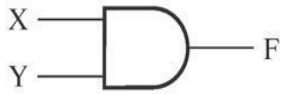
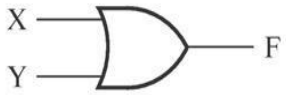
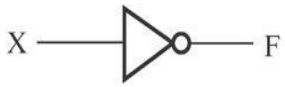
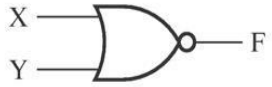
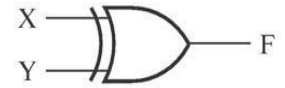
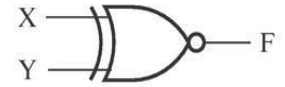
HDL 1 - Combinational Circuit

Nahin Ul Sadad/Farjana Parvin
Lecturer
CSE, RUET

Digital Electronics

Digital Electronics:

Digital Electronics deal with binary numbers. Logic gates are building blocks of digital electronics. There are two types of digital electronics circuit. They are: Combinational Circuit and Sequential Circuit

Name	Distinctive-Shape Graphics Symbol	Algebraic Equation	Truth Table
AND		$F = XY$	X Y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = X + Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT (inverter)		$F = \overline{X}$	X | F 0 | 1 1 | 0
NAND		$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{X + Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = X\overline{Y} + \overline{X}Y$ $= X \oplus Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR (XNOR)		$F = XY + \overline{X}\overline{Y}$ $= X \oplus Y$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Digital Electronics (Combinational Circuit)

Combinational Circuit:

Output of combinational circuit only depends on present inputs. Combinational Circuit is instantaneous. It will give output instantly when it is given inputs. Applications of combinational circuit: Arithmetic and Logic Unit (ALU) Implementation etc.

Example: Half Adder

Inputs		Outputs	
A	B	Carry, C	Sum, S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

