bits and a carry are added.
- The full adder can take three **1**-bit inputs and them to create a **2**-bit value.

The truth table includes five columns with **three inputs** and **two outputs**.

1. input x
2. input y
3. input **carry IN** c_i for the current column
4. output resulting **carry OUT** (carry-over), generated for the next column
5. output the resulting SUM s.

Inputs			outputs	
x	y	c_i	c_{i+1}	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

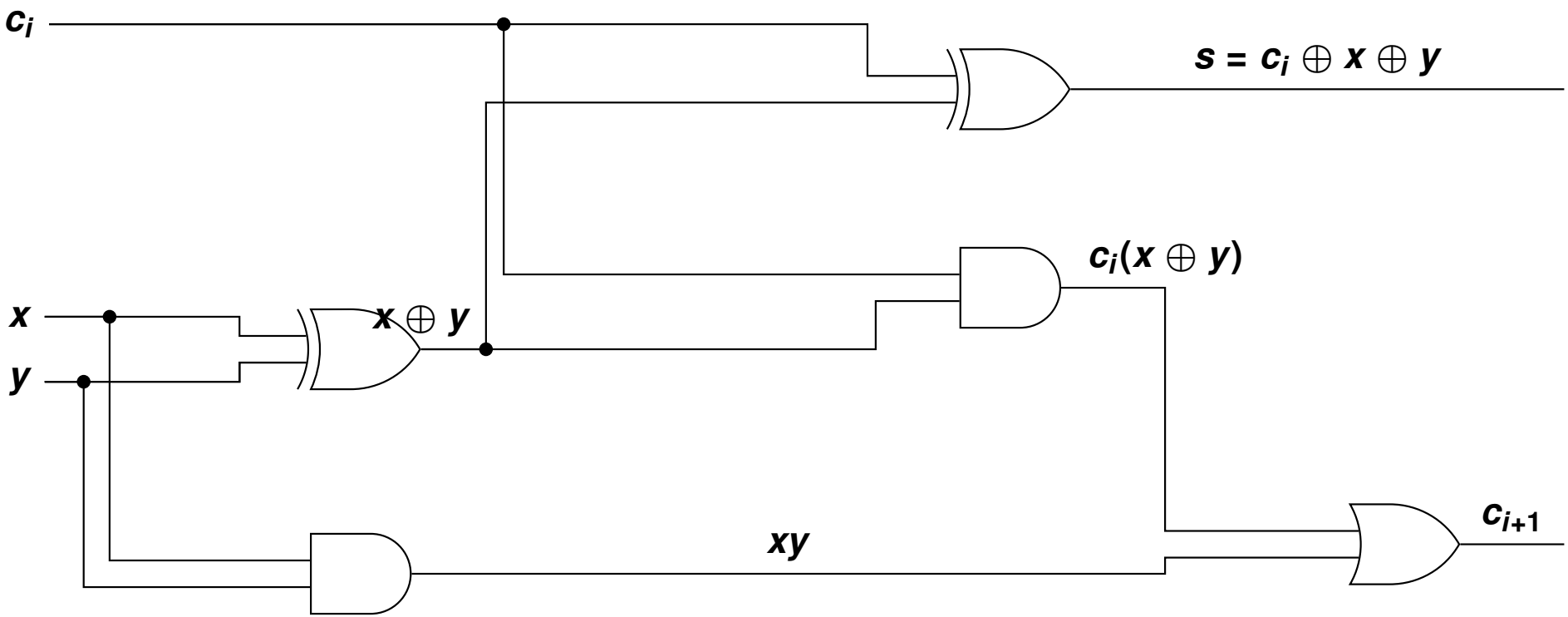
From this table we see that

$$c_{i+1} = \overline{x}yc_i + x\overline{y}c_i + xy\overline{c_i} + xyc_i = xy + c_i(x \oplus y)$$

$$s = \overline{x}\,\overline{y}c_i + \overline{x}y\overline{c_i} + x\overline{y}\,\overline{c_i} + xyc_i = x \oplus y \oplus c_i = s \oplus c_i$$

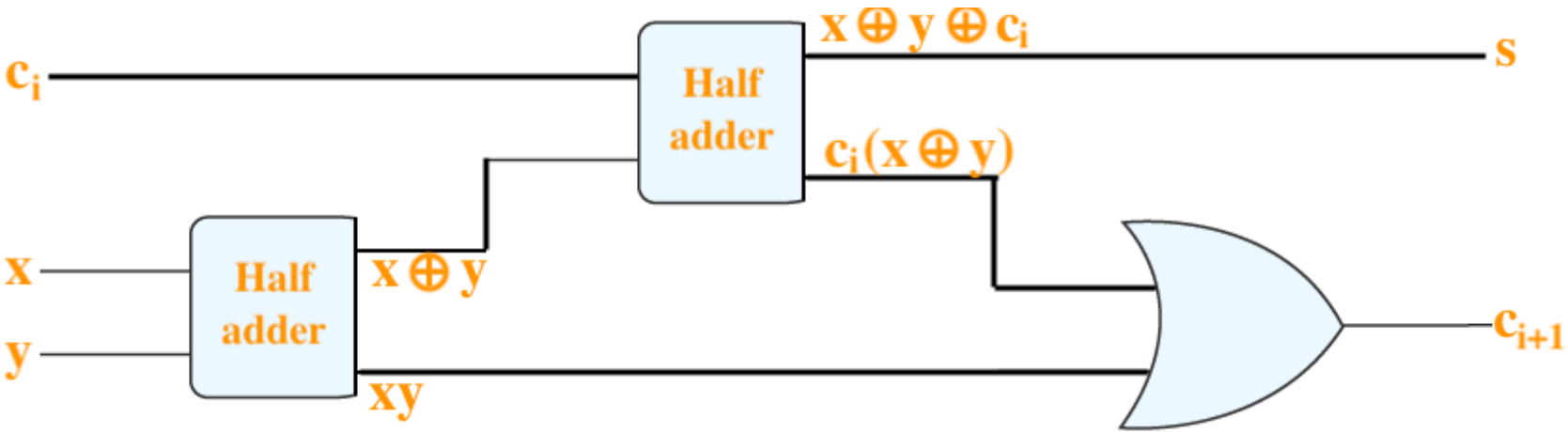
★ Adders ★
Full Adder

However, instead of designing the full adder from scratch, we will use half adders to produce the desired output.



★ Adders ★
Full Adder

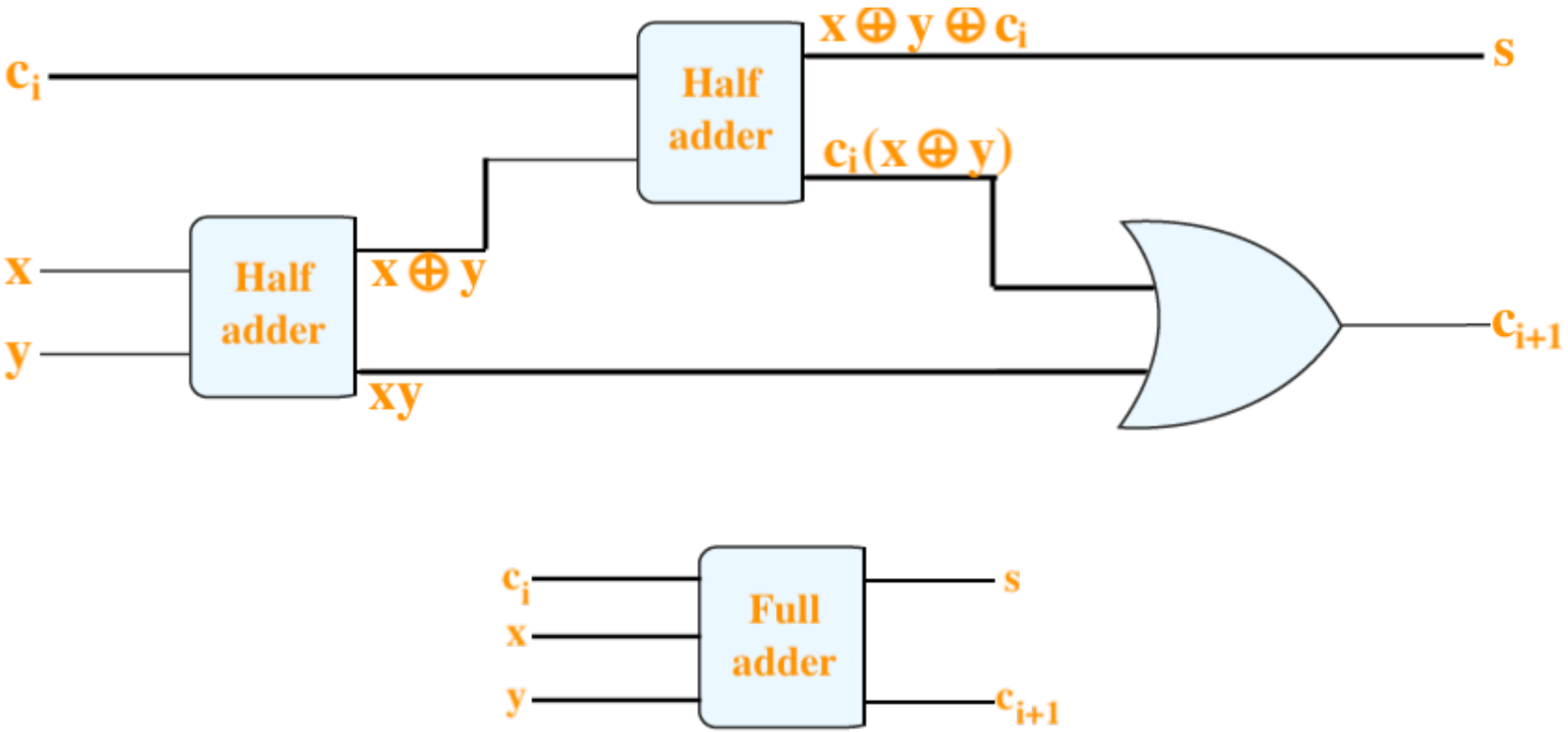
However, instead of designing the full adder from scratch, we will use half adders to produce the desired output.



★ Adders ★

Full Adder

However, instead of designing the full adder from scratch, we will use half adders to produce the desired output.

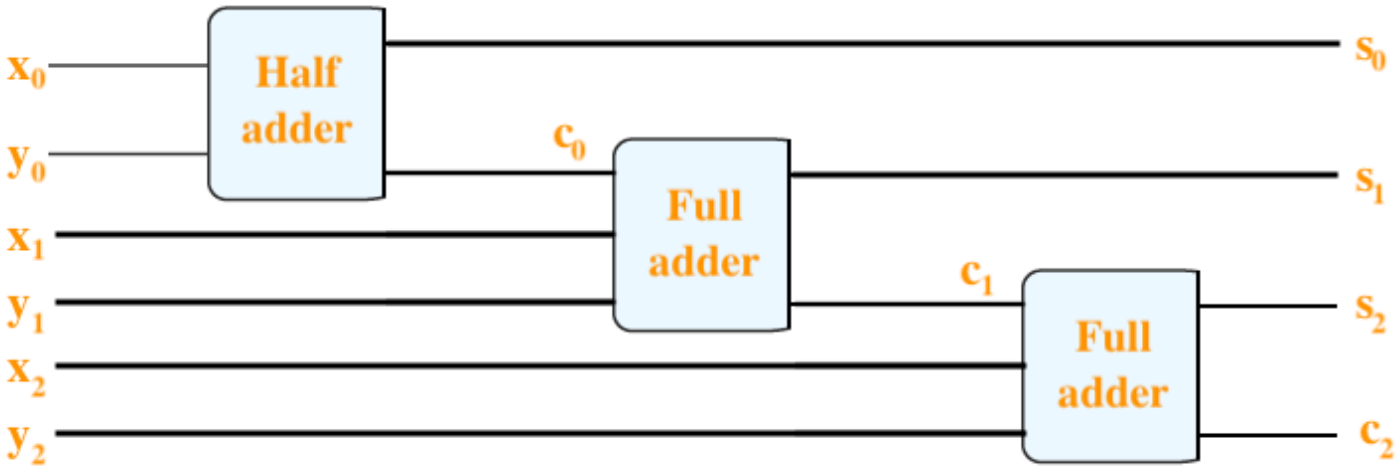


★ Adding two three-Bit integers with full and half adders ★

- Now that we know the basics of using boolean algebra to add bits, we can move onto more complicated adders. Just as we connected two half-adders to create a full-adder, we can connect many full-adders to add more and more bits. This is called a Parallel Adder.

