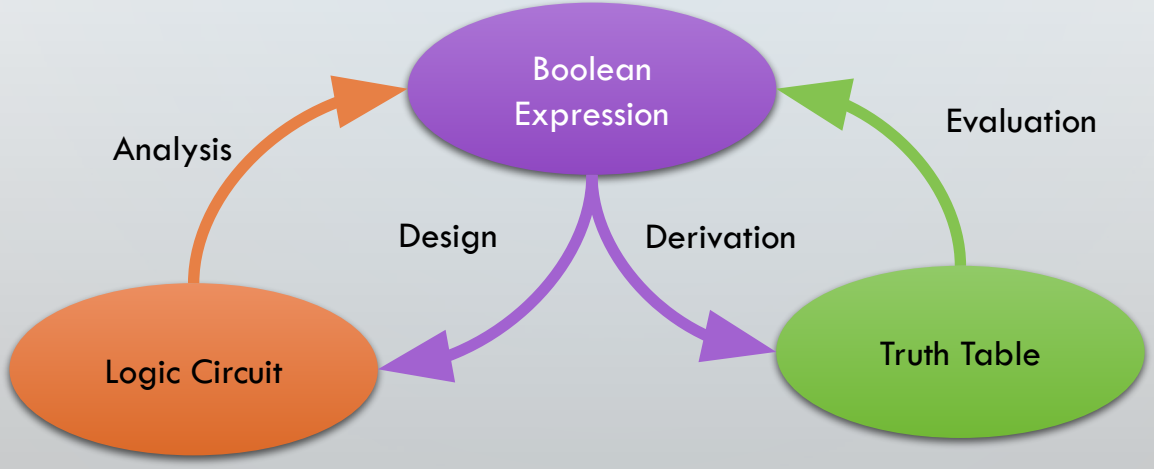
Combinational Circuits Analysis

Dr.-Ing. Nusrat Jahan Lisa
nusrat.lisa@northsouth.edu

Relationships between Expressions, Circuits and Truth Tables



Combinational Circuits

- A combinational circuit consists of a set of logic gates that determine the output values directly from the current input values.
- A combinational circuit is said to perform an information processing operation that can be specified by a set of Boolean equations.
- Combinational circuits are responsible for logical and arithmetic operations within a digital system.
- In addition to logical and arithmetic operations (such as addition, subtraction, complementation, etc.), there are other functions necessary to make connections between the various operators. These functions include multiplexing and decoding. The elements that perform these operations are called multiplexers and decoders, respectively, and are also combinational circuits.

Arithmetic Circuits

- An arithmetic combinational circuit implements arithmetic operations such as addition, subtraction, multiplication, and division with binary numbers.
- The simplest arithmetic operation is the addition of two binary bits, which consists of four possible elementary operations: 0+0 = 0, 0+1 = 1, 1+0 = 1, and 1+1 = 10.
- The first three operations produce a single bit sum. However, when both operands are equal to 1, two bits are needed to express their result. In this case, the carry is added to the next most significant pair of bits.

Arithmetic Circuits

- A combinational circuit that implements two-bit addition is called a Half Adder (HA). A circuit that implements three-bit addition (two significant bits and one carry) is called a Full Adder (FA).
- These names come from the fact that with two half adders a full adder can be implemented. The full adder is the basic arithmetic circuit from which all other arithmetic circuits are built.

Half Adder

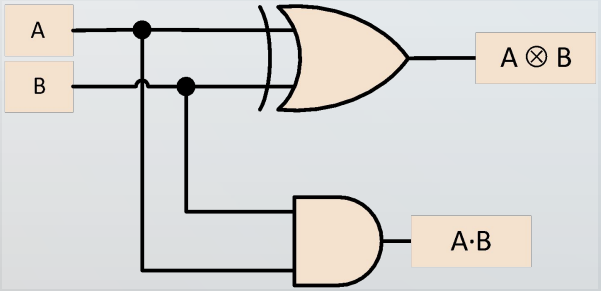
- A half adder takes two input bits and produces two outputs: the sum and the carry-out bit.

A	B	Sum (S)	Carry out (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

When both inputs are equal, the result is zero. When the inputs are different, the result is one. Which logic gate does this?