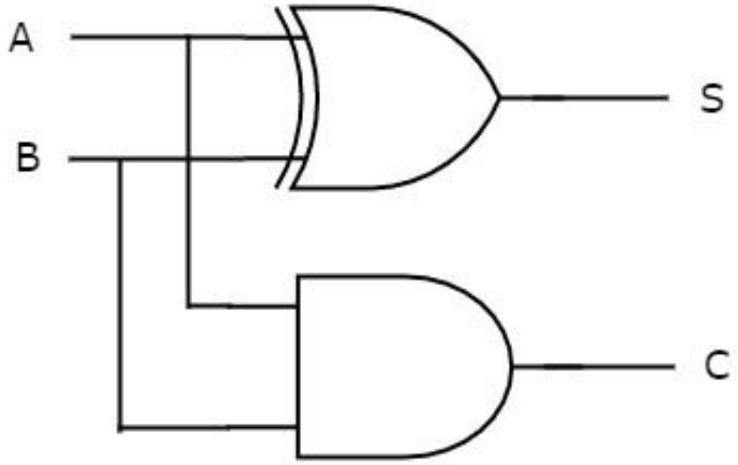


Fig: Half Adder

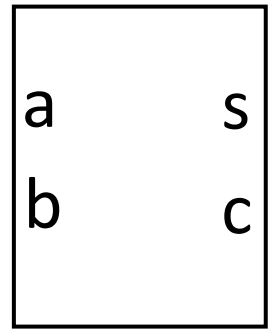


Fig: Half adder chip

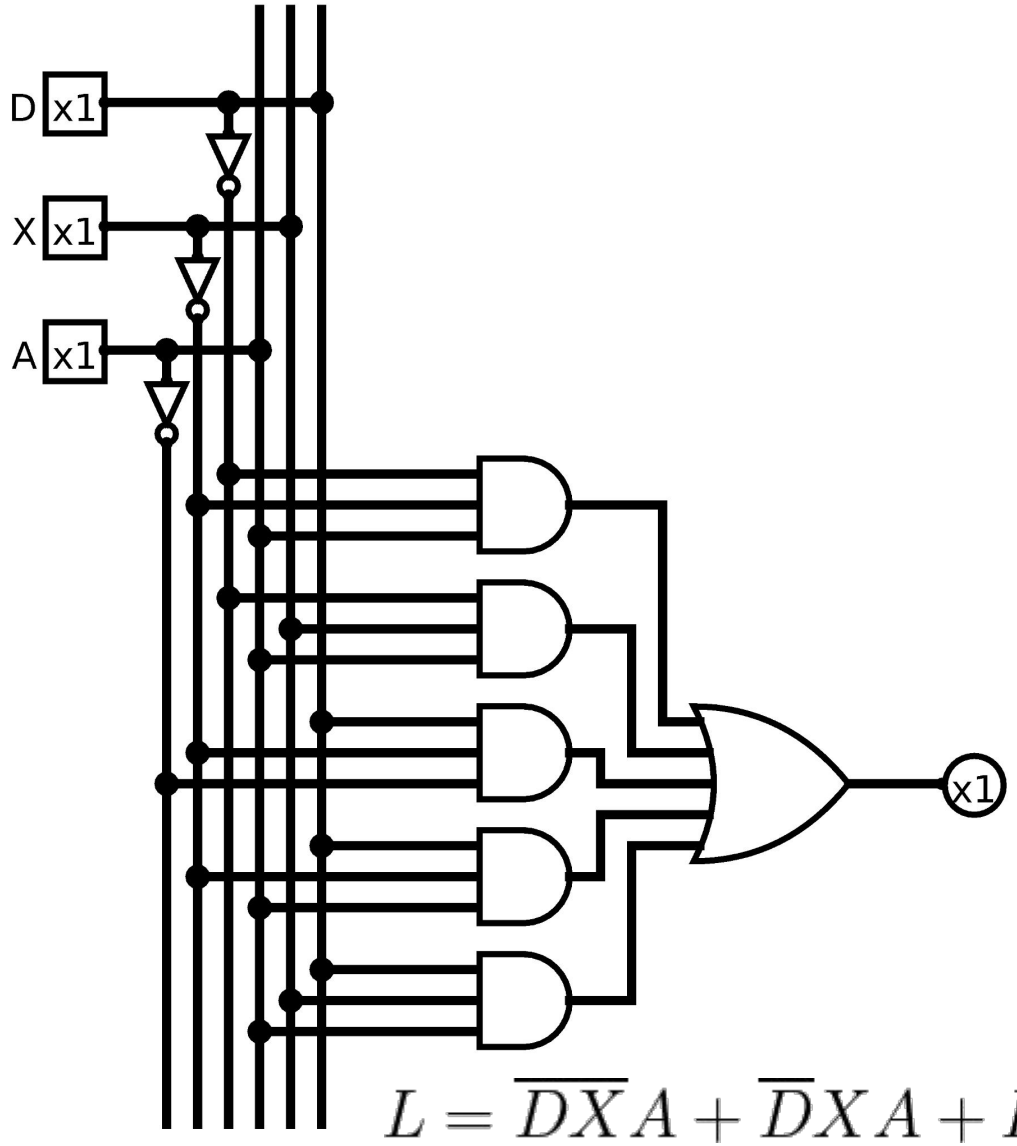
Combinational Circuit

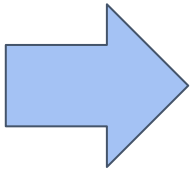
Truth Table

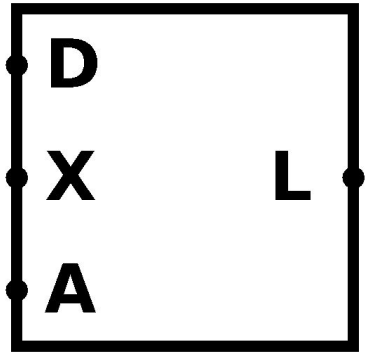
D	X	A	L
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

$$L = \overline{DX}A + \overline{D}XA + D\overline{XA} + D\overline{X}A + DXA$$

Combinational Circuit







Module of L

$$L = \overline{DX}A + \overline{D}X\,A + D\overline{XA} + D\overline{X}A + DXA$$

Combinational Circuit of L

Combinational Circuit

Verilog code of previous circuit is shown below:

comb1.v

```
module comb1
(
    input D,
    input X,
    input A,
    output L
);

   assign L = (~D)&(~X)&A | (~D)&X&A | D&(~X)&(~A) | D&(~X)&A | D&X&A;

endmodule
```

Combinational Circuit

Verilog code of test bench of previous circuit is shown below:

comb1_tb.v

```
`timescale 1ns/1ps

module comb1_tb;
    reg D;
    reg X;
    reg A;
    wire L;

    comb1 uut
    (
        .D(D),
        .X(X),
        .A(A),
        .L(L)
    );

    initial begin
        D = 0;
        X = 0;
        A = 0;

        #20;
        A = 1;

        #20;
        X = 1;

        #20;
        D = 1;

        #20;
        X = 0;

        #20;
        A = 0;

        #20;
    end

 initial begin
  $monitor("D=%d X=%d A=%d L=%d\n",
                              D, X, A, L);
 end

endmodule
```