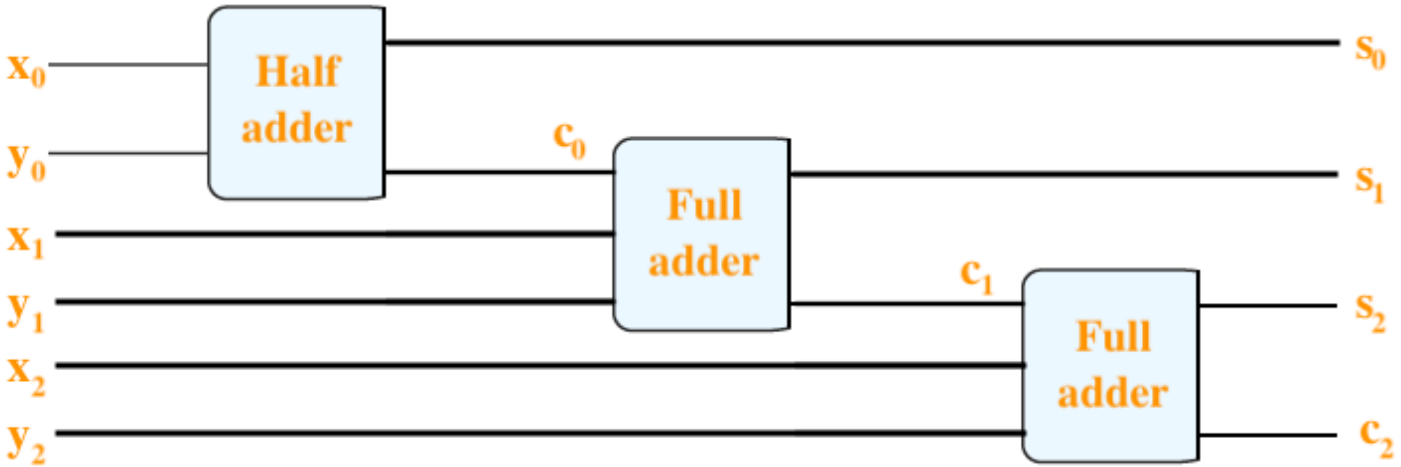
Adding two three-Bit integers with full and half adders

- Now that we know the basics of using boolean algebra to add bits, we can move onto more complicated adders. Just as we connected two half-adders to create a full-adder, we can connect many full-adders to add more and more bits. This is called a Parallel Adder.
- It consists of full adders connected in a chain where the output carry from each full adder is connected to the carry input of the next higher order full adder in the chain.
- The figure below, we show how full and half adders can be used to add the two three-bit integers $(x_2\ x_1\ x_0)_2$ and $(y_2\ y_1\ y_0)_2$ to produce the sum $(s_3\ s_2\ s_1\ s_0)_2$.

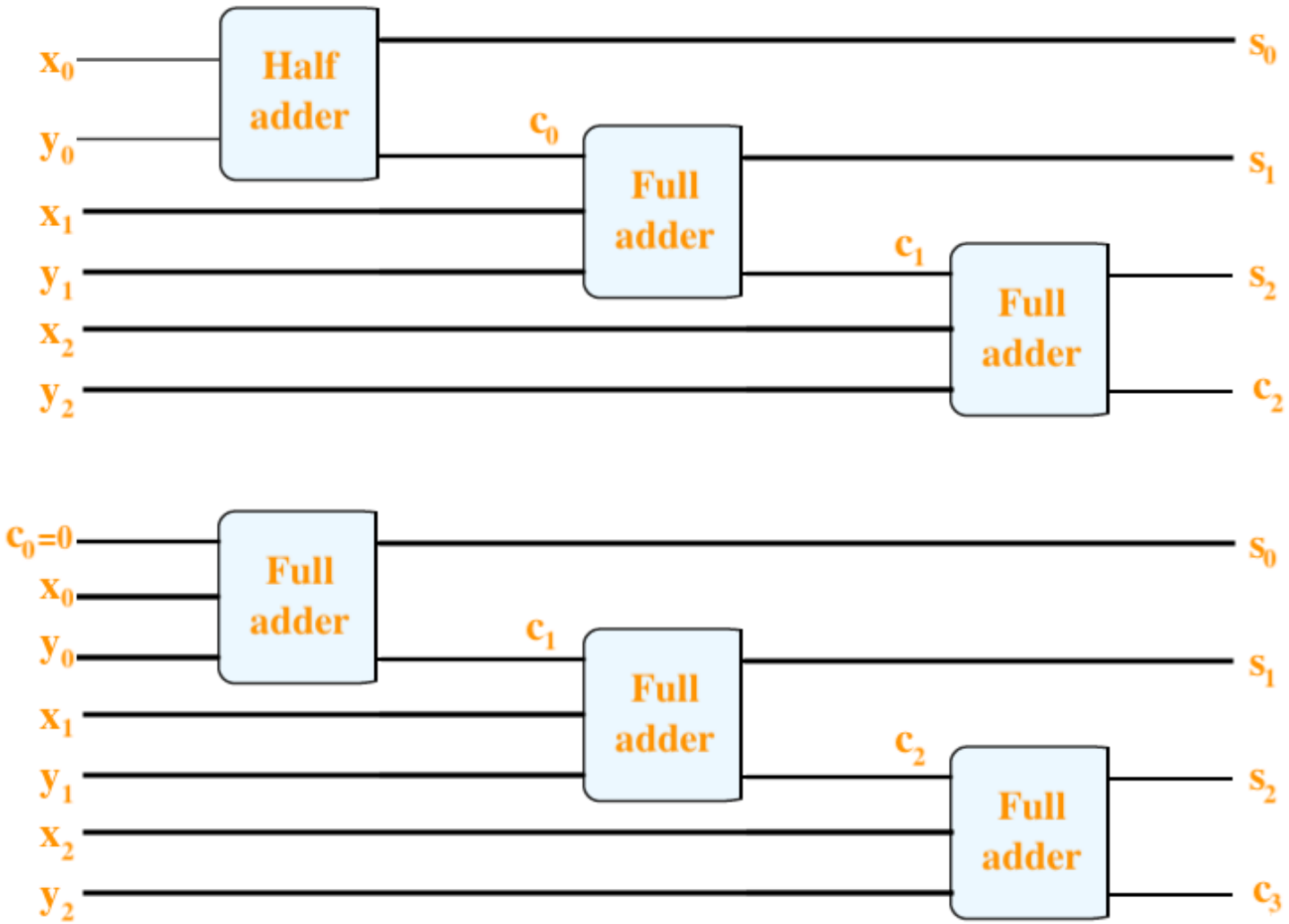


★ Adding two three-Bit integers with full and half adders ★

- Now that we know the basics of using boolean algebra to add bits, we can move onto more complicated adders. Just as we connected two half-adders to create a full-adder, we can connect many full-adders to add more and more bits. This is called a Parallel Adder.
- It consists of full adders connected in a chain where the output carry from each full adder is connected to the carry input of the next higher order full adder in the chain.
- The figure below, we show how full and half adders can be used to add the two three-bit integers $(x_2\ x_1\ x_0)_2$ and $(y_2\ y_1\ y_0)_2$ to produce the sum $(s_3\ s_2\ s_1\ s_0)_2$.



Note that s_3 , the highest-order bit in the sum, is given by the carry c_2 .


x0
y0
x1
y1
x2
y2
Half adder
Full adder
Full adder
c0
c1
s0
s1
s2
c2
c0=0
x0
y0
x1
y1
x2
y2
Full adder
Full adder
Full adder
c1
c2
s0
s1
s2
c3

In electronics, a subtractor can be designed using the same approach as that of an adder. The binary subtraction process is summarized below.

In electronics, a subtractor can be designed using the same approach as that of an adder. The binary subtraction process is summarized below.

- As with an adder, in general case of calculations on multi-bit numbers, three bits are involved in performing the subtraction for each bit of the difference:
  - the minuend ($\boldsymbol{x}$),
  - subtrahend ($\boldsymbol{y}$),
  - and a borrow in from the previous (less significant) bit order position ($\boldsymbol{B_{in}}$).

In electronics, a subtractor can be designed using the same approach as that of an adder. The binary subtraction process is summarized below.

- As with an adder, in general case of calculations on multi-bit numbers, three bits are involved in performing the subtraction for each bit of the difference:
  - the minuend (***x***),
  - subtrahend (***y***),
  - and a borrow in from the previous (less significant) bit order position (B_{in}).

- The outputs are the difference bit (D_{iff}) and borrow bit (B_{in+1}).

In electronics, a subtractor can be designed using the same approach as that of an adder. The binary subtraction process is summarized below.

- As with an adder, in general case of calculations on multi-bit numbers, three bits are involved in performing the subtraction for each bit of the difference:
  - the minuend (***x***),
  - subtrahend (***y***),
  - and a borrow in from the previous (less significant) bit order position ($\boldsymbol{B_{in}}$).

- The outputs are the difference bit ($\boldsymbol{D_{iff}}$) and borrow bit $\boldsymbol{(B_{in+1})}$.

- The subtractor is best understood by considering that the subtrahend and both borrow bits have negative weights, whereas the ***A*** and ***D*** bits are positive.

★ Half Subtractor ★

- The half subtractor is a combinational circuit which is used to perform subtraction of two bits.

★ Half Subtractor ★

- The half subtractor is a combinational circuit which is used to perform subtraction of two bits.
- It has two inputs, the minuend ($\boldsymbol{x}$) and subtrahend ($\boldsymbol{y}$) and two outputs the difference $\boldsymbol{D_{iff}}$ and borrows out $\boldsymbol{B_{out}}$.

★ Half Subtractor ★

- The half subtractor is a combinational circuit which is used to perform subtraction of two bits.
- It has two inputs, the minuend ($\boldsymbol{x}$) and subtrahend ($\boldsymbol{y}$) and two outputs the difference $\boldsymbol{D_{iff}}$ and borrows out $\boldsymbol{B_{out}}$.
- Borrows out signal is set when the subtractor needs to borrow from the next digit in a multi digit subtraction. That is, $\boldsymbol{B_{out}} = \mathbf{1}$ when $\boldsymbol{x} < \boldsymbol{y}$ if and only if $\boldsymbol{x} = \mathbf{0}$ and $\boldsymbol{y} = \mathbf{1}$.

