Digital Logic Design

Combinational Design

Outline

- Combinational Circuits
- Analysis Procedure
- Design Procedure
- Adders and multipliers
- Comparators
- Decoders and Encoders
- Multiplexers
- HDL For Combinational Circuits

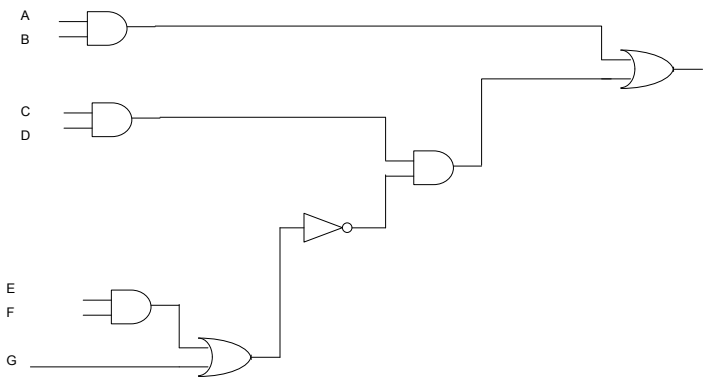
Combinational Circuits

- What is a combinational circuit?
- What is the difference between combinational and sequential circuits
- Implementation MSI and standard cells in ASIC

Analysis Procedure

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Find the Boolean function of these gates
2. Label all the gates that are functions of input variables and previously labeled gates with arbitrary symbols. Find the Boolean function of these gates.
3. Repeat step 3 until the outputs of all the gates are labeled
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables (or truth table)

Analysis Procedure



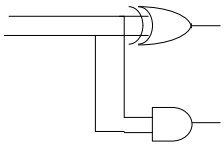
Design Procedure

1. From the specification of the circuit. Determine the required number of input and output and assign a symbol to each.
2. Derive the truth that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean expressions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design.

- Example: BCD to Excess-3 Code Converter.

