Fundamentals of Digital System Design

Combinational Logic Circuits Part II

Uvindu Kodikara

Adders

- **Half Adder**:
  - A combinational circuit that performs the addition of two bits is called a half adder.

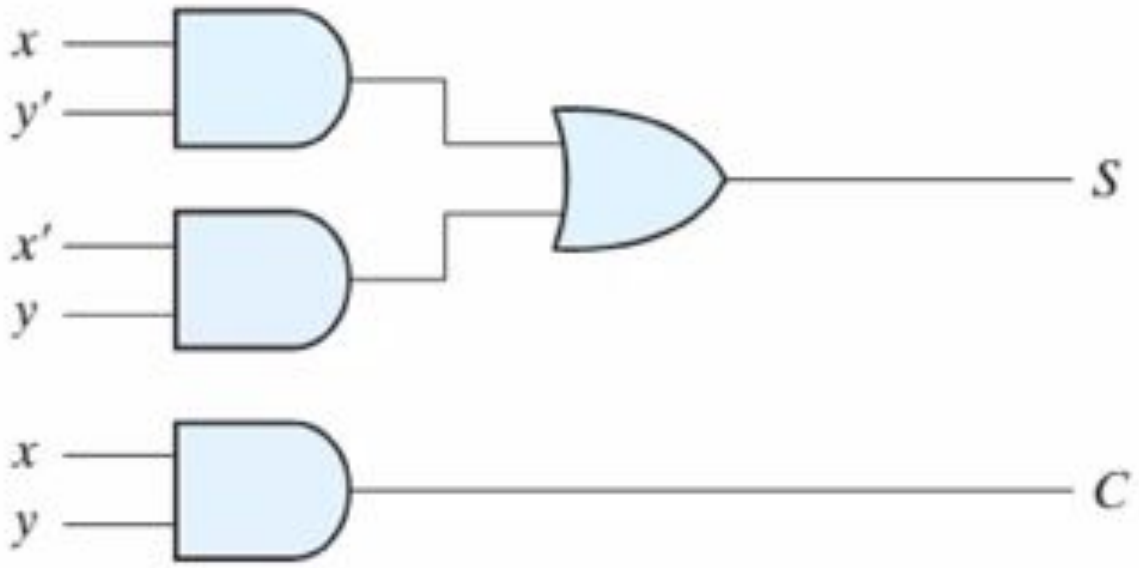
- **Full Adder**:
  - One that performs the addition of three bits (two significant bits and a previous carry) is a full adder.

Half Adder

- We assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs.

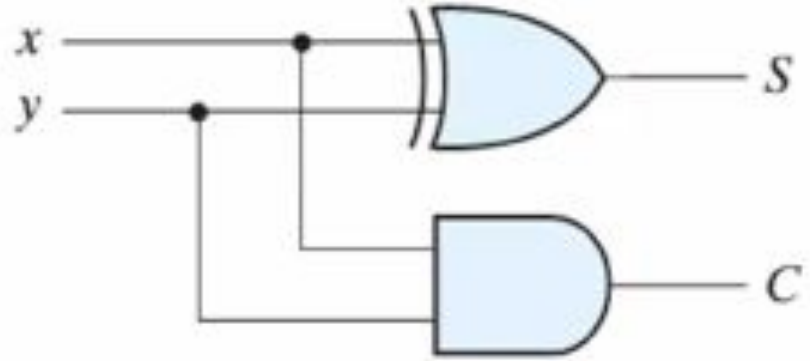
x	y	S	C

Implementation of half adder



$S = x'y + xy'$

$C = xy$



$S = x \oplus y$

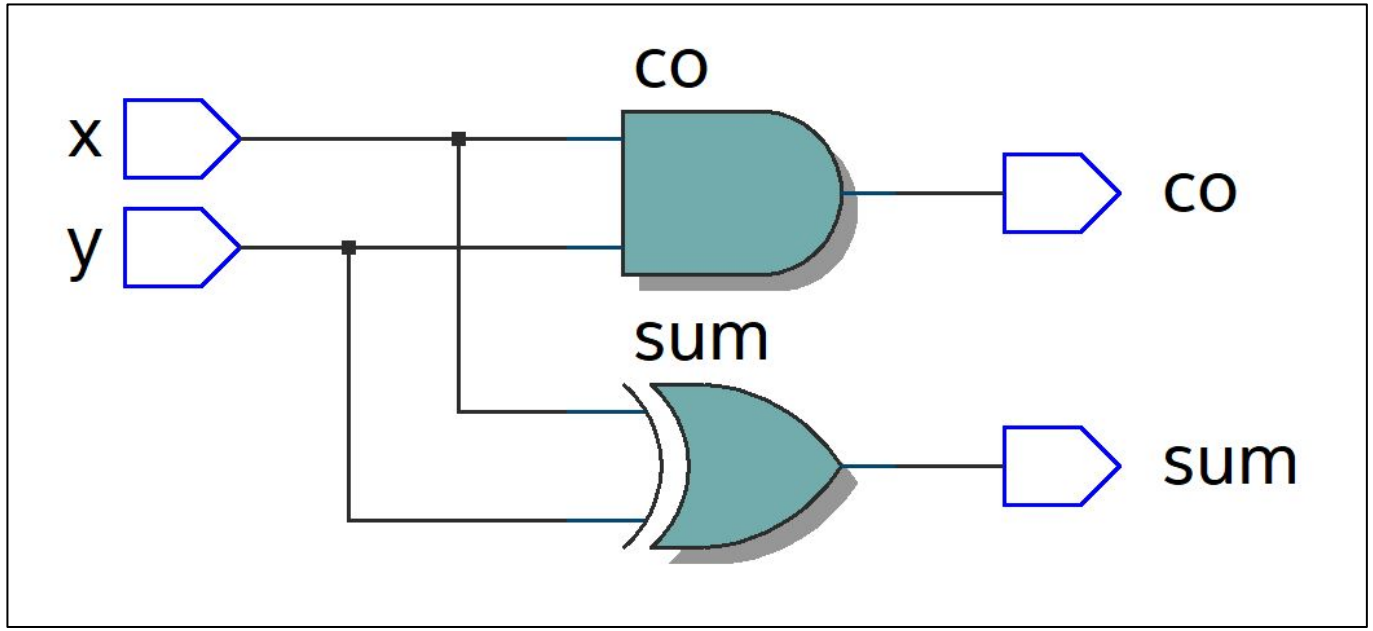
$C = xy$

Half Adder

design.sv

```
1  // This code describes a half_adder circuit
2  module half_adder
3  (
4      input logic x, y,
5      output logic sum, co
6  );
7
8      always_comb begin
9          // AND gate between x and y
10         co = x & y;
11
12         // XOR gate between x and y
13         sum = x ^ y ;
14     end
15
16 endmodule
```