★ Half Subtractor ★

- The half subtractor is a combinational circuit which is used to perform subtraction of two bits.
- It has two inputs, the minuend ($\boldsymbol{x}$) and subtrahend ($\boldsymbol{y}$) and two outputs the difference $\boldsymbol{D_{iff}}$ and borrows out $\boldsymbol{B_{out}}$.
- Borrows out signal is set when the subtractor needs to borrow from the next digit in a multi digit subtraction. That is, $\boldsymbol{B_{out} = 1}$ when $\boldsymbol{x} < \boldsymbol{y}$ if and only if $\boldsymbol{x = 0}$ and $\boldsymbol{y = 1}$.
- From this table we see that

$$\boldsymbol{B_{out} = \overline{x}y} \quad \text{and that} \quad \boldsymbol{D_{iff} = \overline{x}y + x\overline{y} = x \oplus y}.$$

X	–	Y	=	$Borrow$	D_{iff}
0	–	0	=	0	0
0	–	1	=	1	1
1	–	0	=	0	1
1	–	1	=	0	0

★ Half Subtractor ★

- The half subtractor is a combinational circuit which is used to perform subtraction of two bits.
- It has two inputs, the minuend ($\boldsymbol{x}$) and subtrahend ($\boldsymbol{y}$) and two outputs the difference $\boldsymbol{D_{iff}}$ and borrows out $\boldsymbol{B_{out}}$.
- Borrows out signal is set when the subtractor needs to borrow from the next digit in a multi digit subtraction. That is, $\boldsymbol{B_{out} = 1}$ when $\boldsymbol{x} < \boldsymbol{y}$ if and only if $\boldsymbol{x = 0}$ and $\boldsymbol{y = 1}$.
- From this table we see that

$$\boldsymbol{B_{out} = \overline{x}y} \quad \text{and that} \quad \boldsymbol{D_{iff} = \overline{x}y + x\overline{y} = x \oplus y}.$$

X	$-$	Y	$=$	$Borrow$	D_{iff}
0	–	0	=	0	0
0	–	1	=	1	1
1	–	0	=	0	1
1	–	1	=	0	0

- An important point worth mentioning is that the half subtractor diagram aside implements $\boldsymbol{x - y}$ and not $\boldsymbol{y - x}$ since $\boldsymbol{B_{out}}$ on the diagram is given by $\boldsymbol{B_{out} = \overline{x}y}$.

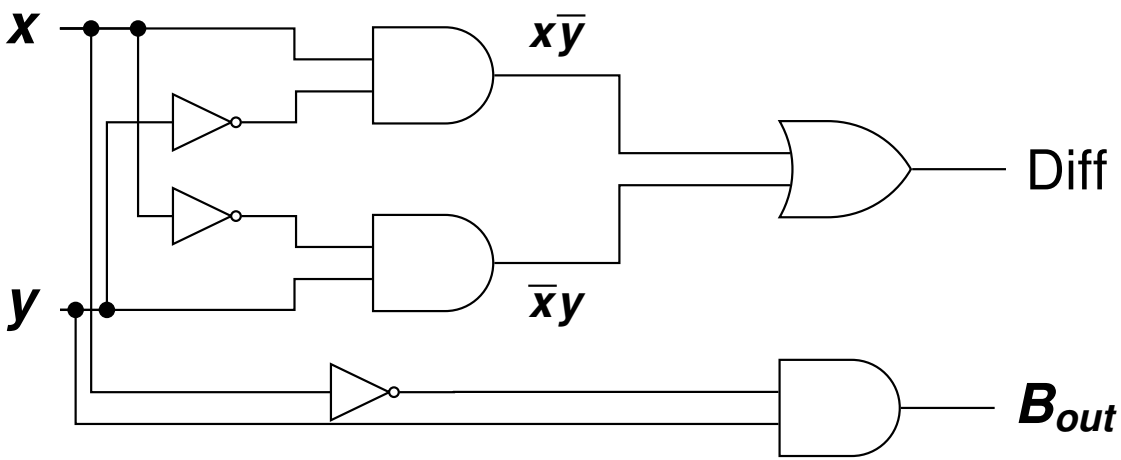
★ Half Subtractor ★

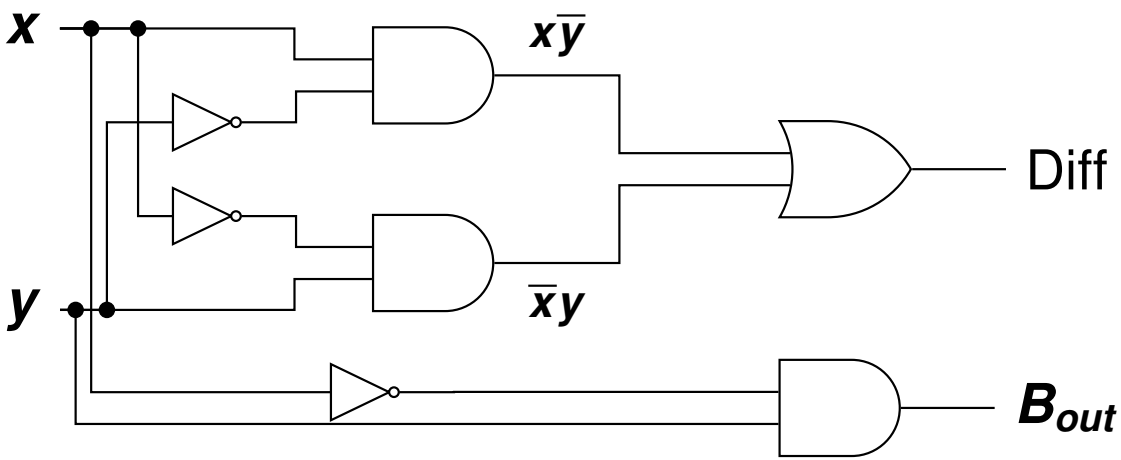
- The half subtractor is a combinational circuit which is used to perform subtraction of two bits.
- It has two inputs, the minuend (x) and subtrahend (y) and two outputs the difference D_{iff} and borrows out B_{out}.
- Borrows out signal is set when the subtractor needs to borrow from the next digit in a multi digit subtraction. That is, $B_{out} = 1$ when $x < y$ if and only if $x = 0$ and $y = 1$.
- From this table we see that

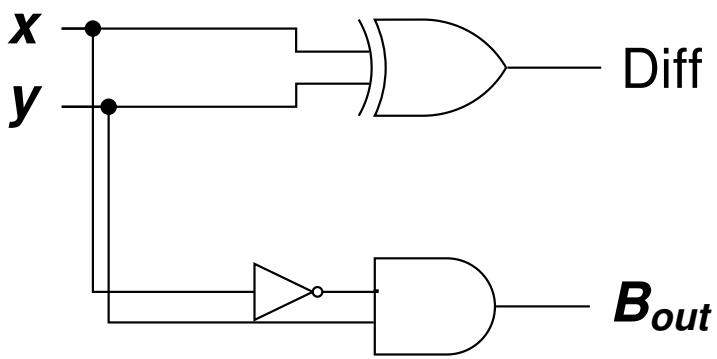
  $$B_{out} = \overline{x}y \quad \text{and that} \quad D_{iff} = \overline{x}y + x\overline{y} = x \oplus y.$$

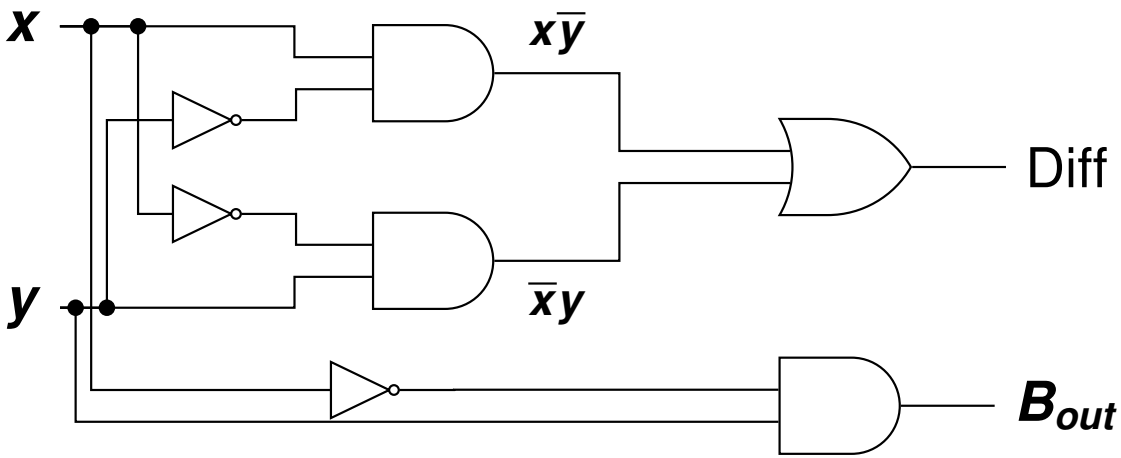
X	$-$	Y	$=$	$Borrow$	D_{iff}
0	–	0	=	0	0
0	–	1	=	1	1
1	–	0	=	0	1
1	–	1	=	0	0

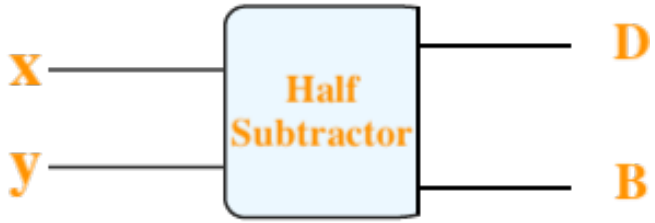
- An important point worth mentioning is that the half subtractor diagram aside implements $x - y$ and not $y - x$ since B_{out} on the diagram is given by $B_{out} = \overline{x}y$.
- This is an important distinction to make since subtraction itself is not commutative, but the difference bit D_{iff} is calculated using an XOR gate which is commutative.


x
y
$x\bar{y}$
$\bar{x}y$
Diff
B_{out}


x
y
$x\bar{y}$
$\bar{x}y$
Diff
B_{out}


x
y
Diff
B_{out}





- Half subtractor is used to subtract the least significant column numbers. For subtraction of multi-digit numbers, it can be used for the LSB
- Half subtractor is used to reduce the force of audio or radio signals
- It can be used in amplifiers to reduce the sound distortion
- Half subtractor is used in ALU of processor
- It can be used to increase and decrease operators and also calculates the addresses

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.
- This circuit has three inputs and two outputs. The three inputs x, y and $B_{in} = b_i$, denote the minuend, subtrahend, and previous borrow, respectively.

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.
- This circuit has three inputs and two outputs. The three inputs x, y and $B_{in} = b_i$, denote the minuend, subtrahend, and previous borrow, respectively.
- The two outputs, $D_{iff} = d$ and $B_{out} = b_{i+1}$ represent the difference and output borrows, respectively.

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.
- This circuit has three inputs and two outputs. The three inputs $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{B_{in} = b_i}$, denote the minuend, subtrahend, and previous borrow, respectively.
- The two outputs, $\boldsymbol{D_{iff} = d}$ and $\boldsymbol{B_{out} = b_{i+1}}$ represent the difference and output borrows, respectively.
- Generally, the full subtractor is one of the most used and essential combinational logic circuits. It is a basic electronic device, used to perform subtraction of two binary numbers.

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.
- This circuit has three inputs and two outputs. The three inputs x, y and $B_{in} = b_i$, denote the minuend, subtrahend, and previous borrow, respectively.
- The two outputs, $D_{iff} = d$ and $B_{out} = b_{i+1}$ represent the difference and output borrows, respectively.
- Generally, the full subtractor is one of the most used and essential combinational logic circuits. It is a basic electronic device, used to perform subtraction of two binary numbers.

Inputs			outputs	
x	y	b_i	d	b_{i+1}
1	1	1	1	1
1	1	0	0	0
1	0	1	0	0
1	0	0	1	0
0	1	1	0	1
0	1	0	1	1
0	0	1	1	1
0	0	0	0	0

From this table we see that

d =?

b_{i+1}?

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.
- This circuit has three inputs and two outputs. The three inputs x, y and $B_{in} = b_i$, denote the minuend, subtrahend, and previous borrow, respectively.
- The two outputs, $D_{iff} = d$ and $B_{out} = b_{i+1}$ represent the difference and output borrows, respectively.
- Generally, the full subtractor is one of the most used and essential combinational logic circuits. It is a basic electronic device, used to perform subtraction of two binary numbers.