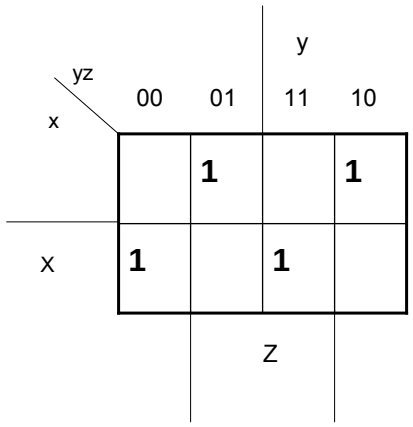
Binary Adder-Subtractor

- Half adder: S=x'y+xy', C=xy

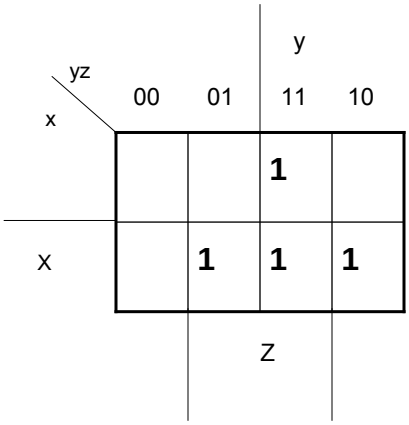


Full Adder

x	y	z	s	c
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

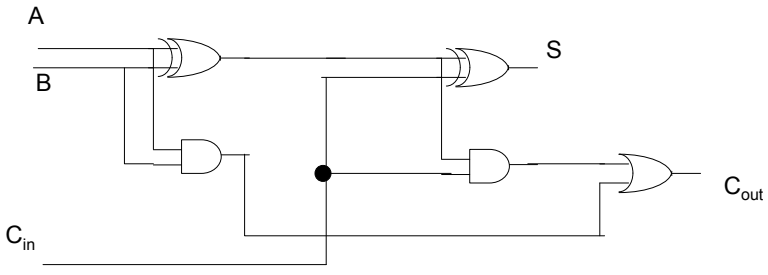


C

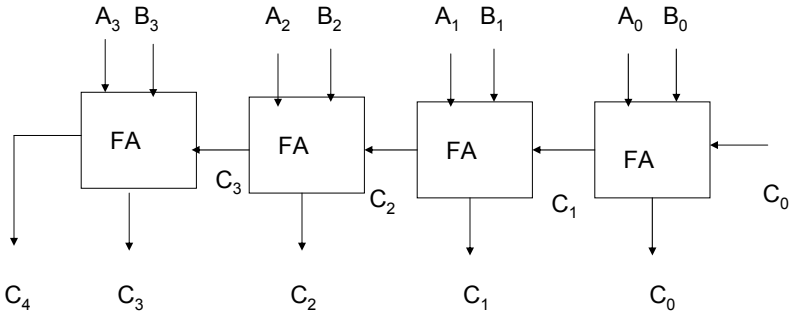


S=XZ+XY

Full Adder



Binary Adder



Carry Propagation

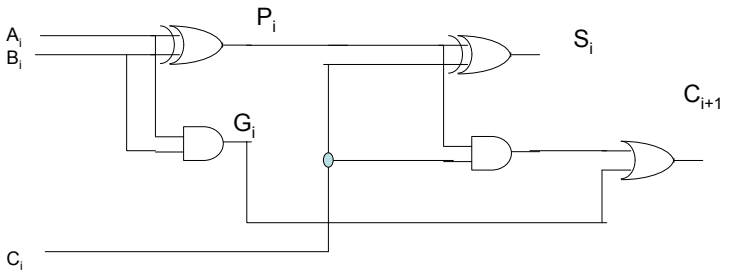
- The value of s_i depends on the current A_i and Bi and C_i. C_i depends on A_{i-1}, and B_{i-1}, and so on.
- That means the carry propagates across all the digits in the two numbers to be added.
- Carry propagation time is a limiting factor on the speed of addition (basic operation in virtually everything).
- S_i's will not be ready at the same time
- We need to speed-up addition

Carry Propagation

$Pi=A_i \oplus B_i$ $G_i=A_iB_i$ G is called the carry Generator

Gi=1 if A_i and B_i=1

$S_i=P_i \oplus C_i$ $C_{i+1}=G_i+P_iC_i$ Carry if either G_i or on of A,B and C



Carry Lookahead

- G_i is called the carry Generator, and P_i is the carry propagate.
- We can calculate the carry at every stage by recursively substituting C_i