Combinational Circuit

Table 3.1 Verilog operators

Type of operation	Operator symbol	Description	Number of operands
Arithmetic	+	addition	2
	-	subtraction	2
	*	multiplication	2
	/	division	2
	%	modulus	2
	**	exponentiation	2
Shift	>>	logical right shift	2
	<<	logical left shift	2
	>>>	arithmetic right shift	2
	<<<	logical left shift	2
Relational	>	greater than	2
	<	less than	2
	>=	greater than or equal to	2
	<=	less than or equal to	2
Equality	==	equality	2
	!=	inequality	2
	===	case equality	2
	!==	case inequality	2
Bitwise	~	bitwise negation	1
	&	bitwise and	2
	|	bitwise or	2
	^	bitwise xor	2
Reduction	&	reduction and	1
	|	reduction or	1
	^	reduction xor	1
Logical	!	logical negation	1
	&&	logical and	2
	||	logical or	2
Concatenation	{ }	concatenation	any
	{ { } }	replication	any
Conditional	? :	conditional	3

Figure: Verilog Operators

To know more about Verilog Operators, follow this link:

https://www.chipverify.com/verilog/verilog-operators

Combinational Circuit

Table 3.2 Operator precedence

Operator	Precedence
! ~ + - (unary)	highest
**	
* / %	
+ - (binary)	
>> << >>> <<<	
< <= > >=	
== != === !==	
&	
^	
|	
&&	
||	
?:	lowest

Figure: Verilog Operators

Combinational Circuit

Verilog code of previous circuit is shown below:

comb2.v

```
module comb2
(
    input [3:0] A,
    input [3:0] B,
    input [3:0] C,
    input [3:0] D,
    output [3:0] Arithmetic,
    output [3:0] Shift,
    output [3:0] Relational,
    output [3:0] Equality,
    output [3:0] Bitwise,
    output [3:0] Reduction,
    output [3:0] Logical,
    output [3:0] Concatenation,
    output [3:0] Conditional
);
```

```
    assign Arithmetic = B+C;
    assign Shift = B>>C;
    assign Relational = A > B;
    assign Equality = A == D;
    assign Bitwise = B & C;
    assign Reduction = |B;
    assign Logical = (A > B) || (A > D);
    assign Concatenation = {C[1:0], D[3:2]};
    assign Conditional = (A>B) ? A : B;

endmodule
```

Combinational Circuit

Verilog code of test bench of previous circuit is shown below:

comb2_tb.v

```
`timescale 1ns/1ps

module comb2_tb;
    reg [3:0] A;
    reg [3:0] B;
    reg [3:0] C;
    reg [3:0] D;
    wire [3:0] Arithmetic;
    wire [3:0] Shift;
    wire [3:0] Relational;
    wire [3:0] Equality;
    wire [3:0] Bitwise;
    wire [3:0] Reduction;
    wire [3:0] Logical;
    wire [3:0] Concatenation;
    wire [3:0] Conditional;
```

```
comb2 uut
(
  .A(A),
  .B(B),
  .C(C),
  .D(D),
  .Arithmetic(Arithmetic),
  .Shift(Shift),
  .Relational(Relational),
  .Equality(Equality),
  .Bitwise(Bitwise),
  .Reduction(Reduction),
  .Logical(Logical),
  .Concatenation(Concatenation),
  .Conditional(Conditional)
);
```

Combinational Circuit

```
initial begin
  A = 4'b1100;
  B = 4'b0110;
  C = 4'b0010;
  D = 4'b1100;
  #20;
end

initial begin
$monitor("A=%4b, B=%4b, C=%4b, D=%4b\n", A, B, C, D,
         "Arithmetic=%4b, Shift=%4b, Relational=%4b\n", Arithmetic, Shift, Relational,
         "Equality=%4b, Bitwise=%4b, Reduction=%4b\n",  Equality, Bitwise, Reduction,
         "Logical=%4b, Concatenation=%4b, Conditional=%4b\n", Logical, Concatenation, Conditional);
end



endmodule
```

Simulation Result

```
A=1100, B=0110, C=0010, D=1100
Arithmetic=1000, Shift=0001, Relational=0001
Equality=0001, Bitwise=0010, Reduction=0001
Logical=0001, Concatenation=1011, Conditional=1100
```

Combinational Circuit

Decoder

Decoder is a combinational circuit that has ‘n’ input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code.