Inputs			outputs	
x	y	b_i	d	b_{i+1}
1	1	1	1	1
1	1	0	0	0
1	0	1	0	0
1	0	0	1	0
0	1	1	0	1
0	1	0	1	1
0	0	1	1	1
0	0	0	0	0

From this table we see that

$$d = xyb_i + x\overline{y}\,\overline{b_i} + \overline{x}y\overline{b_i} + \overline{x}\,\overline{y}b_i = b_i(xy + \overline{x}\,\overline{y}) + \overline{b_i}(x\overline{y} + \overline{x}y)$$

$$b_{i+1} = xyb_i + \overline{x}yb_i + \overline{x}y\overline{b_i} + \overline{x}\,\overline{y}b_i = \overline{x}y(b_i + \overline{b_i}) + b_i(xy + \overline{x}\,\overline{y})$$

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.
- This circuit has three inputs and two outputs. The three inputs x, y and $B_{in} = b_i$, denote the minuend, subtrahend, and previous borrow, respectively.
- The two outputs, $D_{iff} = d$ and $B_{out} = b_{i+1}$ represent the difference and output borrows, respectively.
- Generally, the full subtractor is one of the most used and essential combinational logic circuits. It is a basic electronic device, used to perform subtraction of two binary numbers.

Inputs			outputs	
x	y	b_i	d	b_{i+1}
1	1	1	1	1
1	1	0	0	0
1	0	1	0	0
1	0	0	1	0
0	1	1	0	1
0	1	0	1	1
0	0	1	1	1
0	0	0	0	0

From this table we see that

$$d = xyb_i + x\overline{y}\,\overline{b_i} + \overline{x}y\,\overline{b_i} + \overline{xy}\,b_i = b_i(xy + \overline{xy}) + \overline{b_i}(x\overline{y} + \overline{x}y)$$

$$= b_i(x \odot y) + \overline{b_i}(x \oplus y) = b_i\overline{(x \oplus y)} + \overline{b_i}(x \oplus y)$$

$$b_{i+1} = xyb_i + \overline{x}yb_i + \overline{x}y\,\overline{b_i} + \overline{xy}\,b_i = \overline{x}y(b_i + \overline{b_i}) + b_i(xy + \overline{x}\,\overline{y})$$

$$= \overline{x}y + b_i\overline{(x \oplus y)}$$

★ Full Subtractor ★

- A **full subtractor** is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.
- This circuit has three inputs and two outputs. The three inputs x, y and $B_{in} = b_i$, denote the minuend, subtrahend, and previous borrow, respectively.
- The two outputs, $D_{iff} = d$ and $B_{out} = b_{i+1}$ represent the difference and output borrows, respectively.
- Generally, the full subtractor is one of the most used and essential combinational logic circuits. It is a basic electronic device, used to perform subtraction of two binary numbers.

Inputs			outputs	
x	y	b_i	d	b_{i+1}
1	1	1	1	1
1	1	0	0	0
1	0	1	0	0
1	0	0	1	0
0	1	1	0	1
0	1	0	1	1
0	0	1	1	1
0	0	0	0	0

From this table we see that

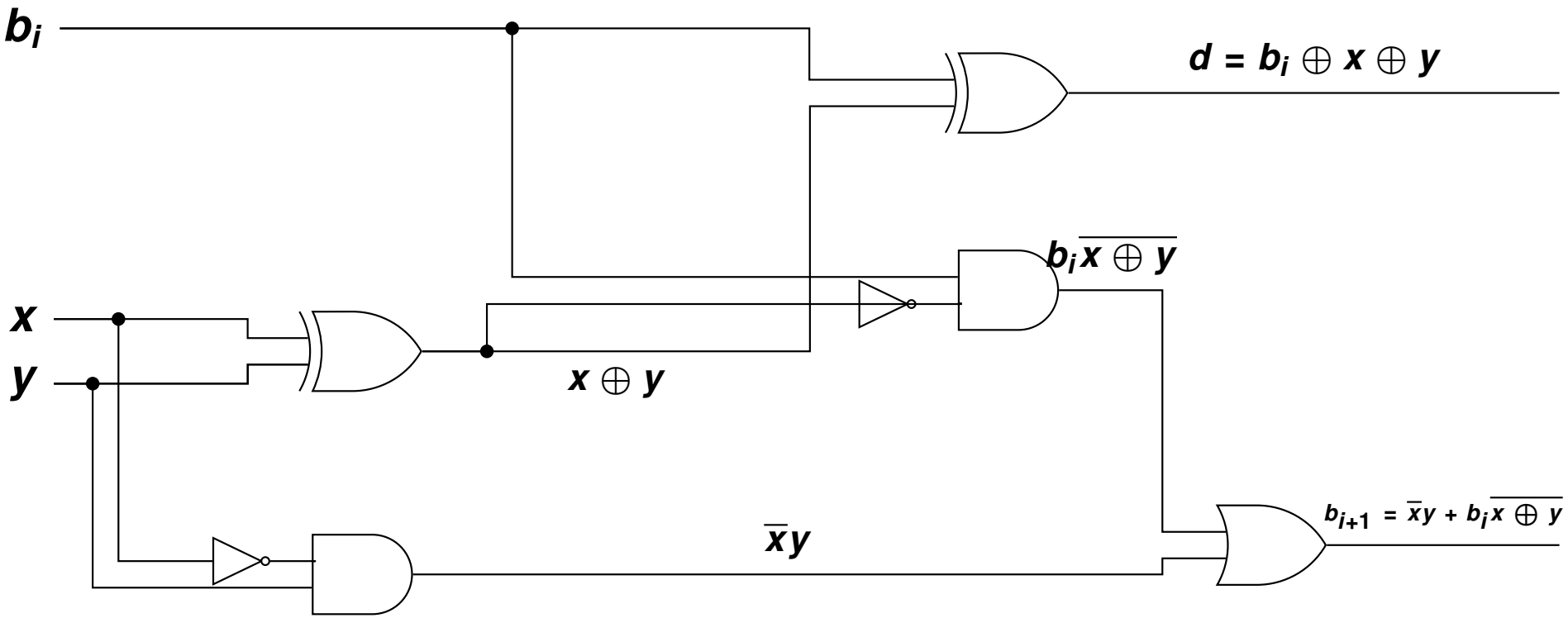
$$d = xyb_i + x\overline{y}\,\overline{b_i} + \overline{x}y\,\overline{b_i} + \overline{x}\,\overline{y}b_i = b_i(xy + \overline{x}\,\overline{y}) + \overline{b_i}(x\overline{y} + \overline{x}y)$$

$$= b_i(x \odot y) + \overline{b_i}(x \oplus y) = b_i\overline{(x \oplus y)} + \overline{b_i}(x \oplus y)$$

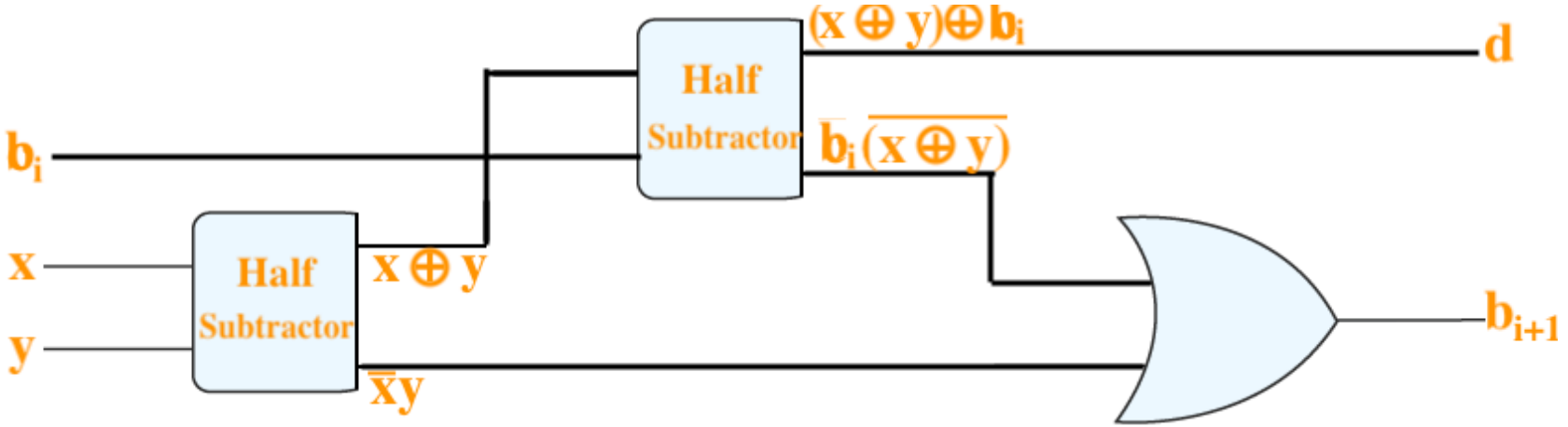
$$= b_i \oplus (x \oplus y)$$

$$b_{i+1} = xyb_i + \overline{x}yb_i + \overline{x}y\,\overline{b_i} + \overline{x}\,\overline{y}b_i = \overline{x}y(b_i + \overline{b_i}) + b_i(xy + \overline{x}\,\overline{y})$$

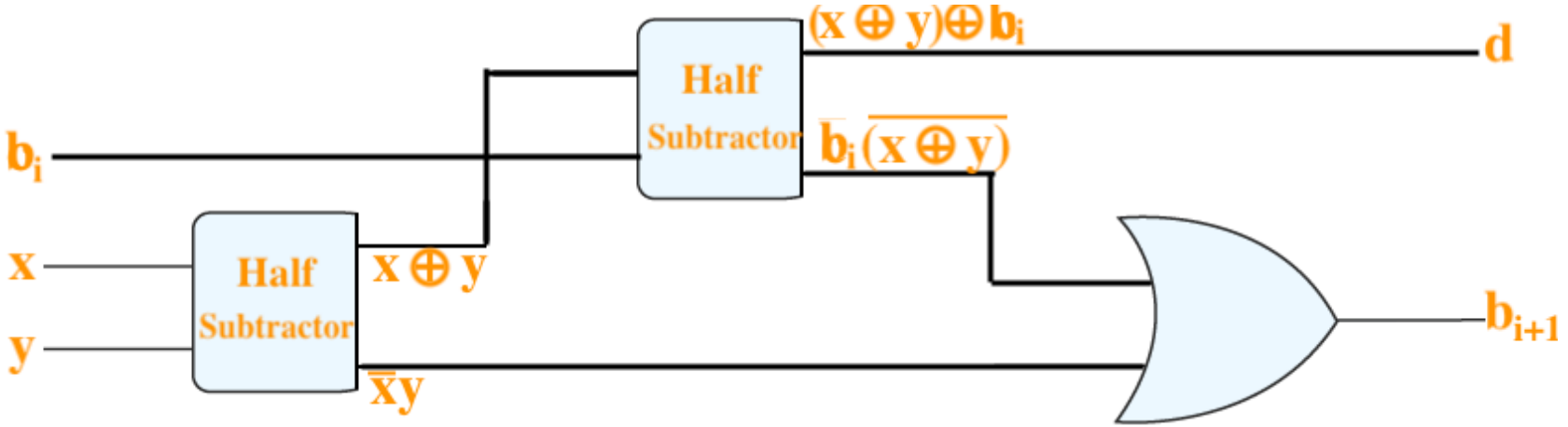
$$= \overline{x}y + b_i\overline{(x \oplus y)}$$

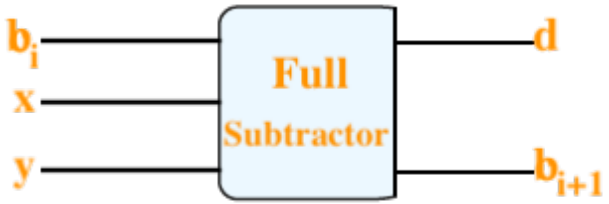

b_i
$d = b_i \oplus x \oplus y$
$b_i\overline{x \oplus y}$
x
y
$x \oplus y$
$\overline{x}y$
$b_{i+1} = \overline{x}y + b_i\overline{x \oplus y}$

However, instead of designing the full subtractor from scratch, we will use half subtractors to produce the desired output.