Half-Adder Gate Level Modelling



RTL View

Half Adder Test Bench

testbench.sv

```
module half_adder_tb;

    logic x;
    logic y;
    logic sum;
    logic co;

    // Instantiate the half_adder module
    half_adder dut (
        .x(x),
        .y(y),
        .sum(sum),
        .co(co)
    );

    // Testbench procedure
    initial begin
        // Setup the VCD dump
        $dumpfile("dump.vcd");   // Specify the name of the VCD file
        $dumpvars(0, dut); // Dump all variables in the half_adder

        // Test case 1: x = 0, y = 0
        #10 x = 0; y = 0;

        // Test case 2: x = 0, y = 1
        #10 x = 0; y = 1;

        // Test case 1: x = 1, y = 0
        #10 x = 1; y = 0;

        // Test case 1: x = 1, y = 1
        #10 x = 1; y = 1;

        #10 $finish; // End simulation
    end

endmodule
```

Wave Forms

https://www.edaplayground.com/

Full Adder

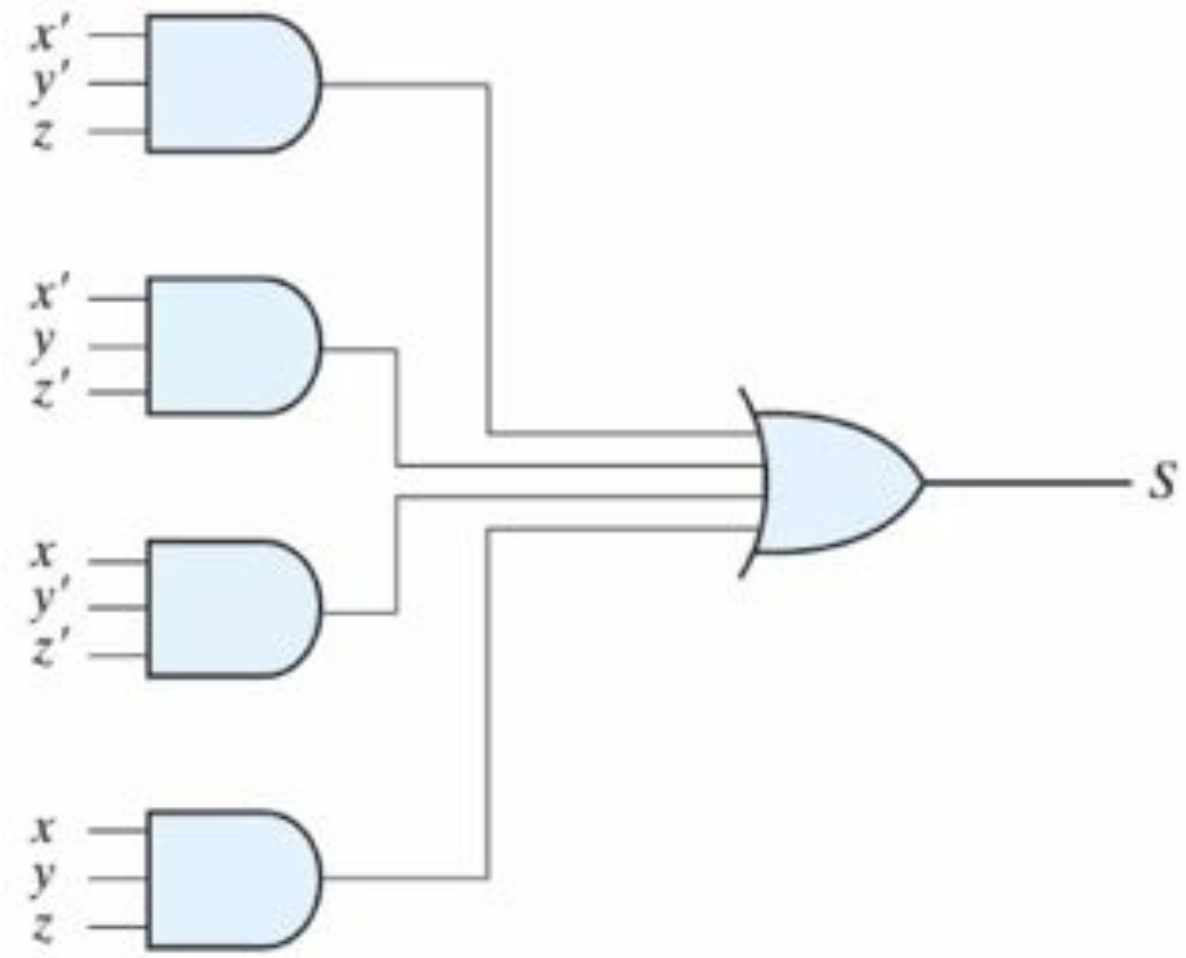
- It consists of three inputs and two outputs.