When one of the inputs is zero, the result is zero.
Which logic gate does this?

Half Adder Circuit




```
module adder (A, B, S, C);

input A, B;
output S, C;

assign S = A ^ B;
assign C = A & B;

endmodule
```

Full Adder

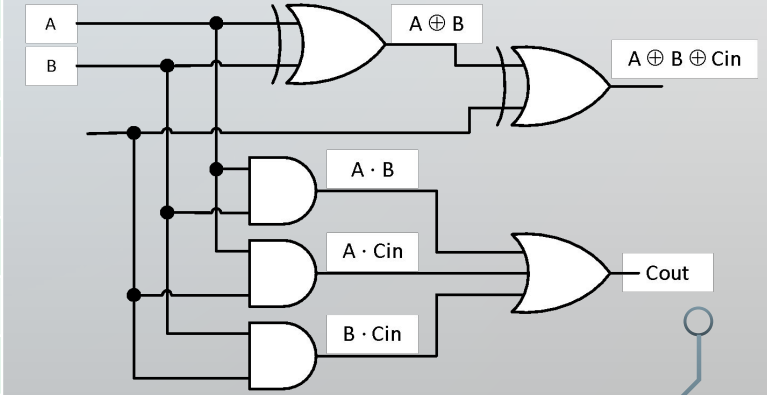
- The half adder, as the name suggests, is capable of adding only two bits.
- However, we need a circuit capable of adding three bits (A, B and Cin), generating the result (S) and the carry out (Cout).
- This circuit is called a full adder.

Full Adder

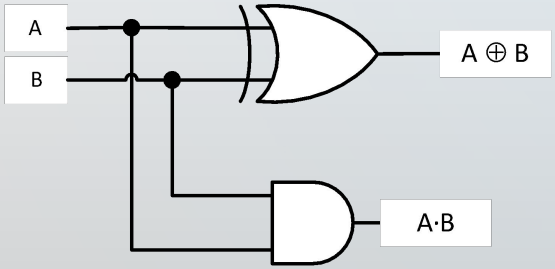
- Let's look at the following table:

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

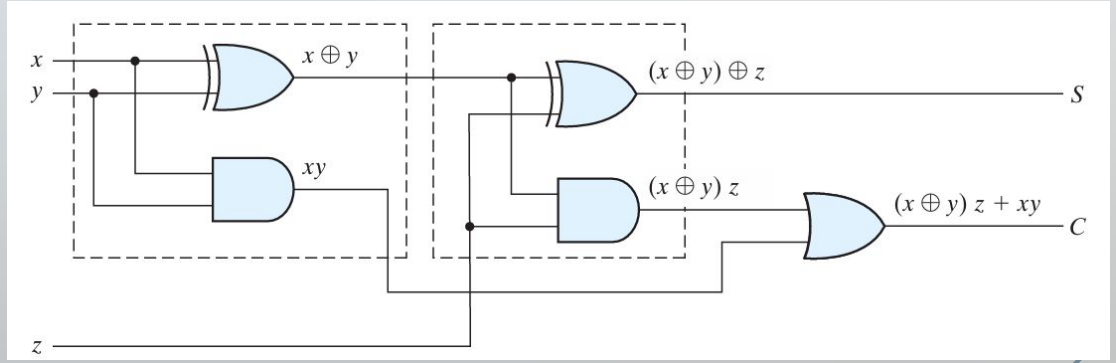
Note that when at least two of the three inputs are 1, Cout will be 1.