However, instead of designing the full subtractor from scratch, we will use half subtractors to produce the desired output.





Subtracting two three-Bit integers with full and half subtractors

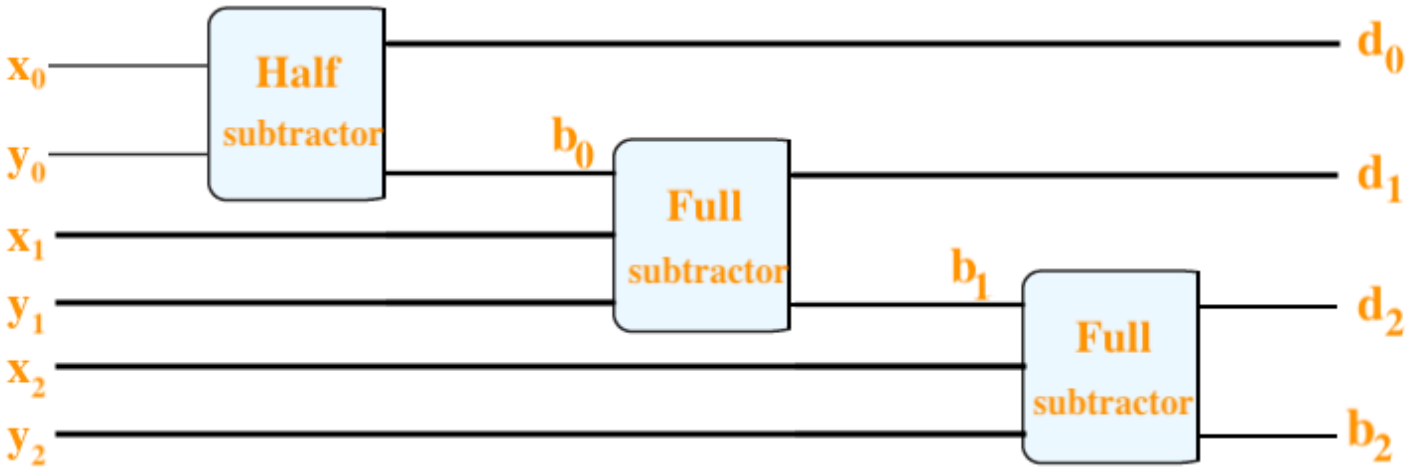
- A single full subtractor performs the difference between two one bit numbers and an input borrow. But a Parallel subtractor is a digital circuit capable of finding the arithmetic difference of two binary numbers that is greater than one bit in length by operating on corresponding pairs of bits in parallel.

Subtracting two three-Bit integers with full and half subtractors

- A single full subtractor performs the difference between two one bit numbers and an input borrow. But a Parallel subtractor is a digital circuit capable of finding the arithmetic difference of two binary numbers that is greater than one bit in length by operating on corresponding pairs of bits in parallel.
- It consists of full adders connected in a chain where the output borrow from each full adder is connected to the borrow input of the next higher order full adder in the chain.

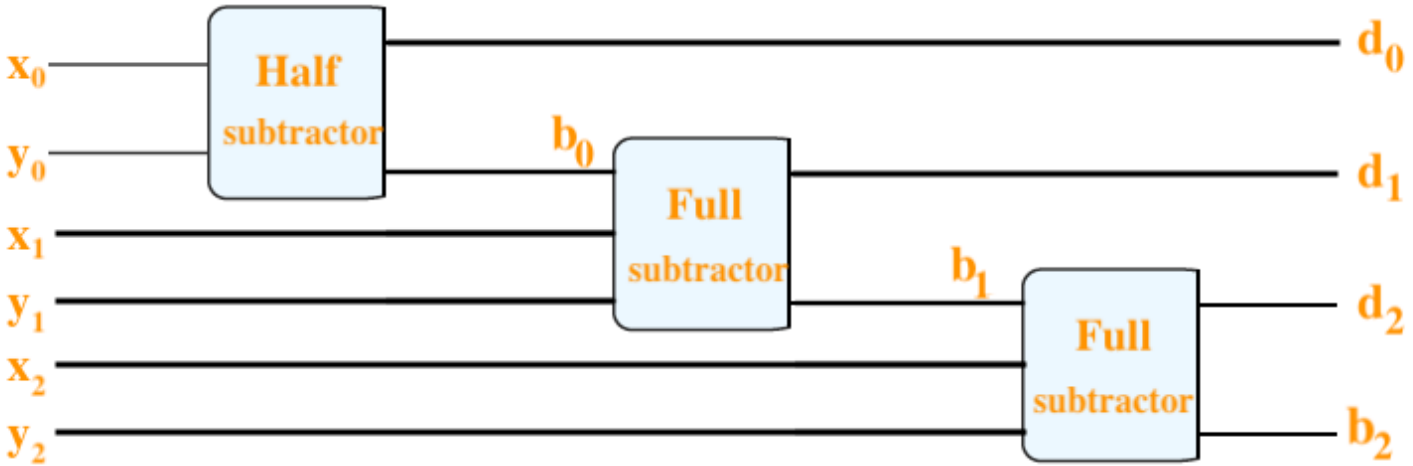
★ Subtracting two three-Bit integers with full and half subtractors ★

- A single full subtractor performs the difference between two one bit numbers and an input borrow. But a Parallel subtractor is a digital circuit capable of finding the arithmetic difference of two binary numbers that is greater than one bit in length by operating on corresponding pairs of bits in parallel.
- It consists of full adders connected in a chain where the output borrow from each full adder is connected to the borrow input of the next higher order full adder in the chain.
- The figure below, we show how full and half adders can be used to add the two three-bit integers $(x_2\ x_1\ x_0)_2$ and $(y_2\ y_1\ y_0)_2$ to produce the difference $(d_3\ d_2\ d_1\ d_0)_2$.

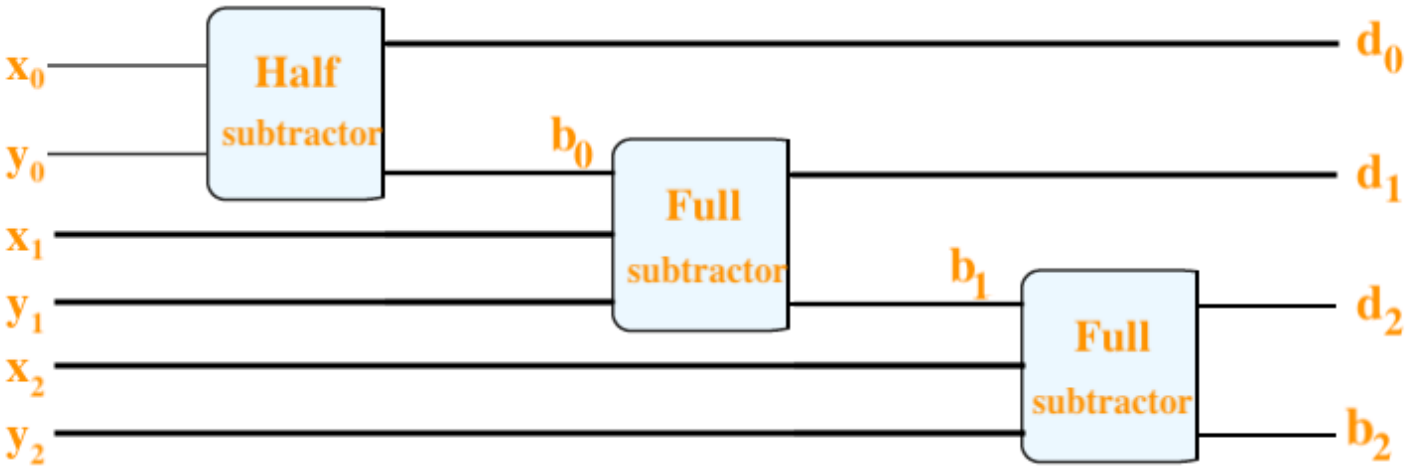


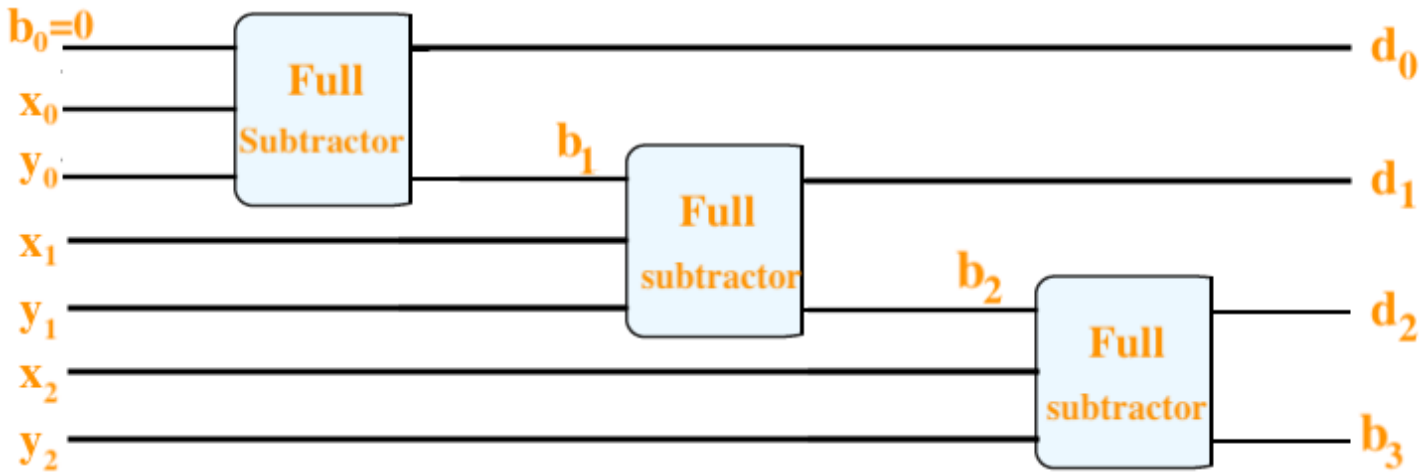
★ Subtracting two three-Bit integers with full and half subtractors ★

- A single full subtractor performs the difference between two one bit numbers and an input borrow. But a Parallel subtractor is a digital circuit capable of finding the arithmetic difference of two binary numbers that is greater than one bit in length by operating on corresponding pairs of bits in parallel.
- It consists of full adders connected in a chain where the output borrow from each full adder is connected to the borrow input of the next higher order full adder in the chain.
- The figure below, we show how full and half adders can be used to add the two three-bit integers $(x_2\ x_1\ x_0)_2$ and $(y_2\ y_1\ y_0)_2$ to produce the difference $(d_3\ d_2\ d_1\ d_0)_2$.