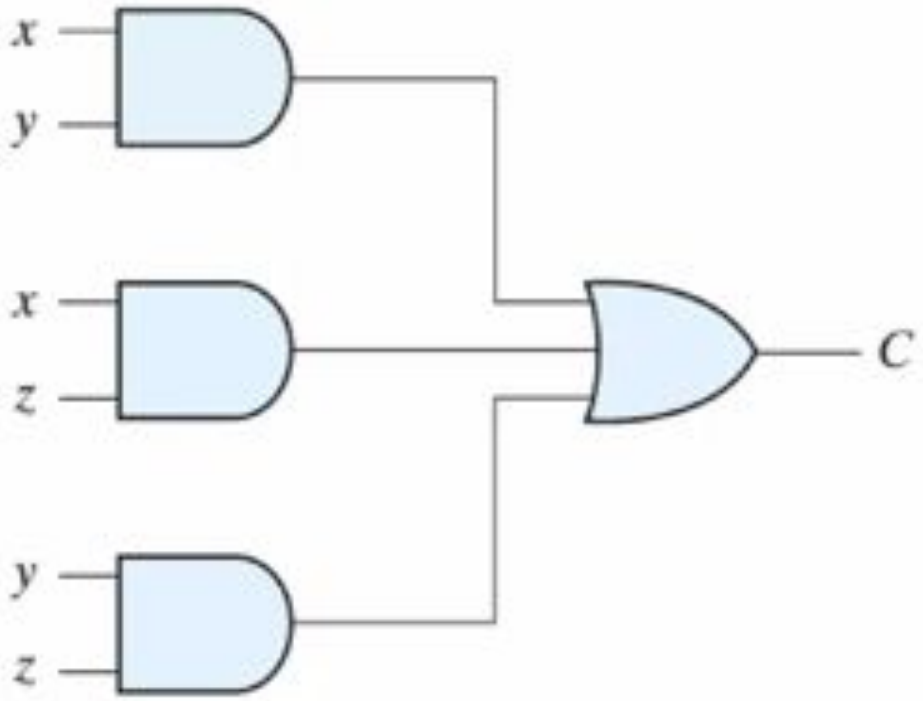
- Two of the input variables, denoted by x and y, represent the two significant bits to be added. The third input, z, represents the carry from the previous lower significant position.

Full Adder

x	y	z	S	C

Implementation of full adder

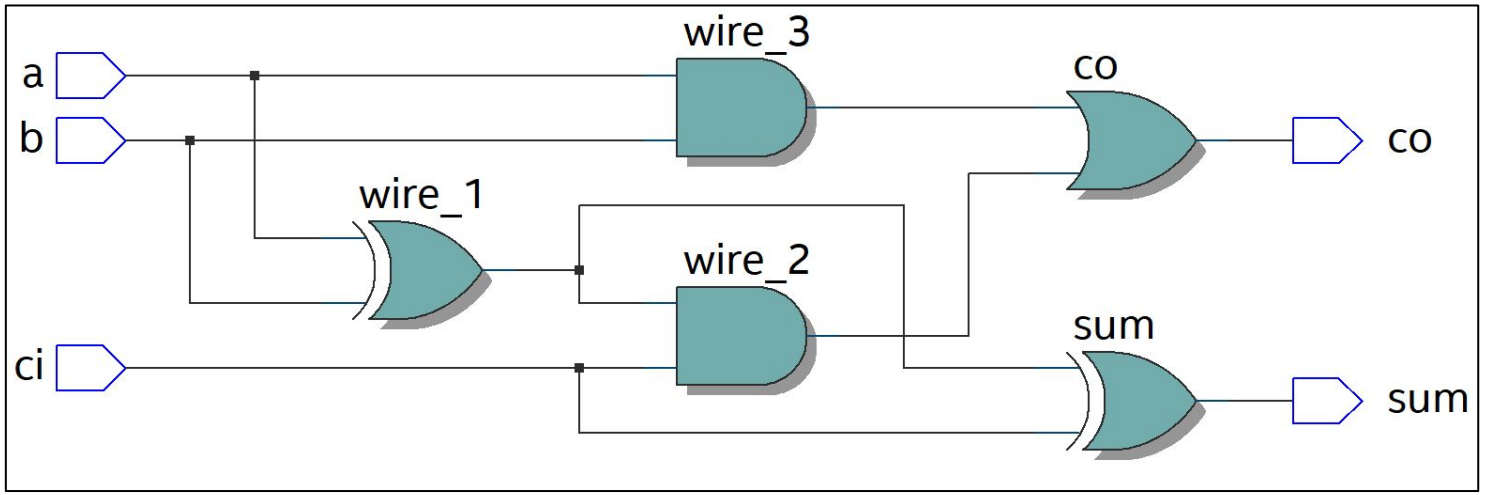




Full Adder

design.sv

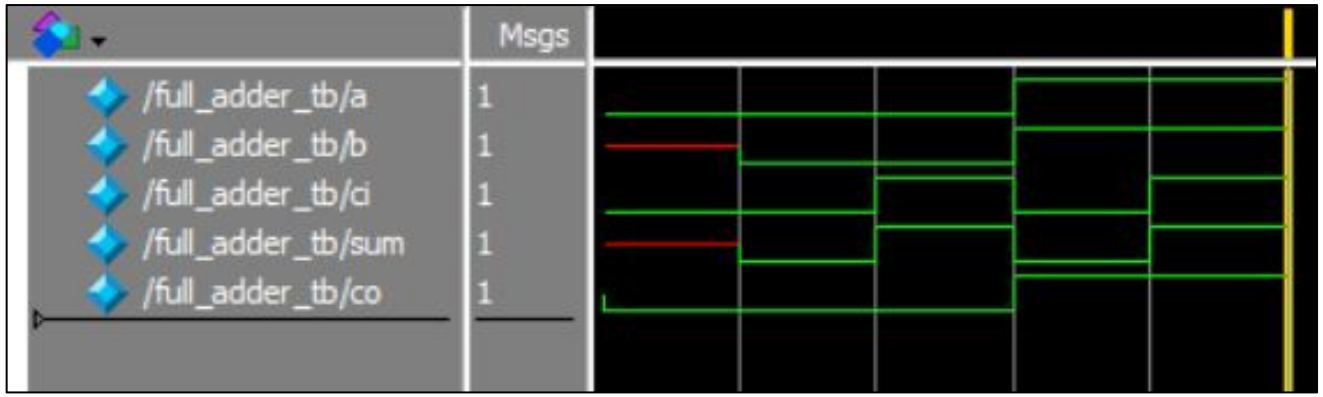
```
1  // This code describes a full_adder circuit
2  module full_adder
3  (
4      input logic a, b, ci,
5      output logic sum, co
6  );
7
8      logic wire_1, wire_2, wire_3;
9
10     always_comb begin
11         //Intermediate wires
12         wire_1 = a ^ b;
13         wire_2 = wire_1 & ci;
14         wire_3 = a & b;
15
16         co = wire_2 | wire_3; // bitwise OR
17         sum = wire_1 ^ ci ; // bitwise XOR
18     end
19
20 endmodule
```