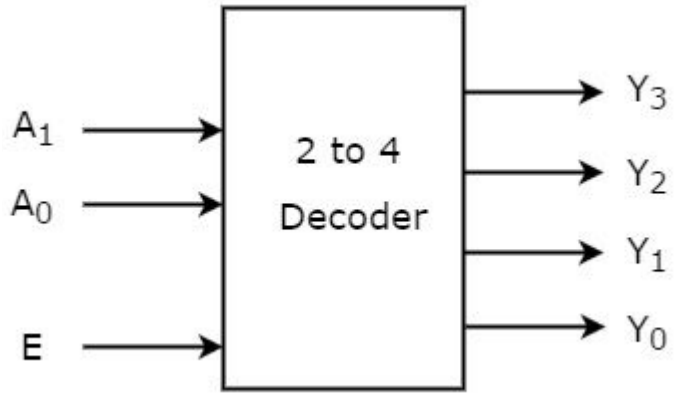


Table: Truth Table of 2 to 4 Decoder

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$$Y_3 = E.A_1.A_0$$

$$Y_2 = E.A_1.A_0'$$

$$Y_1 = E.A_1'.A_0$$

$$Y_0 = E.A_1'.A_0'$$

Combinational Circuit
Decoder

Table: Truth Table of 2 to 4 Decoder

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

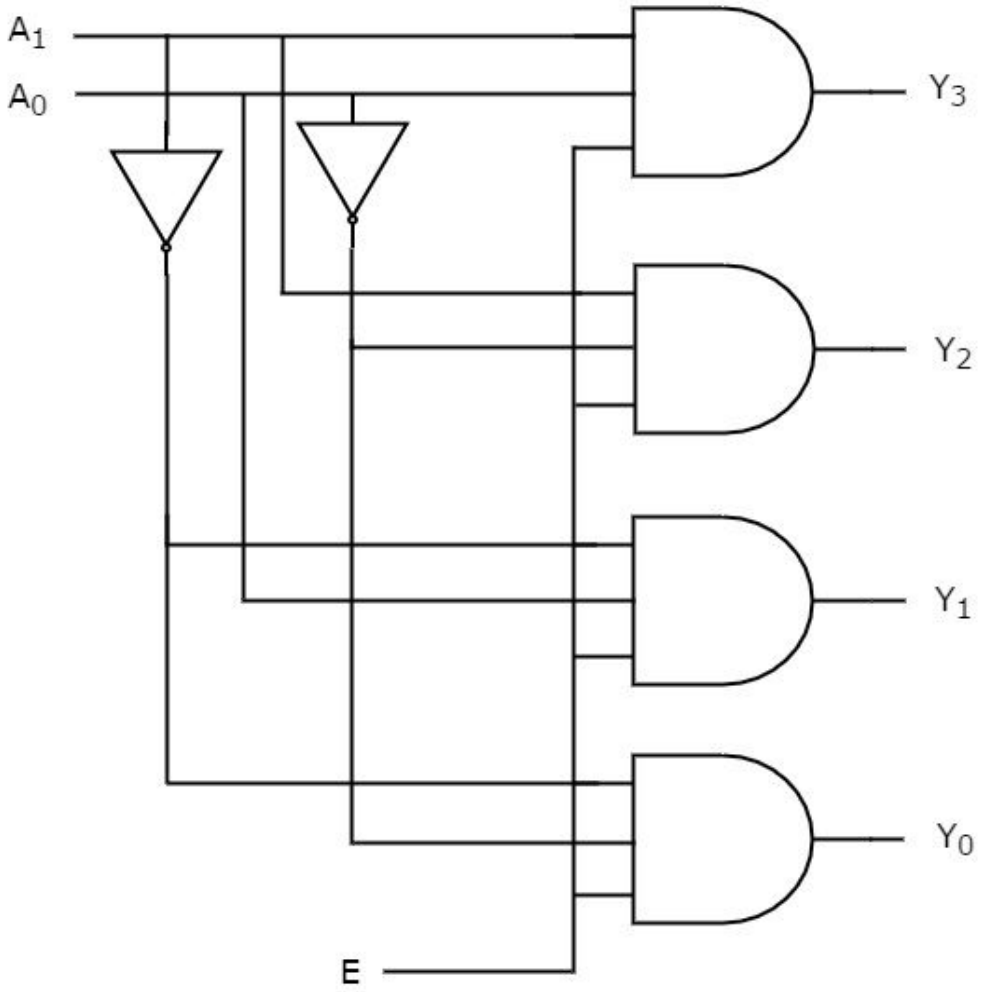


Figure: 2 to 4 Decoder Circuit

$$Y_3 = E.A_1.A_0$$

$$Y_2 = E.A_1.A_0'$$

$$Y_1 = E.A_1'.A_0$$

$$Y_0 = E.A_1'.A_0'$$

Combinational Circuit

Decoder in HDL (using `assign` statement)

decoder2to4.v

```
module decoder2to4
(
    input [1:0] A,
    input E,
    output [3:0] Y
);
    assign Y[3] = E & A[1] & A[0];
    assign Y[2] = E & A[1] & ~A[0];
    assign Y[1] = E & ~A[1] & A[0];
    assign Y[0] = E & ~A[1] & ~A[0];
endmodule
```