Full Adder Circuit



Half Adder



Full Adder

ADDER: VERILOG PROGRAMMING

```
module  fulladd (Cin, x, y, s, Cout);
    input  Cin, x, y;
    output  s, Cout;

    assign  s = x ^ y ^ Cin;
    assign  Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```

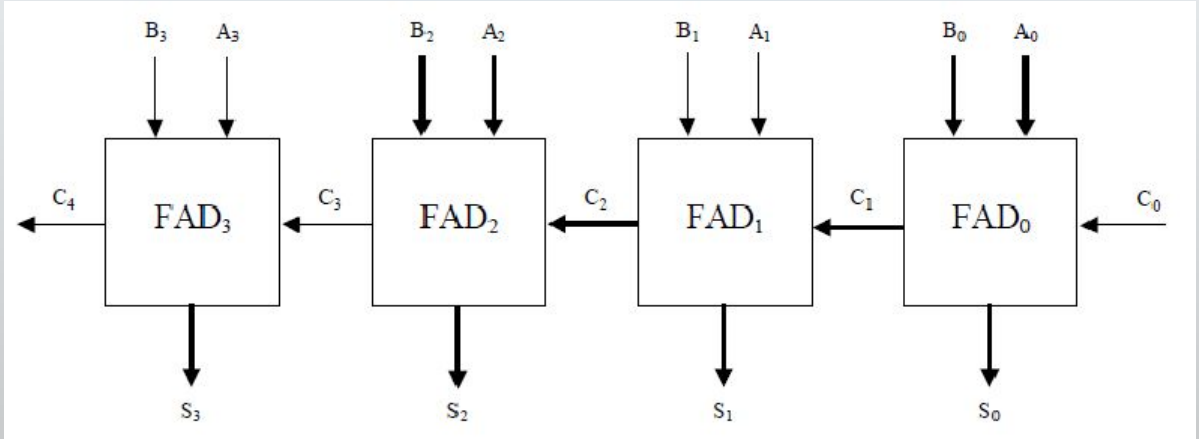
```
module  fulladd (Cin, x, y, s, Cout);
    input  Cin, x, y;
    output  s, Cout;

    assign   s = x ^ y ^ Cin,
             Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```

ADDER: VERILOG PROGRAMMING

```
module  fulladd (Cin, x, y, s, Cout);
    input  Cin, x, y;
    output  s, Cout;

    assign  s = x ^ y ^ Cin,
            Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```

```
module full_adder_tb;
reg a, b, cin;
wire sum, carry;

fulladd uut(cin, a, b, sum, carry);

initial begin
a = 0; b = 0; cin = 0;
#10
a = 0; b = 0; cin = 1;
#10
a = 0; b = 1; cin = 0;
#10
a = 0; b = 1; cin = 1;
#10
end

endmodule
```

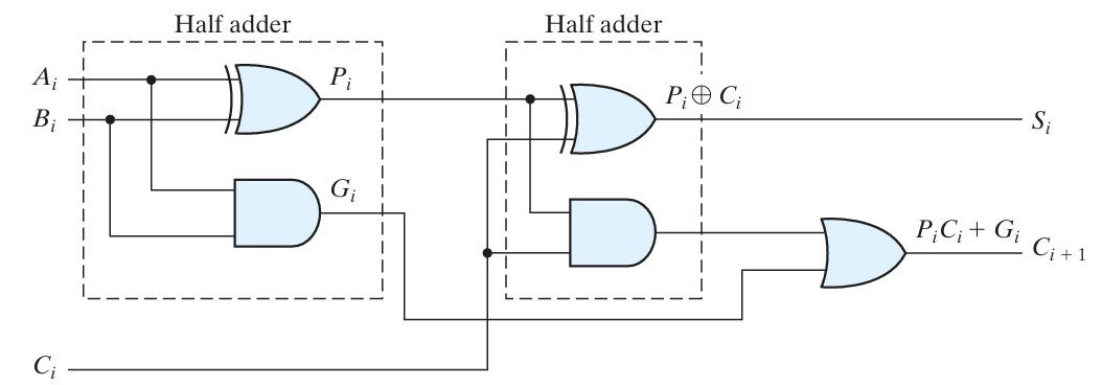
4-bit Parallel Adder



4-bit Parallel Adder: Verilog programming

```
module  adder4 (carryin, x3, x2, x1, x0, y3, y2, y1, y0, s3, s2, s1, s0, carryout);
    input  carryin, x3, x2, x1, x0, y3, y2, y1, y0;
    output  s3, s2, s1, s0, carryout;

    fulladd  stage0  (carryin, x0, y0, s0, c1);
    fulladd  stage1  (c1, x1, y1, s1, c2);
    fulladd  stage2  (c2, x2, y2, s2, c3);
    fulladd  stage3  (c3, x3, y3, s3, carryout);

endmodule

module  fulladd (Cin, x, y, s, Cout);
    input  Cin, x, y;
    output  s, Cout;

    assign  s = x ^ y ^ Cin;
    assign  Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```