Full Adder Test Bench

testbench.sv

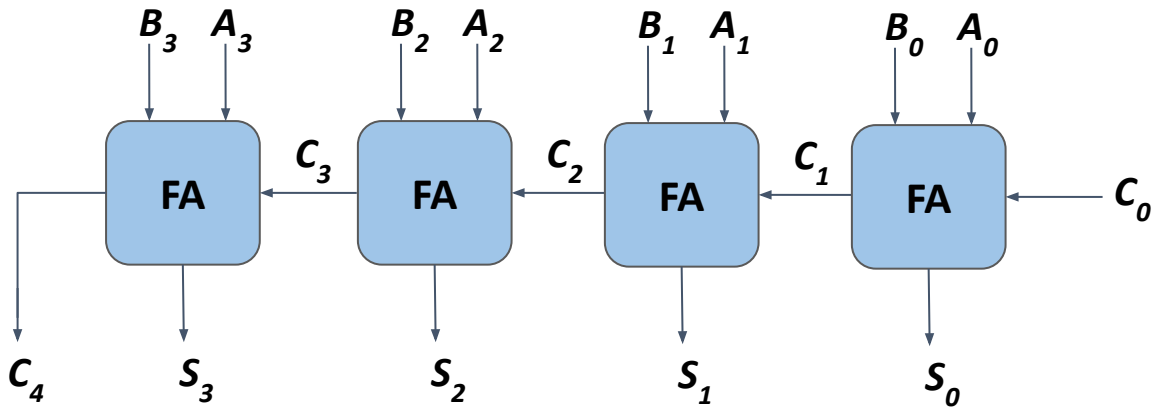
```
1  module full_adder_tb;
2
3  logic a=0, ci=0, b, sum, co;
4
5  full_adder dut (.*);
6
7  initial begin // simulation starts
8      $dumpfile("dump.vcd"); $dumpvars(0, dut);
9
10     #10 a <= 0; b <= 0; ci <= 0;
11     #10 a <= 0; b <= 0; ci <= 1;
12     #10 a <= 1; b <= 1; ci <= 0;
13     #10 a <= 1; b <= 1; ci <= 1;
14
15     #10 $finish(); // simulation ends
16 end
17
18 endmodule
```



Ripple Carry Adder

- A ripple carry adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n-1 full adders.