Combinational Circuit

Decoder Testbench in HDL

decoder2to4_tb.v

```
`timescale 1ns/1ps

module decoder2to4_tb;
    reg [1:0] A;
    reg E;
    wire [3:0] Y;

    decoder2to4 uut
    (
        .A(A),
        .E(E),
        .Y(Y)
    );

    initial begin
        A[1] = 0;
        A[0] = 0;
        E = 0;
        #20;
        E = 1;
        #20;
        A[0] = 1;
        #20;
        A[1] = 1;
        #20;
        A[0] = 0;
        #20;
    end

 initial begin
    $monitor("E=%d A[1]=%d A[0]=%d Y[3]=%d Y[2]=%d Y[1]=%d
Y[0]=%d\n",
                                    E, A[1], A[0], Y[3], Y[2], Y[1],
Y[0]);
 end

endmodule
```

Combinational Circuit

ALWAYS BLOCK for a Combinational Circuit

To facilitate system modeling, Verilog contains a number of procedural statements, which are executed in sequence. Since their behavior is different from the normal concurrent circuit model, these statements are encapsulated inside an **`always block`** or **`initial block`**. The **`initial block`** is executed once when the simulation is started. Syntax for **`always block`** is shown below:

```
always @([sensitivity-list])
   begin [optional name]
      [optional local variable declaration];
      [procedural statement];
      [procedural statement];
   end
```

The `[sensitivity-list]` term is a list of signals and events to which the **`always block`** responds (i.e., is "sensitive to"). For a combinational circuit, all the input signals should be included in this list. We use asterisk (*) to include all the input signals in this list.

We can use `always block` instead of `assign` statement because `always block` allow us to use `if` statement and `case` statement to implement combinational circuit.

Combinational Circuit

ALWAYS BLOCK for a Combinational Circuit

A procedural assignment can only be used within an always block or initial block. There are two types of assignments: blocking assignment and nonblocking assignment. Their basic syntax is:

```
[variable-name] = [expression]; // blocking assignment
[variable-name] <= [expression]; // nonblocking assignment
```

In a blocking assignment, the expression is evaluated and then assigned to the variable immediately, before execution of the next statement (the assignment thus "blocks" the execution of other statements). It behaves like the normal variable assignment in the C language.

In a nonblocking assignment, the evaluated expression is assigned at the end of the always block (the assignment thus does not block the execution of other statements).

The basic rule of thumb is:

Use blocking assignments for a combinational circuit.
Use nonblocking assignments for a sequential circuit.

Combinational Circuit

Decoder in HDL (Using `always block` and `if` statement)

decoder2to4T2.v

```
module decoder2to4T2
(
    input [1:0] A,
    input E,
    reg [3:0] Y
);
```

```
always @*
begin
    if(E == 1'b0)
        Y = 4'b0000;
    else if (A == 2'b00)
        Y = 4'b0001;
    else if (A == 2'b01)
        Y = 4'b0010;
    else if (A == 2'b10)
        Y = 4'b0100;
    else
        Y = 4'b1000;
end
endmodule
```

Combinational Circuit

Decoder Testbench in HDL

decoder2to4T2_tb.v

```
`timescale 1ns/1ps

module decoder2to4_tb;
    reg [1:0] A;
    reg E;
    wire [3:0] Y;

    decoder2to4T2 uut
    (
        .A(A),
        .E(E),
        .Y(Y)
    );

    initial begin
        A[1] = 0;
        A[0] = 0;
        E = 0;
        #20;
        E = 1;
        #20;
        A[0] = 1;
        #20;
        A[1] = 1;
        #20;
        A[0] = 0;
        #20;
    end

 initial begin
    $monitor("E=%d A[1]=%d A[0]=%d Y[3]=%d Y[2]=%d Y[1]=%d
Y[0]=%d\n",
                                    E, A[1], A[0], Y[3], Y[2], Y[1],
Y[0]);
 end

endmodule
```

Combinational Circuit

Decoder in HDL (Using `always block` and `case` statement)

decoder2to4T3.v

```
module decoder2to4T2
(
    input [1:0] A,
    input E,
    reg [3:0] Y
);
```

```
always @*
begin
    case({E,A})
        3'b100: Y = 4'b0001;
        3'b101: Y = 4'b0010;
        3'b110: Y = 4'b0100;
        3'b111: Y = 4'b1000;
        default: Y = 4'b0000;
    endcase
end

endmodule
```

Combinational Circuit

Encoder

An Encoder is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and ‘n’ output lines. It will produce a binary code equivalent to the input, which is active High.