Carry Propagation



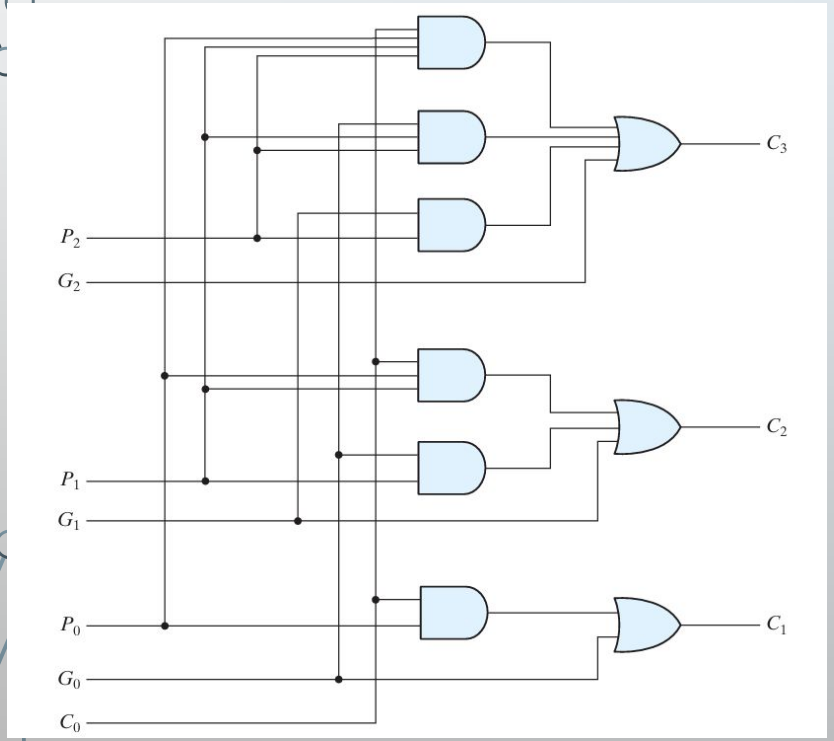
$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_iC_i$$

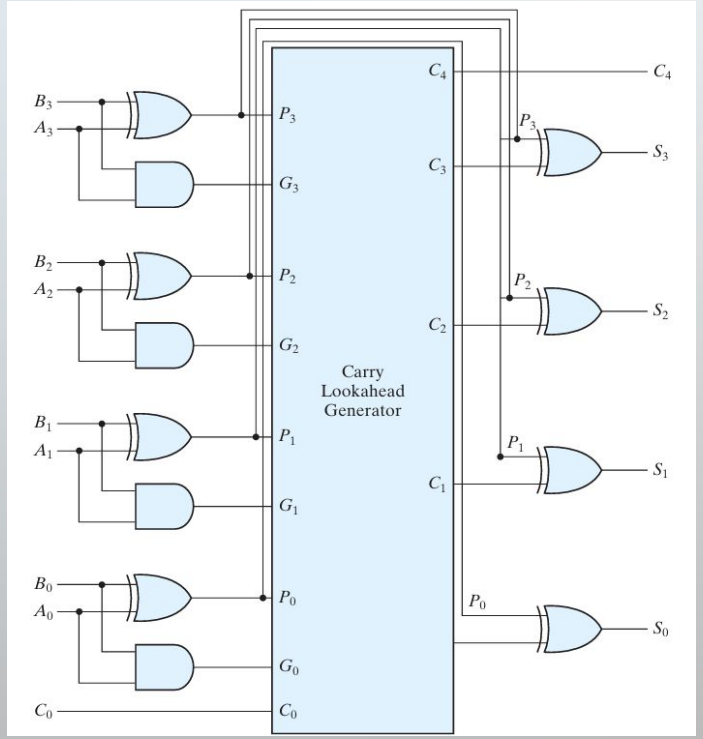
$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