Note that d_3 , the highest-order bit in the sum, is given by the borrow out b_2 .


x_0
y_0
x_1
y_1
x_2
y_2
Half subtractor
b_0
Full subtractor
b_1
Full subtractor
d_0
d_1
d_2
b_2


$b_0=0$
x_0
y_0
x_1
y_1
x_2
y_2
Full Subtractor
b_1
Full subtractor
b_2
Full subtractor
d_0
d_1
d_2
b_3

Parallel subtractor

Binary Subtractor using 2's Complement

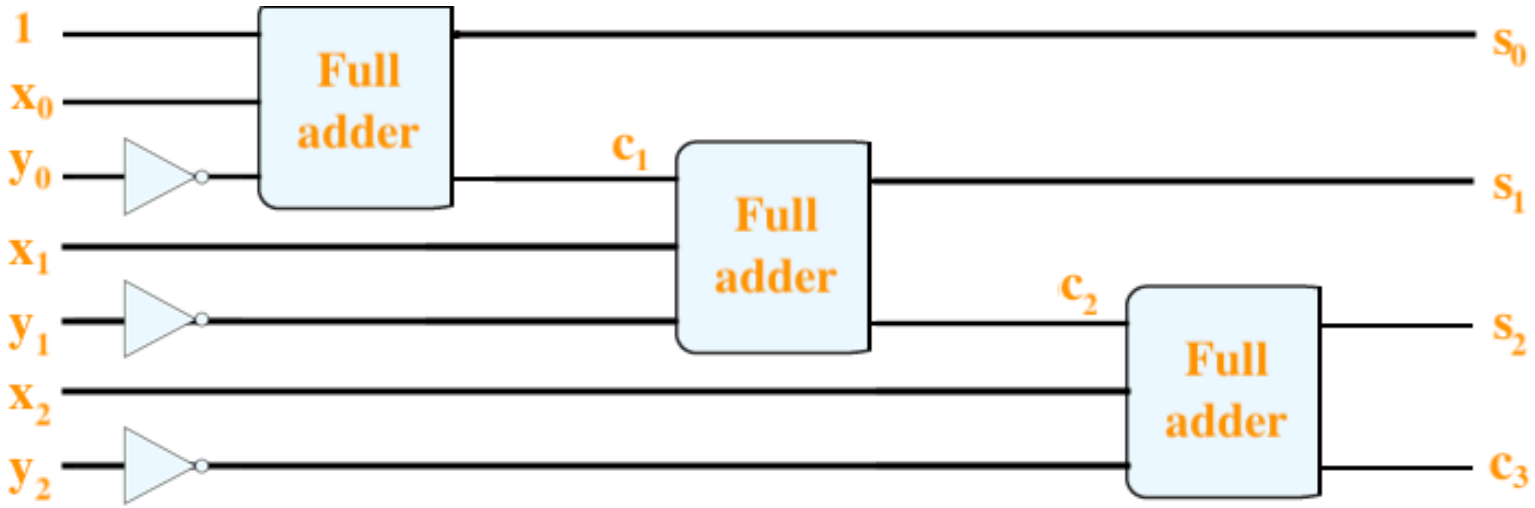
- A **2**'s complement of a number can be achieved by complementing each digit of the the subtrahend like zero's to ones and ones to zeros and add one.
- $$(\blacksquare\ \blacksquare\ \blacksquare\ \blacksquare)^{2's} = \underbrace{(\blacksquare\ \blacksquare\ \blacksquare\ \blacksquare)^{1's}}_{\begin{array}{c}1 \longrightarrow 0\\ 0 \longrightarrow 1\end{array}} + 0001 = \overline{\blacksquare}\ \overline{\blacksquare}\ \overline{\blacksquare}\ \overline{\blacksquare} + 0001$$

$$\text{Minuend} - \text{Subtrahend} = \text{Minuend} + (\text{Subtrahend})^{2's} = \text{Minuend} + (\text{Subtrahend})^{1's} + 1$$

- So by using an ***n***-bit adder and ***n*** number of inverters (NOT Gates), the process of subtraction becomes an addition as we can use two's complement notation on all the bits in the subtrahend and setting the carry input of the least significant bit to a logic "**1**" (HIGH).
- $\boldsymbol{X} - \boldsymbol{Y}$ is the same as saying, $\boldsymbol{X} + (-\boldsymbol{Y})$ which equals $\boldsymbol{X}$ plus the two's complement of $\boldsymbol{Y}$.

$$x_0x_1x_2 - y_0y_1y_2 = x_0x_1x_2 + (y_0y_1y_2)^{2's} = x_0x_1x_2 + (y_0y_1y_2)^{1's} + 0001 = x_0x_1x_2 + \overline{y}_0\overline{y}_1\overline{y}_2 + 0001$$

$$x_0x_1x_2 - y_0y_1y_2 = x_0x_1x_2 + (y_0y_1y_2)^{2's} = x_0x_1x_2 + (y_0y_1y_2)^{1's} + 0001 = x_0x_1x_2 + \overline{y}_0\overline{y}_1\overline{y}_2 + 0001$$

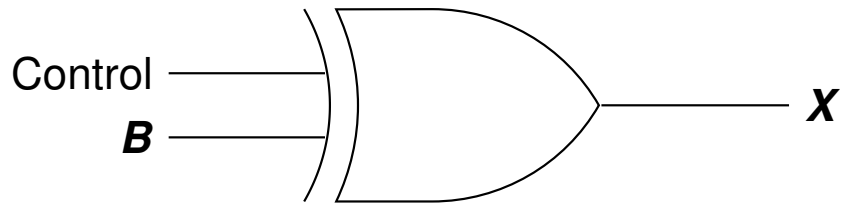


★ Adder / Subtractor Control ★

- How an XOR gate is used here to change the adder into a subtractor by inverting the B inputs can be seen from the truth table for an XOR gate.

- Notice that if CONTROL input of the XOR gate is at logic **0**, then the XOR gate selects input ***B***, but if CONTROL input is logic **1**, then it selects the inverse of input ***B*** (i.e. $\overline{B}$).

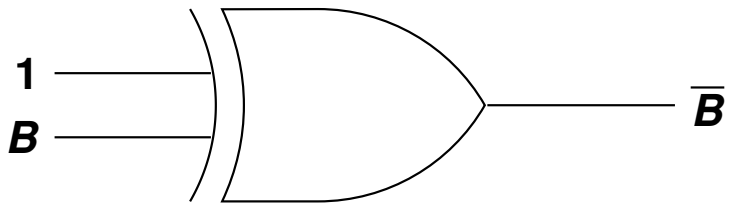
	Control	***B***	***X***
$X = B$	**0**	**0**	**0**
	0	**1**	**1**
$X = \overline{B}$	**1**	**0**	**1**
	1	**1**	**0**

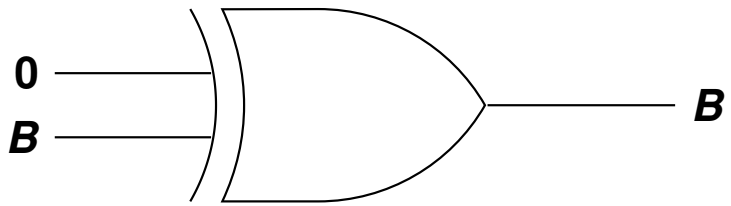


★ Adder / Subtractor Control ★

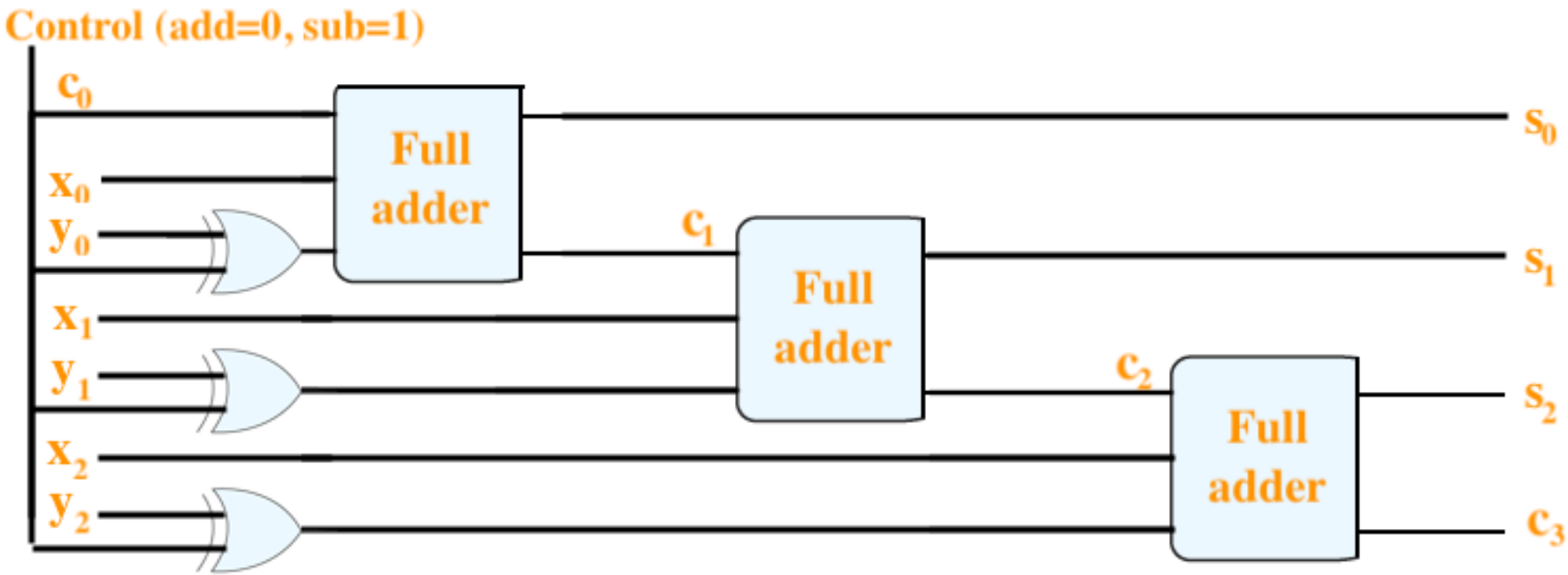
- How an XOR gate is used here to change the adder into a subtractor by inverting the B inputs can be seen from the truth table for an XOR gate.

- Notice that if CONTROL input of the XOR gate is at logic **0**, then the XOR gate selects input ***B***, but if CONTROL input is logic **1**, then it selects the inverse of input ***B*** (i.e. $\overline{B}$).





★ Adder / Subtractor Control ★



- Arithmetic operations have a potential to run into a condition known as overflow.
- Overflow occurs with respect to the **size of the data type** that must accommodate the result.
- Overflow indicates that the result was too large or too small to fit in the original data type.
- When two signed **2**'s complement numbers are added, overflow is detected if:
  1. both operands are positive and the result is negative, or
  2. both operands are negative and the result is positive.
- When two unsigned numbers are added, overflow occurrs if there is a **carry out** of the leftmost bit.

★ Overflow Detection ★

Binary Arithmetic

- Computers don't know the difference between signed and unsigned binary numbers.
- This is a good thing, because it makes logic circuits fast.
- This is also a bad thing, because distinguishing between signed and unsigned is our responsibility.
- The distinction is very important when detecting an **overflow** after addition or subtraction.
- Correct approach to **detect the overflow** is to consider two separate cases:
  1. Overflow when adding unsigned numbers.
  2. Overflow when adding signed numbers.