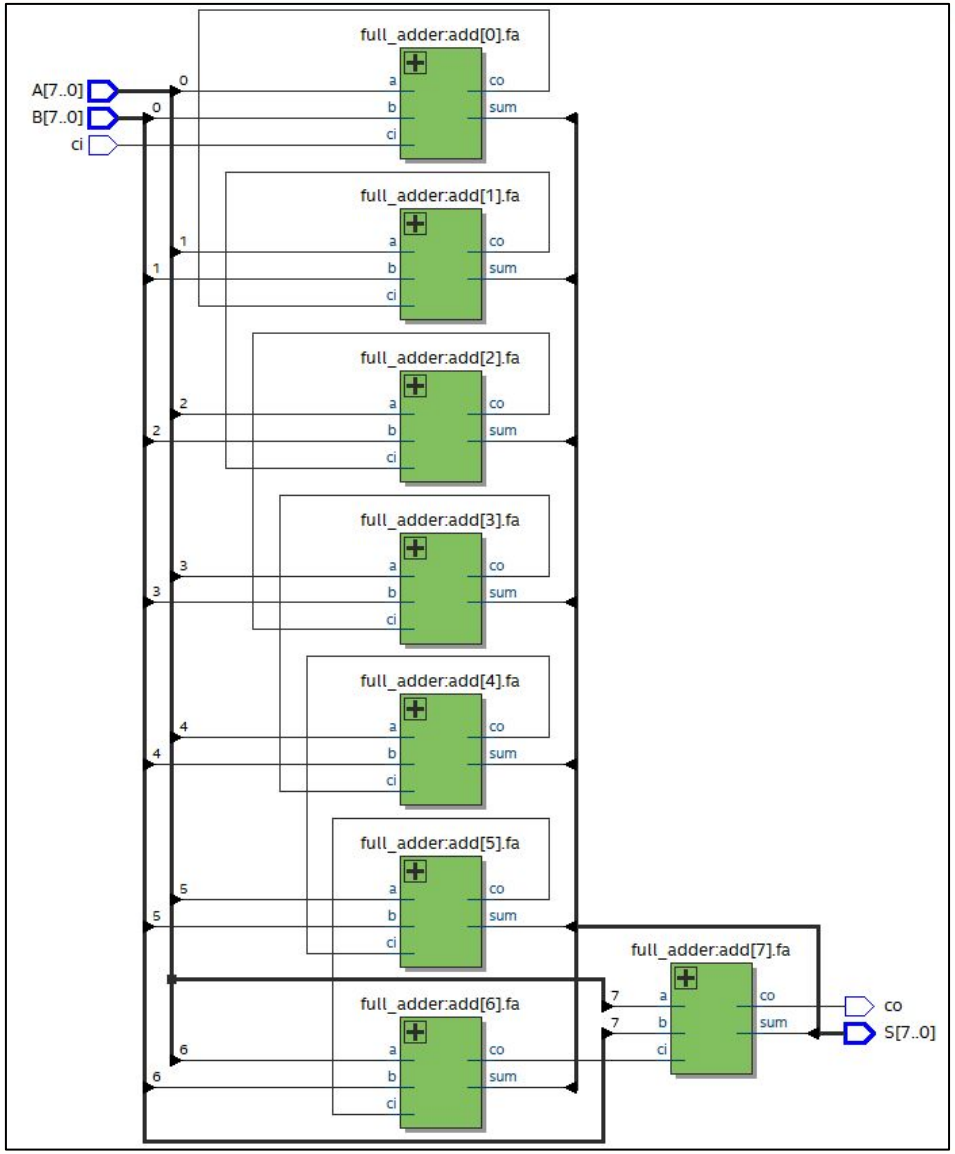
Four Bit Adder



Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

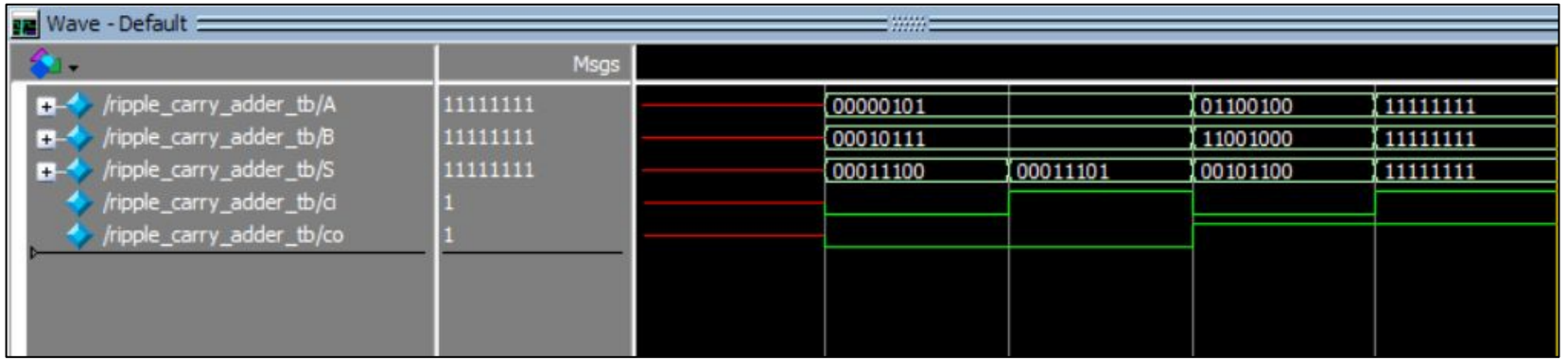
Ripple Carry Adder

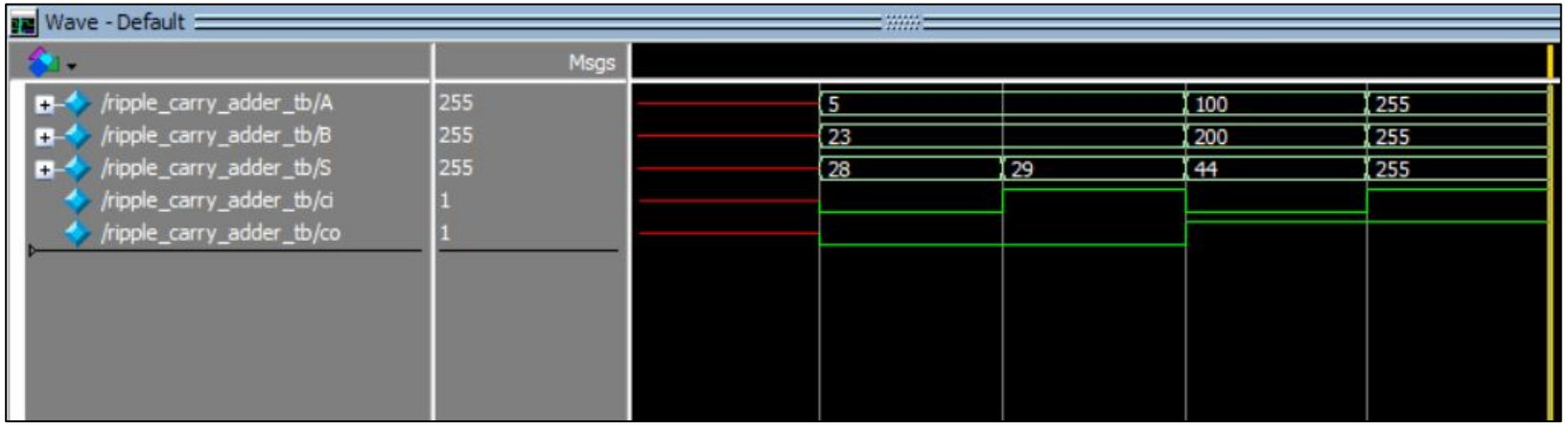
```
module ripple_carry_adder #(parameter N=8)
(
    input logic [N-1:0] A, B,
    input logic ci,
    output logic [N-1:0] S,
    output logic co
);

    logic C [N:0];
    always_comb begin
        C[0] = ci;
        co = C[N];
    end

    genvar i;
    generate
        for (i=0; i<N; i=i+1) begin:add
            // full_adder fa (A[i],B[i],C[i],C[i+1],S[i]);
            full_adder fa (
            .a (A[i]),
            .b (B[i]),
            .ci (C[i]),
            .co (C[i+1]),
            .sum (S[i])
            );
        end
    endgenerate
endmodule
```



Ripple Carry Adder TB

```
1  module ripple_carry_adder_tb;
2
3  localparam N = 8;
4  logic [N-1:0] A, B, S;
5  logic ci, co;
6
7  ripple_carry_adder #(.N(N)) dut (.*);
8
9  initial begin
10     $dumpfile("dump.vcd");
11     $dumpvars(0, dut);
12
13     #5 A <= 8'd5; B <= 8'd23; ci <= 0;
14     #5 ci <= 1;
15     #5 A <= 8'd100; B <= 8'd200; ci <= 0;
16     #5 A <= 8'd255; B <= 8'd255; ci <= 1;
17
18     #5 $finish();
19 end
20
21 endmodule
```