Carry Lookahead Adder

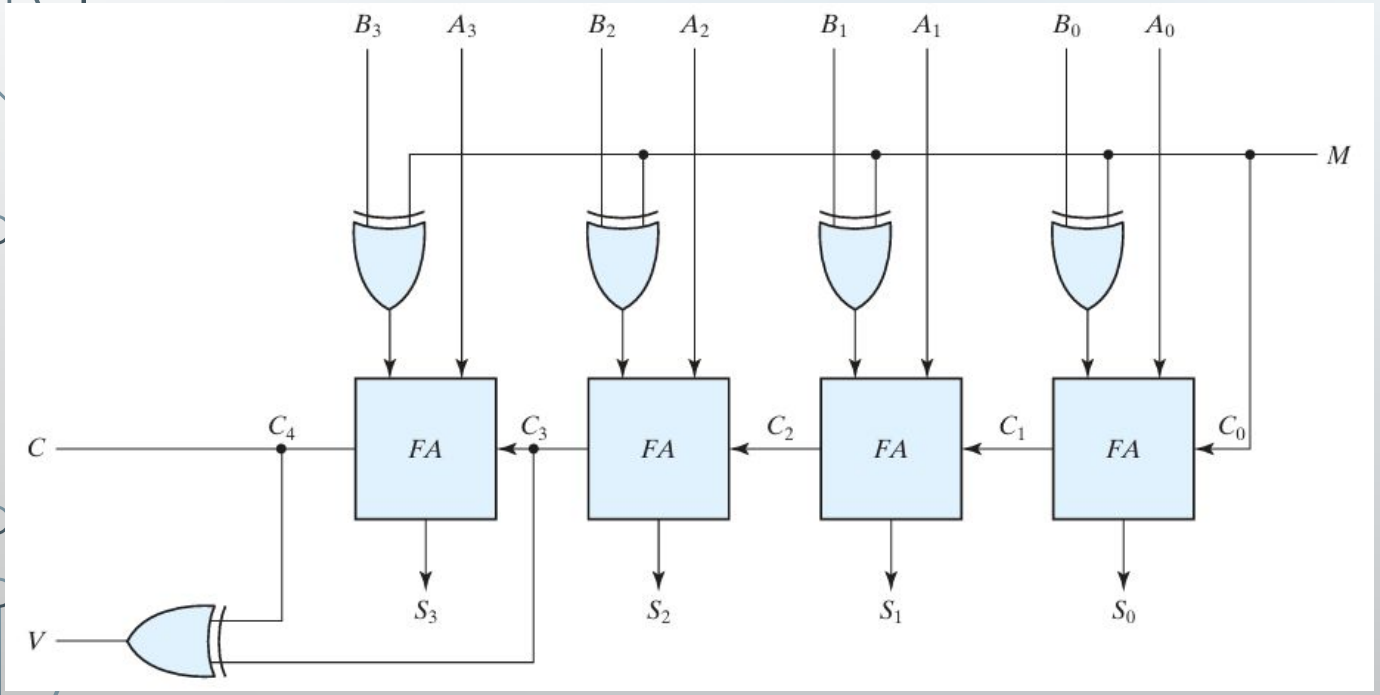


carry lookahead generator



Four-bit adder with carry lookahead

FOUR-BIT ADDER–SUBTRACTOR

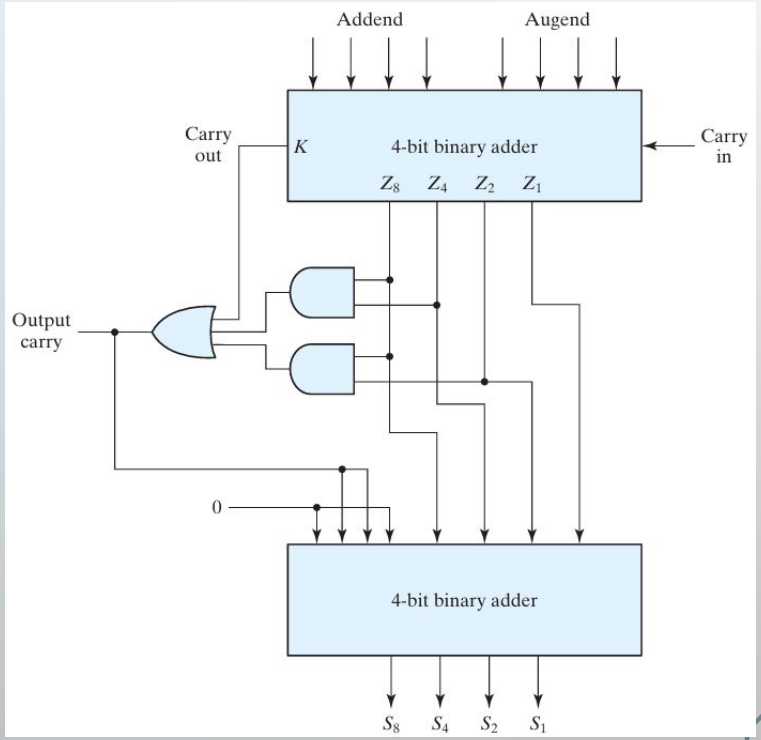


carries:	Final carry → 0	1	carries:	1	0
+70		0 1000110	−70		1 0111010
+80		0 1010000	−80		1 0110000
+150	Sign bit → 1	0010110	−150		0 1101010

BCD Adder

Derivation of BCD Adder

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