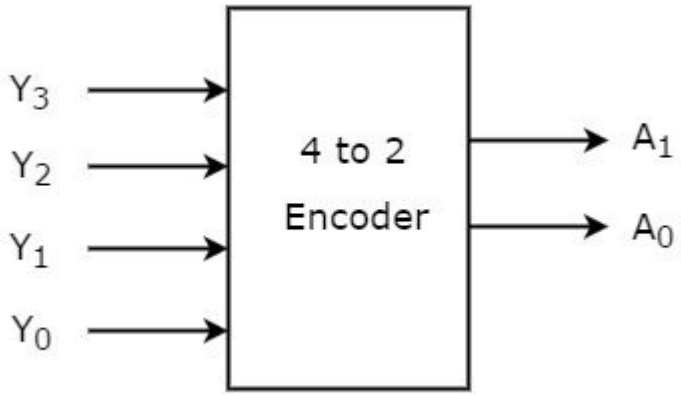


Table: Truth Table of 4 to 2 Encoder

Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

Combinational Circuit
Encoder

Table: Truth Table of 4 to 2 Encoder

Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

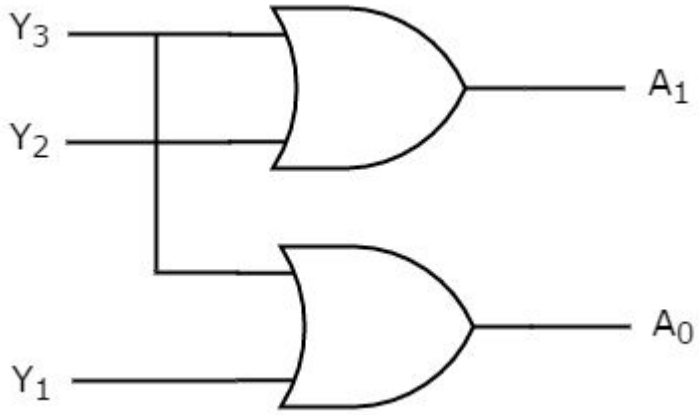


Figure: 4 to 2 Encoder

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

Combinational Circuit

Priority Encoder

A 4 to 2 priority encoder has four inputs Y3, Y2, Y1 & Y0 and two outputs A1 & A0. Here, the input, Y3 has the highest priority, whereas the input, Y0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the binary code corresponding to the input, which is having higher priority.

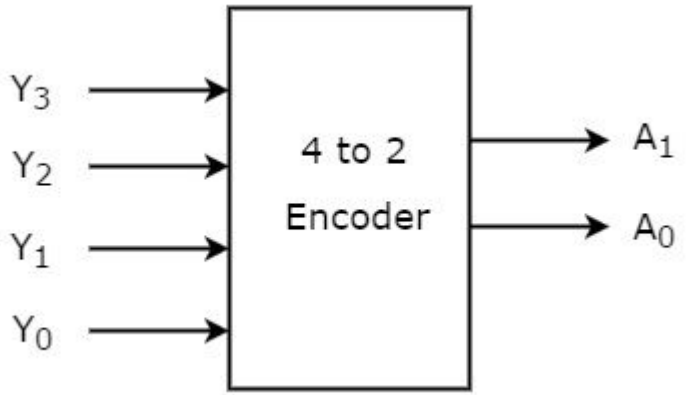


Table: Truth Table of 4 to 2 Priority Encoder

Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

$$A_1 = \bar{Y_3} Y_2 + Y_3$$
$$A_0 = \bar{Y_3} \bar{Y_2} Y_1 + Y_3$$

Combinational Circuit

Multiplexer

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, ‘n’ selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