★ Overflow Detection ★

Adding Unsigned Numbers

1. Let's first solve the problem for addition of one-bit quantities:

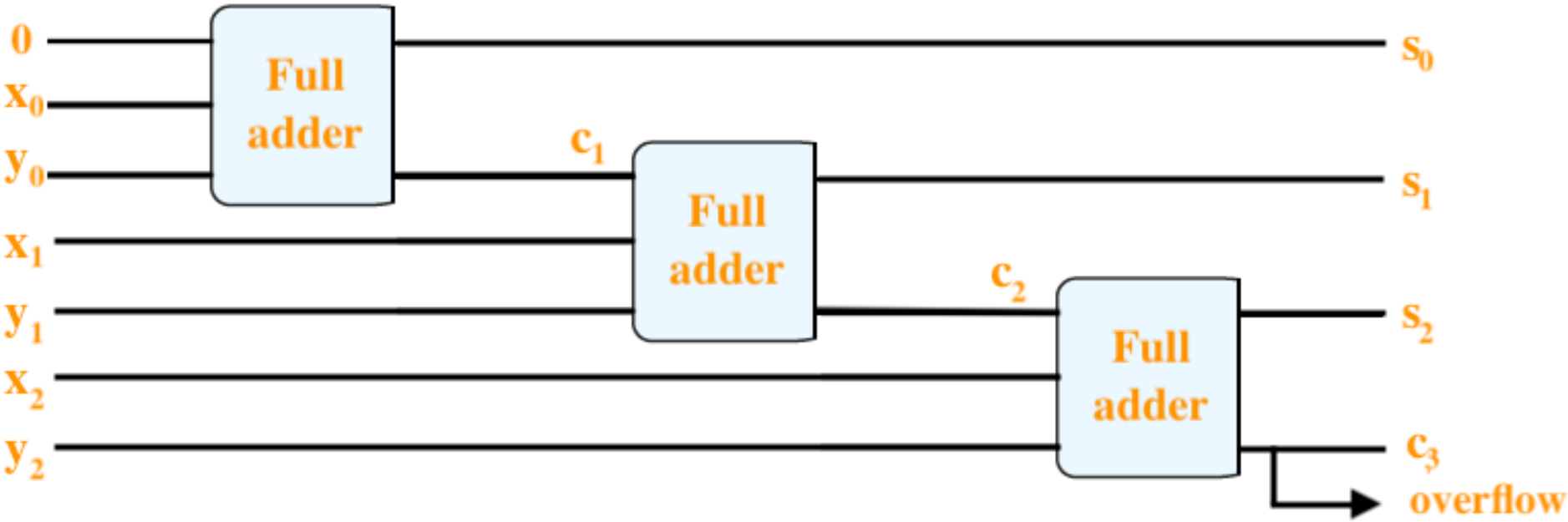
0	**+**	**0**	**=**	**0**
0	**+**	**1**	**=**	**1**
1	**+**	**0**	**=**	**1**
1	**+**	**1**	**=**	**10**

   - The last line indicates that we have a carry output.
   - That is, **one-bit** quantity cannot accommodate (**1 + 1**).
   - Therefore, larger data type is required for (**1 + 1**) to succeed.

2. When **multi-bit** unsigned quantities are added, overflow occurrs if there is a **carry out** from the leftmost (most significant) bit.

Overflow Detection

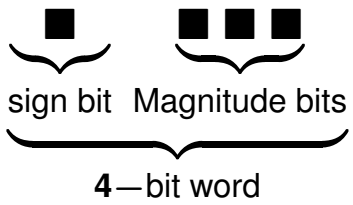
Overflow Detection Circuit for Unsigned Addition



★ Binary Arithmetic ★

Adding Signed Numbers

- Consider overflow detection when adding two **four-bit** (byte) signed quantities.
- Although the first bit is required to represent the sign and the other bits has to represent the data.



- Therefore, four-bit data type is required:

1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
−8	−7	−6	−5	−4	−3	−2	−1	0	+1	+2	+3	+4	+5	+6	+7

- **1000 = −8 + 0 = −8**,
- **1001 = −8 + 1 = −7**,
- **1010 = −8 + 2 = −6**,
- **1011 = −8 + 2 + 1 = −5**,
- **1100 = −8 + 4 = −4**,
- **1110 = −8 + 4 + 2 = −2**,
- **1111 = −8 + 4 + 2 + 1 = −1**.

★ Binary Arithmetic ★

The Full Adder Truth Table

Recall the truth table for the 2's complement full adder logic:
The truth table includes five columns with **three inputs** and **two outputs**.

Inputs			outputs	
x	y	c_i	c_{i+1}	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

1. input x
2. input y
3. carry IN c_i for the current column
4. resulting **carry OUT** c_{i+1} (carry-over), generated for the next column
5. the resulting **SUM**.

- The full adder knows nothing about the difference between signed and unsigned numbers.
- In 2's complement binary representation, the sign bit is simply the leftmost, or most significant, bit of the data type.
- The full adder circuit will be adding the sign bit column just as any other bit.

★ Adding Signed Numbers ★

The Overflow Output

Full adder truth table for the sign bit can be extended to include new output which indicates if overflow condition has occurred.

Our task is to populate the OVERFLOW column with corresponding values.

Inputs			outputs		
x	y	c_i	c_{i+1}	s	**Overflow**
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	

★ Adding Signed Numbers ★

The Overflow Output

Full adder truth table for the sign bit can be extended to include new output which indicates if overflow condition has occurred.

Inputs			outputs		
x	y	c_i	c_{i+1}	s	**Overflow**
0	0	0	0	0	
0	0	1	0	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	

Our task is to populate the OVERFLOW column with corresponding values.

1. When operands have ***opposite*** signs, their sum will ***never overflow***.

★ Adding Signed Numbers ★

The Overflow Output

Full adder truth table for the sign bit can be extended to include new output which indicates if overflow condition has occurred.

Our task is to populate the OVERFLOW column with corresponding values.

1. When operands have ***opposite*** signs, their sum will ***never overflow***.
2. There is ***no overflow***, if:
   - ***both*** operands are ***positive*** and the sum is ***positive***.
   - ***both*** operands are ***negative*** and the sum is ***negative***.

Inputs			outputs		
x	***y***	c_i	c_{i+1}	***s***	**Overflow**
0	0	0	0	0	0
0	0	1	0	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	0

★ Adding Signed Numbers ★

The Overflow Output

Full adder truth table for the sign bit can be extended to include new output which indicates if overflow condition has occurred.

Our task is to populate the OVERFLOW column with corresponding values.

Inputs			outputs		
x	***y***	c_i	c_{i+1}	***s***	**Overflow**
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

1. When operands have ***opposite*** signs, their sum will ***never overflow***.
2. There is ***no overflow***, if:
   - ***both*** operands are ***positive*** and the sum is ***positive***.
   - ***both*** operands are ***negative*** and the sum is ***negative***.
3. When two signed 2's complement numbers are added, ***overflow*** is detected if:
   - ***both*** operands are ***positive*** and the sum is ***negative***, or
   - ***both*** operands are ***negative*** and the sum is ***positive***.

★ Adding Signed Numbers ★

The Overflow Output

Full adder truth table for the sign bit can be extended to include new output which indicates if overflow condition has occurred.

Our task is to populate the OVERFLOW column with corresponding values.

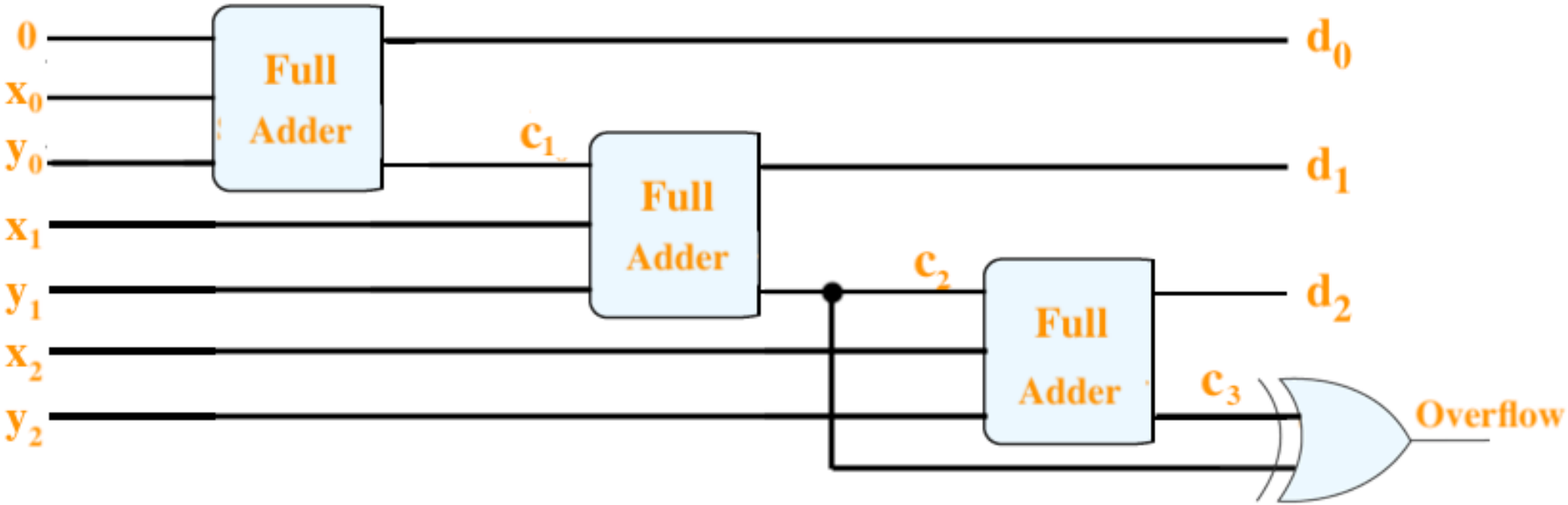
Inputs			outputs		
x	y	c_i	c_{i+1}	s	Overflow
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

1. When operands have ***opposite*** signs, their sum will ***never overflow***.
2. There is ***no overflow***, if:
   - ***both*** operands are ***positive*** and the sum is ***positive***.
   - ***both*** operands are ***negative*** and the sum is ***negative***.
3. When two signed 2's complement numbers are added, ***overflow*** is detected if:
   - ***both*** operands are ***positive*** and the sum is ***negative***, or
   - ***both*** operands are ***negative*** and the sum is ***positive***.

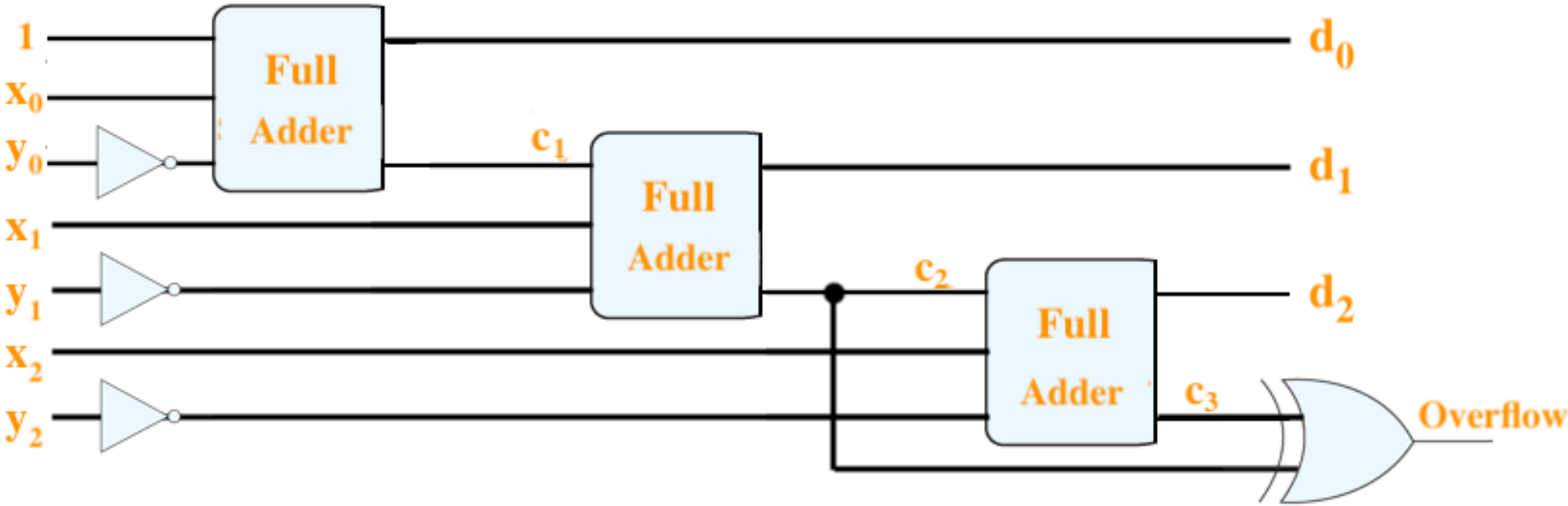
- Notice that overflow occurs only when $c_i \neq c_{i+1}$
- or simply $V = c_i \oplus c_{i+1}$ where V is the overflow signal.