$$C = K + Z_8Z_4 + Z_8Z_2$$

Binary Multiplier

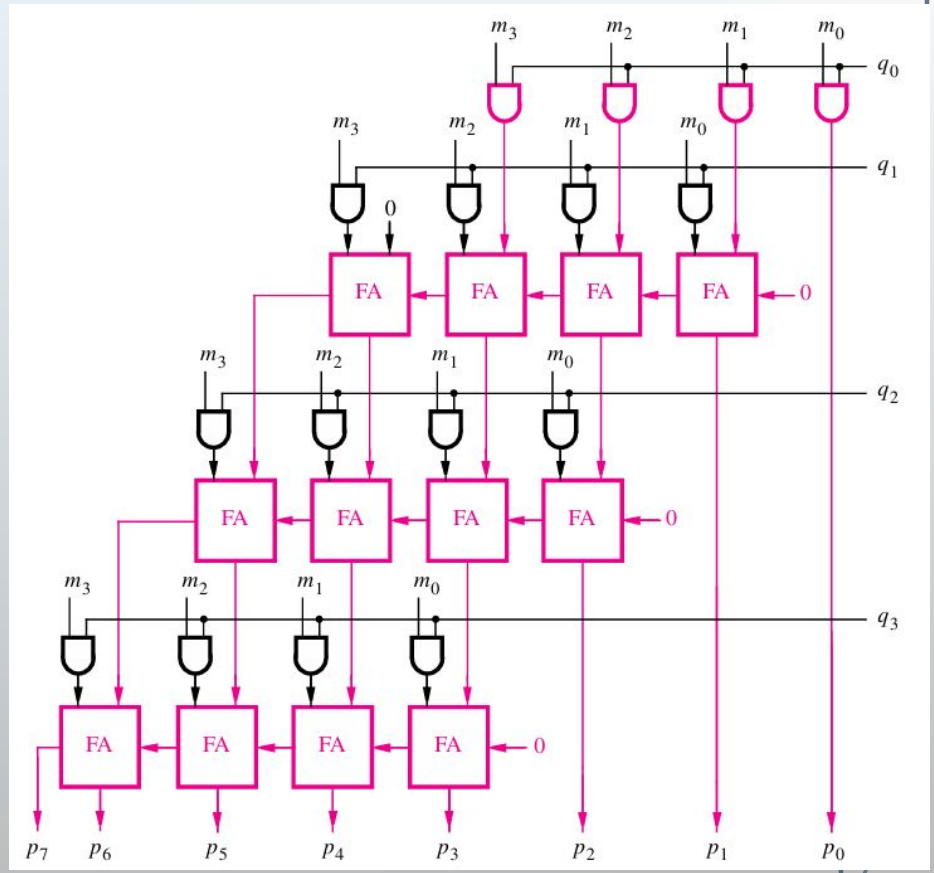
Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	× 1 0 1 1
		1 1 1 0
		1 1 1 0
		0 0 0 0
		1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	× 1 0 1 1
Partial product 0		1 1 1 0
		+ 1 1 1 0
Partial product 1		1 0 1 0 1
		+ 0 0 0 0
Partial product 2		0 1 0 1 0
		+ 1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