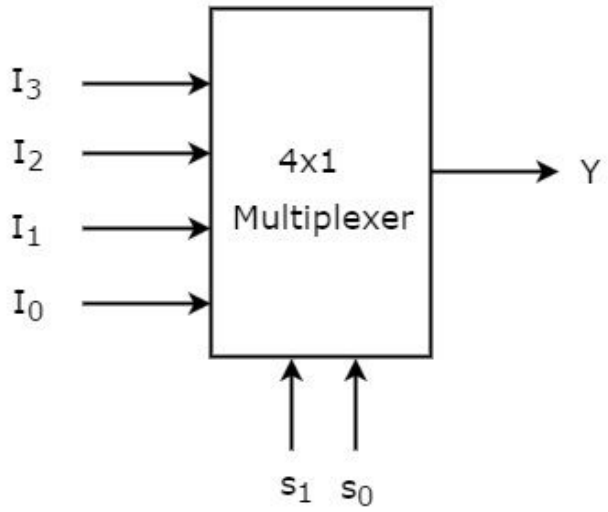


Table: Truth Table of 4x1 Multiplexer

Selection Lines		Output
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

Combinational Circuit
Multiplexer

Table: Truth Table of 4x1 Multiplexer

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

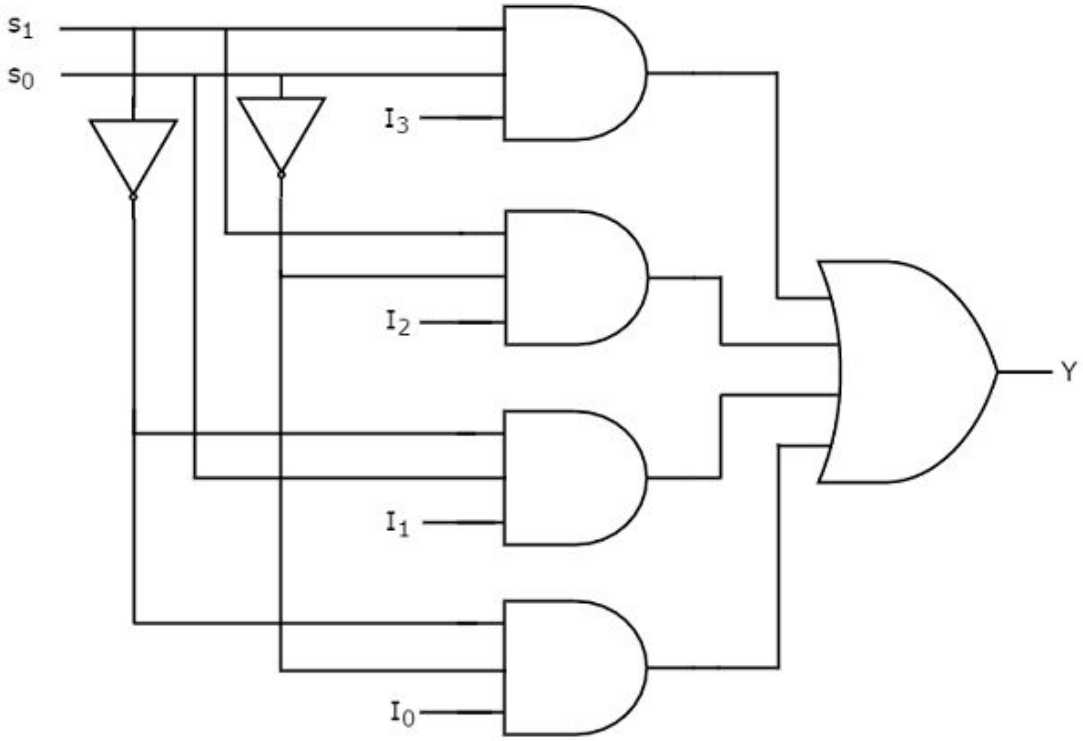


Figure: 4x1 Multiplexer

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

Combinational Circuit

Demultiplexer

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, ‘n’ selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