Adding Signed Numbers

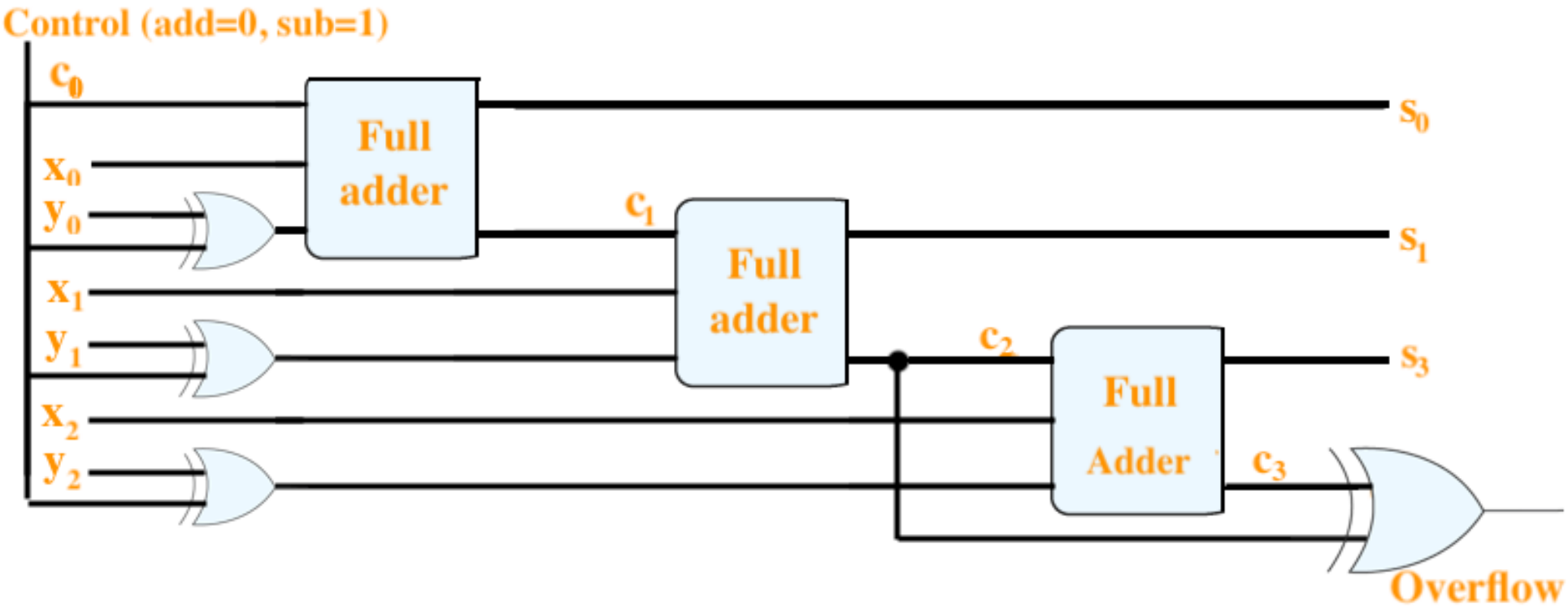
Overflow Detection Circuit for signed Addition



★ Overflow Detection Circuit for 2's Complement Addition ★



★ Overflow Detection Circuit for 2's Complement Addition ★



★ Binary Comparators ★

- Binary comparators, also called digital comparators or logic comparators, are combinational logic circuits that are used for testing whether the value represented by one binary word is greater than, less than, or equal to the value represented by another binary word.
- For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input ***A*** is greater than, smaller than or equal to the value at input ***B*** etc.
- The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean Algebra.
- There are two main types of Digital Comparator available and these are.
  - **Equality or Identity Comparator** : an Identity Comparator is a digital comparator with only one output terminal for when $A = B$, either $A = B = 1$ (HIGH) or $A = B = 0$ (LOW)
  - **Magnitude comparators** : a Magnitude Comparator is a digital comparator which has three output terminals, one each for equality, $A = B$ greater than, $A > B$ and less than $A < B$.

★ Binary Comparators ★

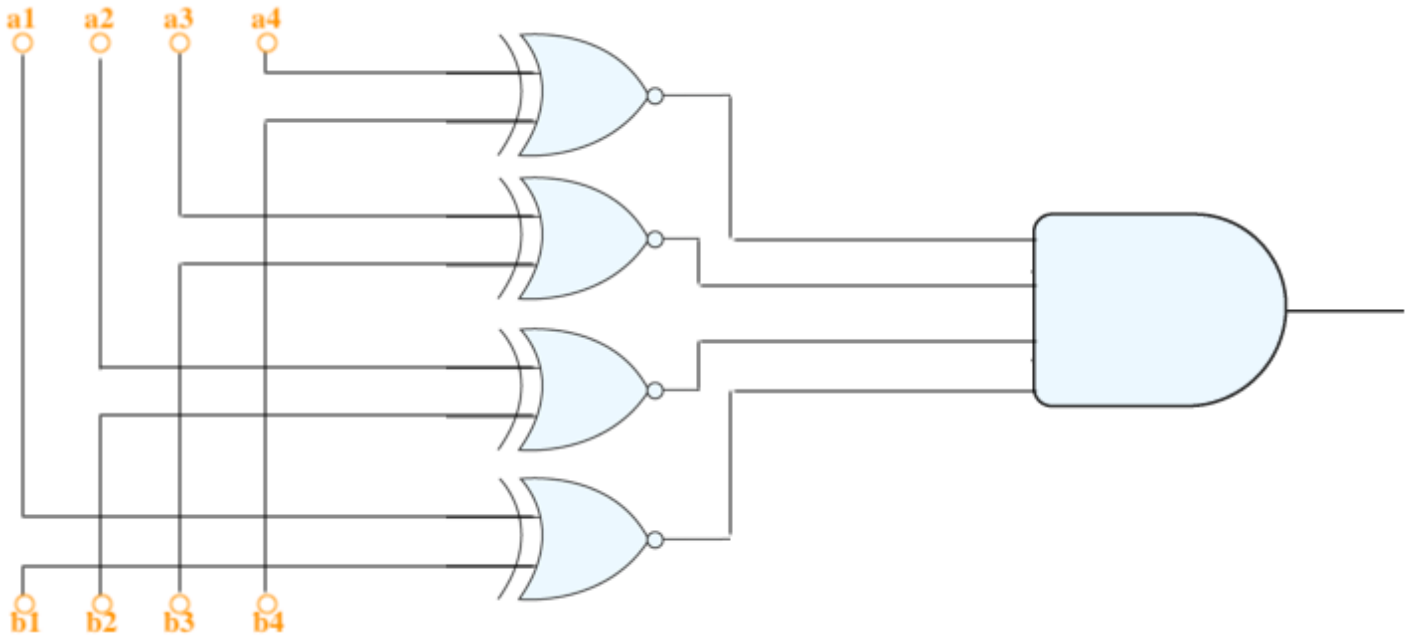
Equality Comparators

- An equality comparator, can be used for such circuits as electronic locks and security devices where a binary password consisting of multiple bits is input to the comparator to be compared with another preset word.
- A logic **1** will be present at the output if the two input words match, otherwise the output remains at **0**.
- Therefore there is only one input combination that is correct, and the more bits the input words possesses, the more possible wrong combinations there are. With extra circuitry for counting, additional security may be provided by limiting the number of tries before the input is inhibited.
- The circuit of the equality comparator consists of an exclusive NOR gate (XNOR) per pair of input bits. If the two inputs are identical (both **1**s or both **0**s) an output of logic **1** is obtained. The outputs of the XNOR gates are then combined in an AND gate, the output of which will be **1**, only when all the XNOR gates indicate matched inputs

★ Binary Comparators ★

Equality Comparators

- The circuit of the equality comparator consists of an exclusive NOR gate (XNOR) per pair of input bits. If the two inputs are identical (both 1s or both 0s) an output of logic 1 is obtained. The outputs of the XNOR gates are then combined in an AND gate, the output of which will be 1, only when all the XNOR gates indicate matched inputs



★ Binary Comparators ★

Magnitude Comparators

- The **magnitude comparator** can also be used to indicate equality, but has a further two outputs, one that is logic 1 when word ***A*** is greater than word ***B***, and another that is logic 1 when word ***A*** is less than word ***B***.

Digital Comparator Truth Table				
inputs		Outputs		
B	A	A>B	A=B	A<B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

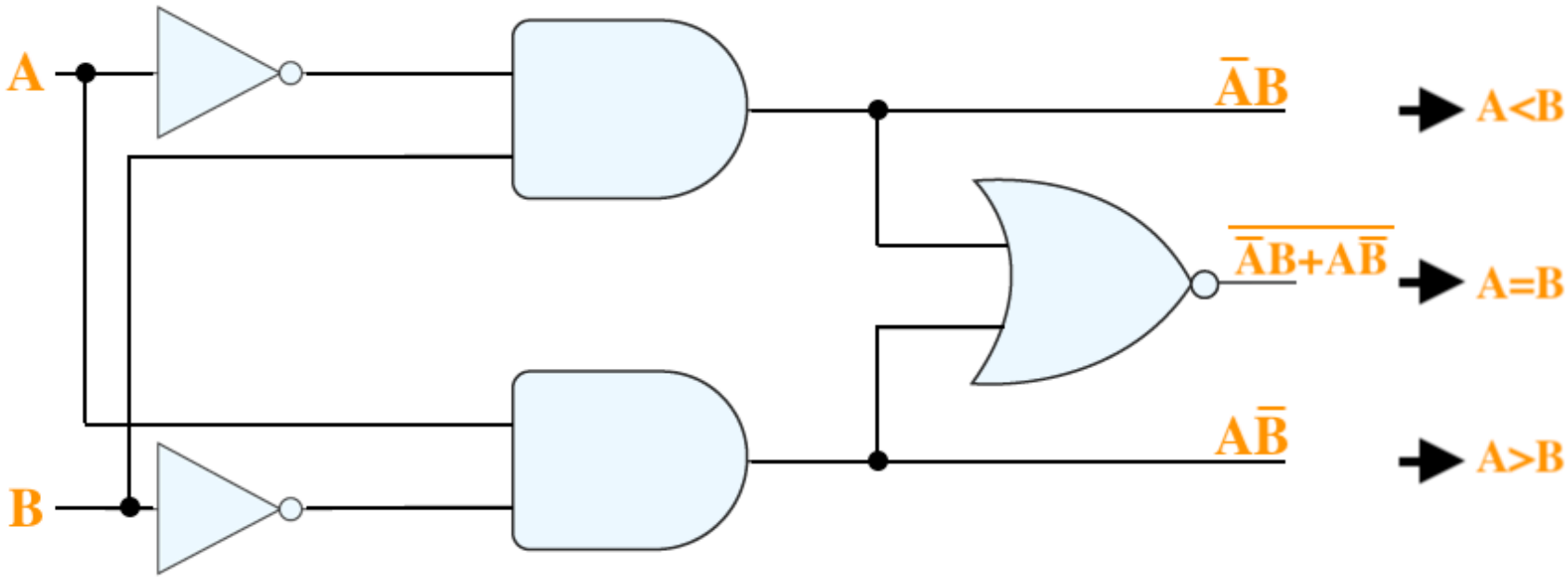
$$\Longrightarrow \quad \begin{array}{lcl} A > B & \Longleftrightarrow & \overline{B} \cdot A \\ A = B & \Longleftrightarrow & \overline{B} \cdot \overline{A} + A \cdot B = A \odot B \\ A < B & \Longleftrightarrow & B \cdot \overline{A} \end{array}$$

- **Digital comparators** actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary values against each other, we are comparing the “magnitude” of these values, a logic “0” against a logic “1” which is where the term Magnitude Comparator comes from.

★ Binary Comparators ★

Magnitude Comparators

- **Digital comparators** actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary values against each other, we are comparing the “magnitude” of these values, a logic “0” against a logic “1” which is where the term Magnitude Comparator comes from.




Thank you! Questions?