			m_3	m_2	m_1	m_0	
		×	q_3	q_2	q_1	q_0	
Partial product 0			m_3q_0	m_2q_0	m_1q_0	m_0q_0	
	+	m_3q_1	m_2q_1	m_1q_1	m_0q_1		
Partial product 1	$PP1_5$	$PP1_4$	$PP1_3$	$PP1_2$	$PP1_1$		
	+	m_3q_2	m_2q_2	m_1q_2	m_0q_2		
Partial product 2	$PP2_6$	$PP2_5$	$PP2_4$	$PP2_3$	$PP2_2$		
	+	m_3q_3	m_2q_3	m_1q_3	m_0q_3		
Product P	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

4×4 Multiplier

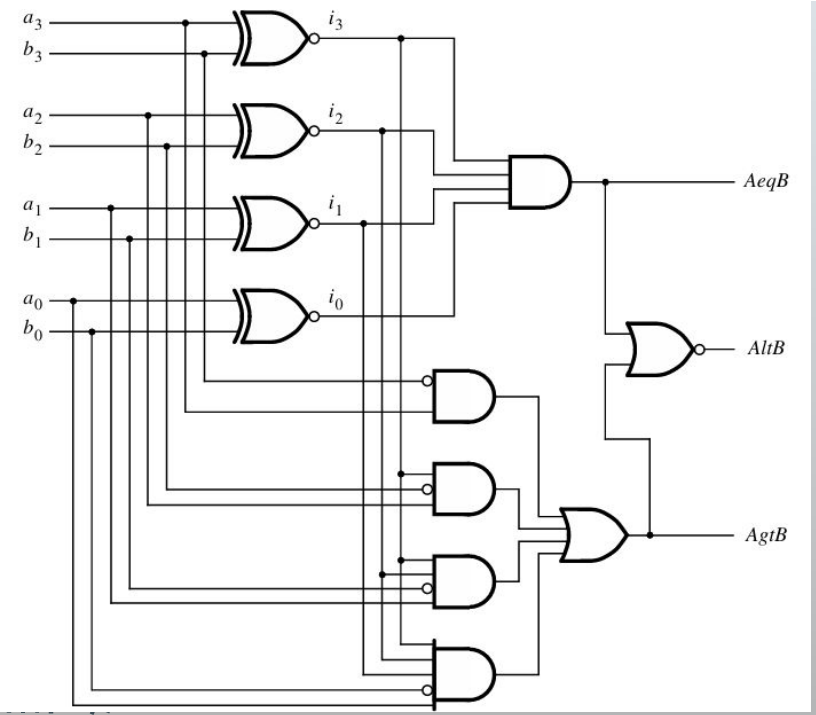
				m_3	m_2	m_1	m_0	
			×	q_3	q_2	q_1	q_0	
Partial product 0				m_3q_0	m_2q_0	m_1q_0	m_0q_0	
		+	m_3q_1	m_2q_1	m_1q_1	m_0q_1		
Partial product 1		$PP1_5$	$PP1_4$	$PP1_3$	$PP1_2$	$PP1_1$		
	+	m_3q_2	m_2q_2	m_1q_2	m_0q_2			
Partial product 2	$PP2_6$	$PP2_5$	$PP2_4$	$PP2_3$	$PP2_2$			
+	m_3q_3	m_2q_3	m_1q_3	m_0q_3				
Product P	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0



Magnitude Comparator

Let $A = a_3a_2a_1a_0$ and $B = b_3b_2b_1b_0$. Define a set of intermediate signals called i_3, i_2, i_1, and i_0. Each signal, i_k, is 1 if the bits of A and B with the same index are equal. That is, $i_k = \overline{a_k \oplus b_k}$. The comparator's $AeqB$ output is then given by

$$AeqB = i_3i_2i_1i_0$$



If $a_k = 0$ and $b_k = 1$, then A < B.
But if $a_k = 1$ and $b_k = 0$, then A > B.

$$AgtB = a_3\bar{b}_3 + i_3a_2\bar{b}_2 + i_3i_2a_1\bar{b}_1 + i_3i_2i_1a_0\bar{b}_0$$

$$AltB = \overline{AeqB + AgtB}$$