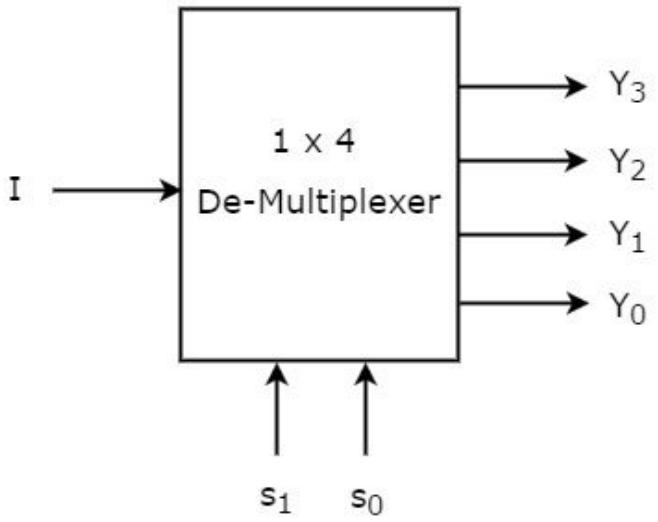


Table: Truth Table of 1x4 Demultiplexer

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

$$Y_3 = s_1 s_0 I$$

$$Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I$$

Combinational Circuit
Demultiplexer

Table: Truth Table of 1x4 Demultiplexer

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

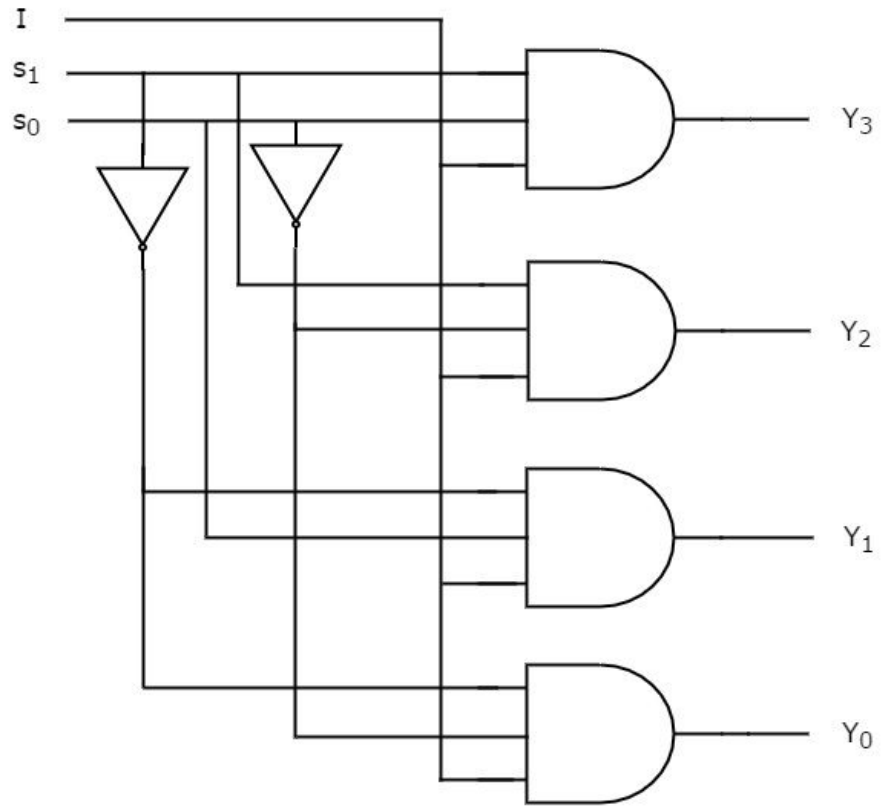


Figure: 1x4 Demultiplexer

$$Y_3 = s_1 s_0 I$$

$$Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I$$

Example: Verilog

Question: Implement a **2 to 4 Decoder** in Verilog HDL along with a test bench.

Answer:

Boolean Table is shown below:

Table: Truth Table of 2 to 4 Decoder

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$$Y_3 = E.A_1.A_0$$

$$Y_2 = E.A_1.A_0'$$

$$Y_1 = E.A_1'.A_0$$

$$Y_0 = E.A_1'.A_0'$$

Example: Verilog

Verilog code of **2 to 4 Decoder** is shown below:

decoder2to4.v

```
module decoder2to4T2
(
    input [1:0] A,
    input E,
    reg [3:0] Y
);
always @*
begin
    if(E == 1'b0)
        Y = 4'b0000;
    else if (A == 2'b00)
        Y = 4'b0001;
    else if (A == 2'b01)
        Y = 4'b0010;
    else if (A == 2'b10)
        Y = 4'b0100;
    else
        Y = 4'b1000;
end
endmodule
```

Example: Verilog

Test bench of **2 to 4 Decoder** is shown below:

decoder2to4_tb.v

```
`timescale 1ns/1ps

module decoder2to4_tb;
    reg [1:0] A;
    reg E;
    wire [3:0] Y;

    decoder2to4T2 uut
    (
        .A(A),
        .E(E),
        .Y(Y)
    );

    initial begin
        A[1] = 0;
        A[0] = 0;
        E = 0;
```

Example: Verilog

```
        #20;
        E = 1;
        #20;
        A[0] = 1;
        #20;
        A[1] = 1;
        #20;
        A[0] = 0;
        #20;
    end

  initial begin
     $monitor("E=%d A[1]=%d A[0]=%d Y[3]=%d Y[2]=%d Y[1]=%d Y[0]=%d\n",
                                  E, A[1], A[0], Y[3], Y[2], Y[1], Y[0]);
  end

endmodule
```

Exercises

1. Implement following:
   a. 3 to 8 Decoder
   b. 4 to 2 Priority Encoder
   c. 4 to 1 Multiplexer
   d. 1 to 4 Demultiplexer

   in Verilog HDL along with a test bench.
   a. Using **`assign`** statement
   b. Using **`if else`** statement
   c. Using **`case`** statement

Thank You ☺️