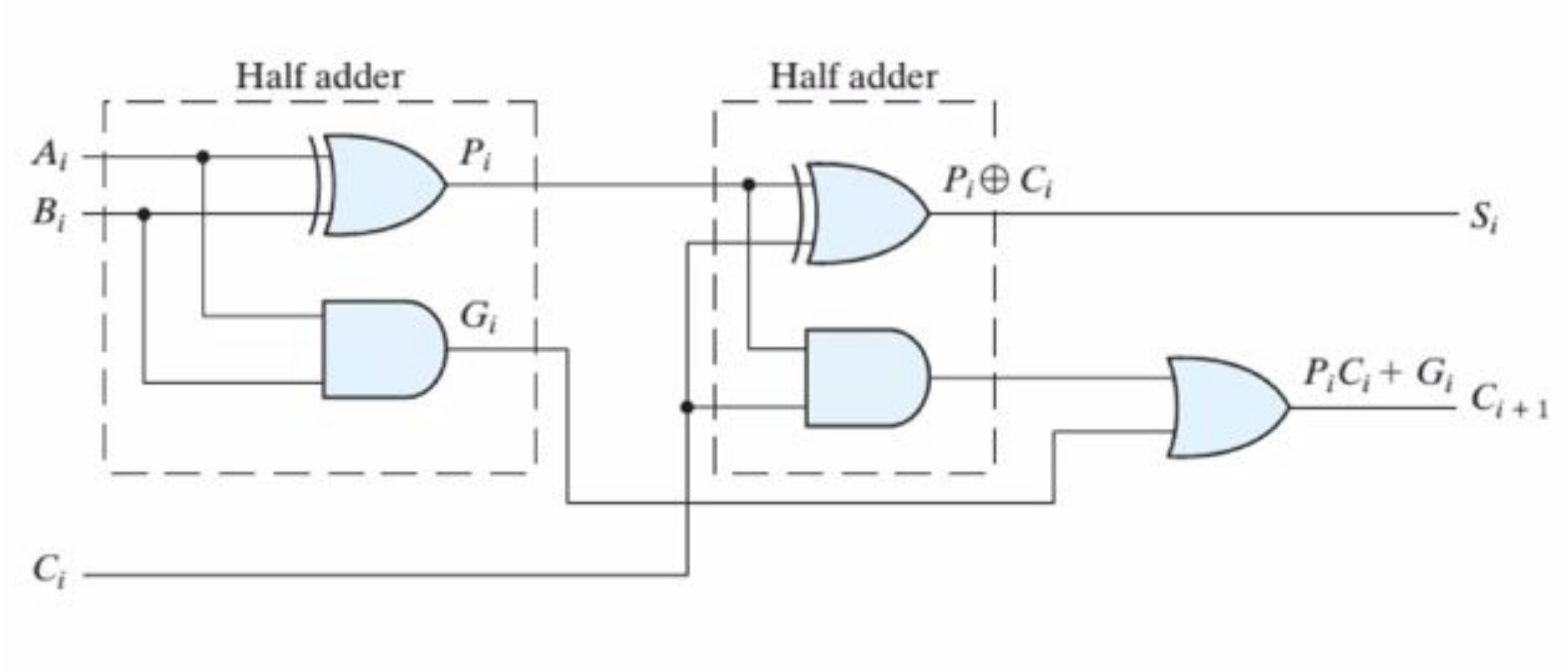
Carry Propagation

- In any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals.
- The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.
- The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.

Carry Propagation

- The signal from the input carry C_i to the output carry C_{i+1} propagates through an AND gate and an OR gate, which constitute two gate levels.
- If there are four full adders in the adder, the output carry C_4 would have **2 x 4 = 8** gate levels from C_0 to C_4. For an n-bit adder, there are **2n** gate levels for the carry to propagate from input to output.
- The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added.
- Outputs will not be correct unless the signals are given enough time to propagate through the gates connected from the inputs to the outputs.

Full adder with Carry Lookahead



Carry Lookahead logic

- Solution for reducing the carry propagation delay time
  - The most widely used technique employs the principle of carry lookahead logic.
- For above circuit,
  - ***Gi*** is called carry generate → it produces a carry of 1 when both ***Ai*** and ***Bi*** are 1, regardless of the input carry ***Ci***.
  - ***Pi*** is called a carry propagate → because it determines whether a carry into stage ***i*** will propagates into stage ***i + 1.***
- With new binary variables:

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

The output sum and carry can respectively be expressed as

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

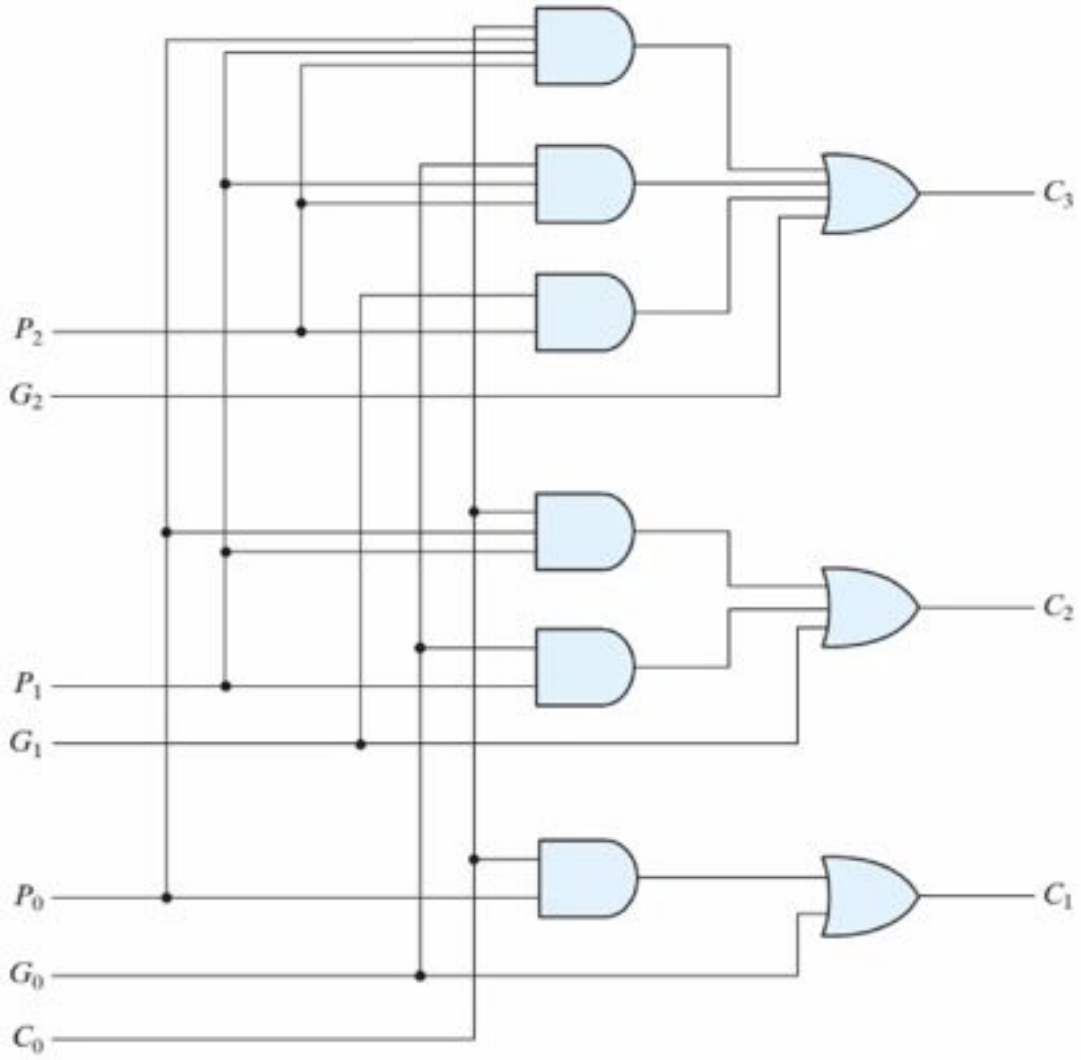
Carry Lookahead logic

- Boolean functions for the carry outputs of each stage and substitute the value of each Ci from the previous equations:

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

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

Logic diagram of Carry Lookahead Generator



Binary to BCD Conversion

- To represent binary values in decimal digits, *useful in display systems such as digital clocks and calculators*.

Example:

- **Binary Input**: 1011 (Binary for 11)
- **BCD Output**: 0001 0001 (1 and 1 representing 11 in BCD format)

Steps:

- Convert the binary number to decimal.
- Represent each decimal digit in 4-bit binary (BCD).

BCD to Binary Conversion

- Useful for systems that store data in BCD but perform internal computations in binary.

Example:

- **BCD Input**: 0001 0001 (BCD for 11)
- **Binary Output**: 1011 (Binary for 11)

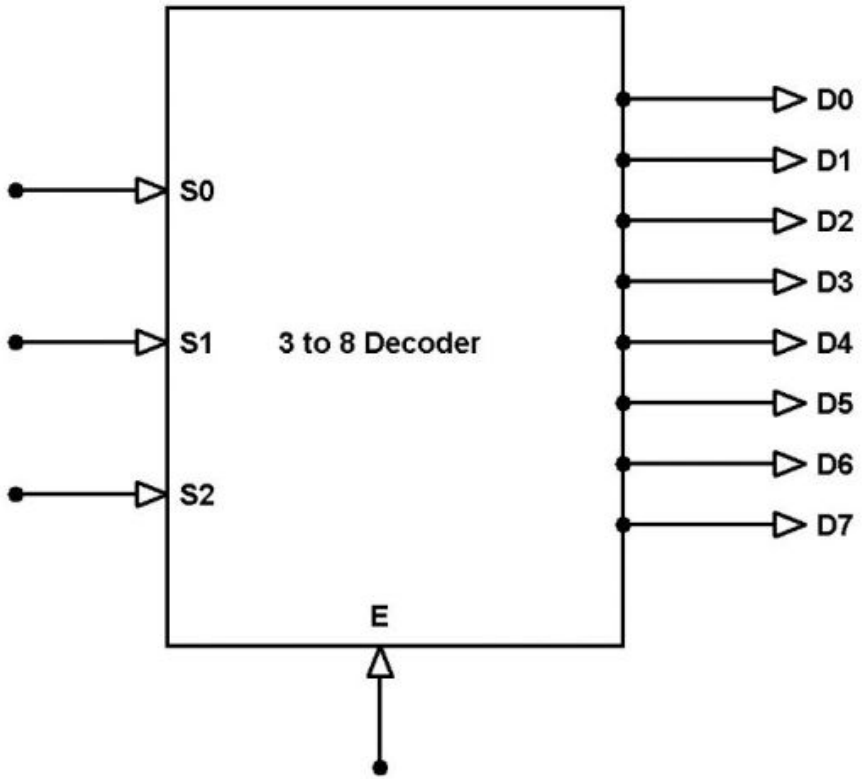
Steps:

- Each group of 4 BCD bits is converted to its equivalent decimal digit.
- The decimal digits are then converted back to binary.

Decoders

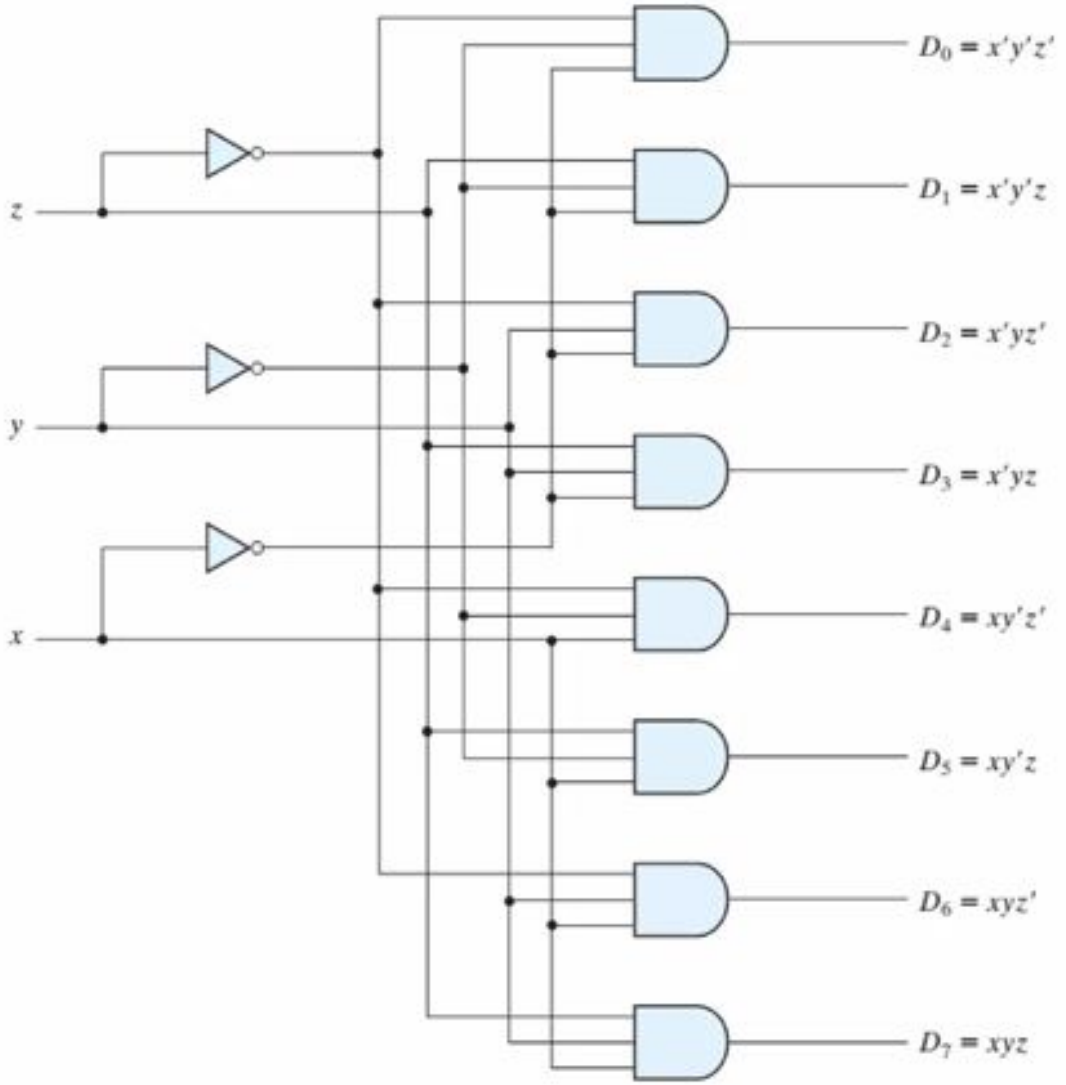
- A decoder is a combinational circuit that converts binary information from n inputs lines to a maximum of 2^n unique output lines.
- The decoders presented here are called n-to-m line decoders, where $m \leq 2^n$.

Truth table of a three-to-eight line decoder

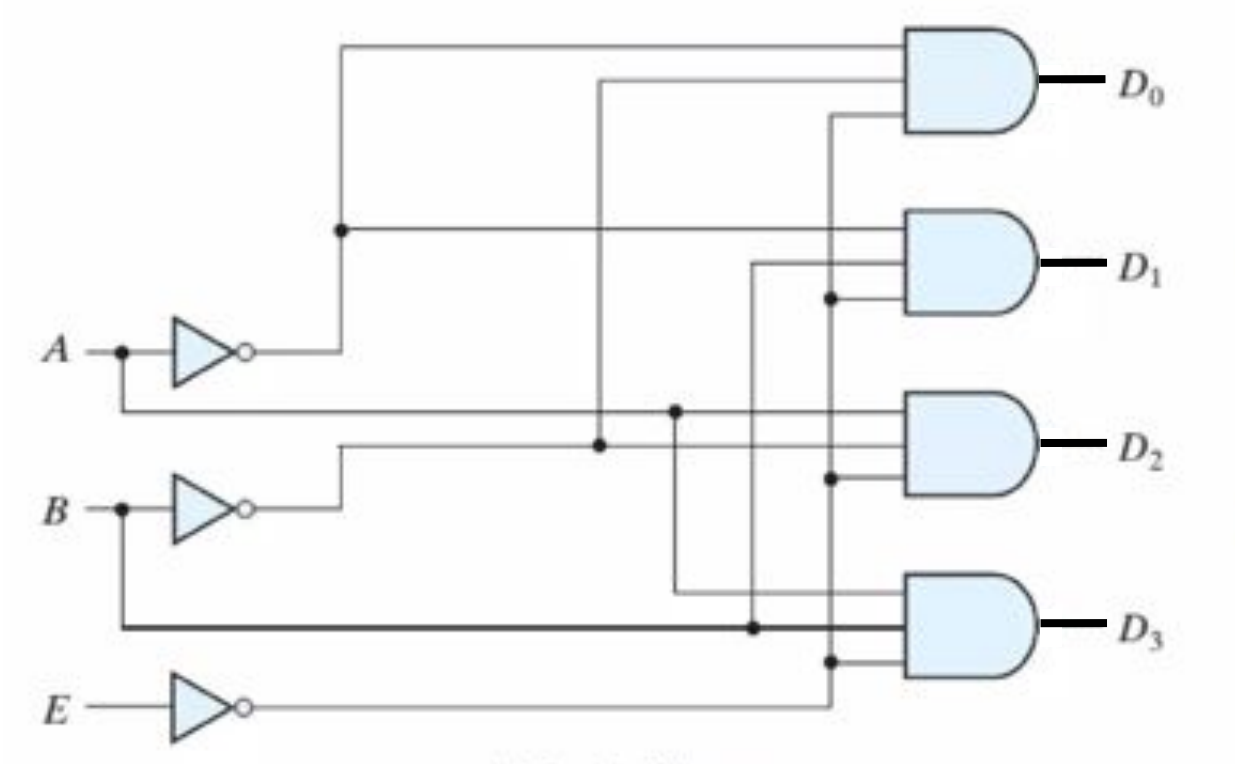


Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Three-to-Eight line decoder



Two-to-four line decoder with enable input



E	A	B	D_0	D_1	D_2	D_3
1	X	X	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or fewer) input lines and n output lines.
- The encoder defined in below has the limitation that only one input can be active at any given time.

Octal-to-binary encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

Priority encoder

- A priority encoder is a type of encoder circuit that processes multiple inputs but assigns priority to the highest-order active input.
- It outputs the binary code of the highest-priority active input.
- If multiple inputs are active simultaneously, the encoder prioritizes the input with the highest priority (typically, the input with the highest numerical value).

Truth table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Inputs (D0, D1, D2, D3):

- D3 has the highest priority, followed by D2, D1, and D0.

Outputs (x, y):

- Binary representation of the highest active input.
- E.g., if D3 = 1, then output is x = 1, y = 1 (since D3 has the highest priority).

Valid Output (V):

- V = 0 if no inputs are active.
- V = 1 if at least one input is active.

Priority encoder

Priority Rule:

- If multiple inputs are high, the encoder selects the input with the highest index (e.g., D3 over D2).

Example:

- Input: D3 = 1, D1 = 1
- Output: Only D3 is encoded, resulting in output x = 1, y = 1.

Valid Output Signal (V):

- When all inputs are 0, V remains 0, indicating no valid inputs.
- If any input is active, V becomes 1.

Thank You!