  C_0=input carry

  $C_1=G_0+P_0C_0$

  $C_2=G_1+P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$

  $C_3=G_2+P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

  Circuit in Figure 4-11

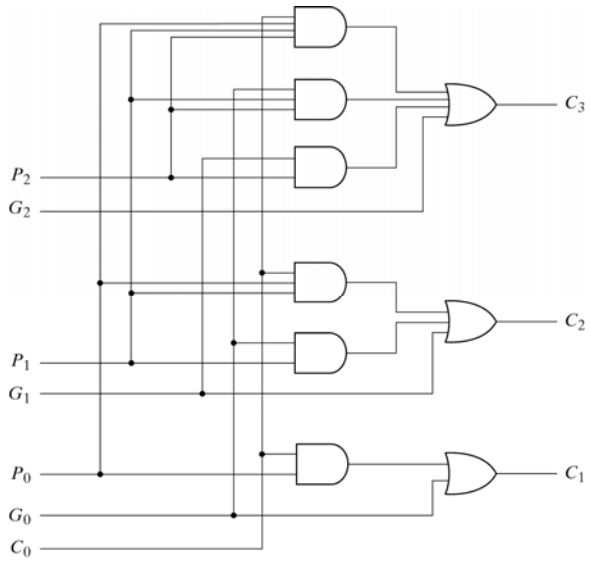


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

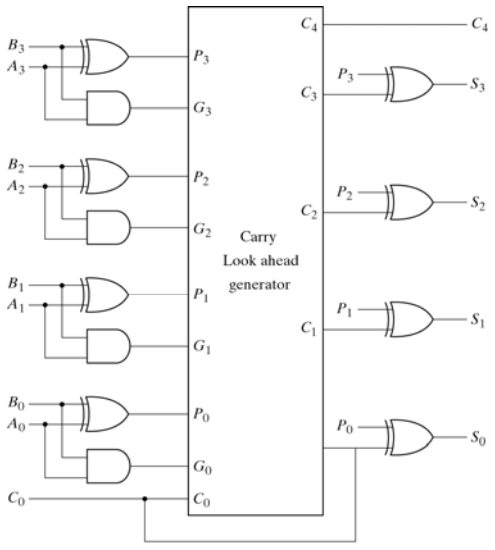
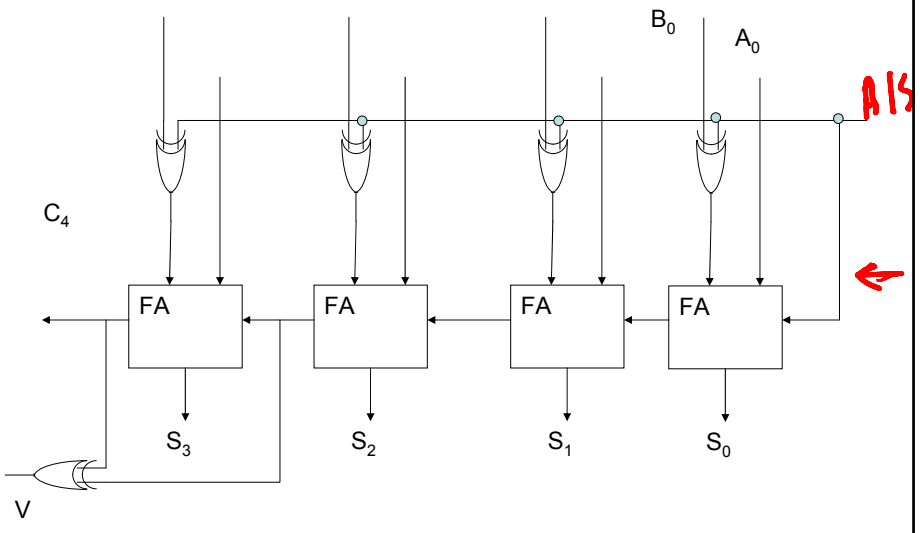


Fig. 4-12 4-Bit Adder with Carry Lookahead

Adder/Subtractor



Overflow

- When adding two n-bits number, the answer has a maximum of n+1 bits.
- If the numbers are represented in the computer by n bits, the n+1st bit is an overflow.
- Usually the overflow is detected and reported to the user.

Overflow

```
+60   0  0111100
+90   0  1011010
-------------------
      1  0010110
```

```
-60   1  1000100
-90   1  0100110
-------------------
1     0  1101010
```

An overflow is detected if the carry into the sign bit and the carry out of the sign bit is not the same

```
-60   1   1000100
+90   0   1011010
-------------------
1     0   0011110
```

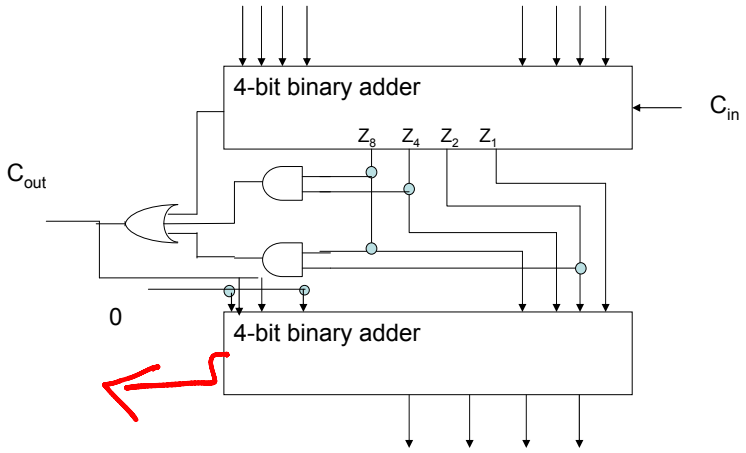

Decimal Adder

Binary Sum					BCD Sum					Decimal
K	Z8	Z4	Z2	Z1	K	Z8	Z4	Z2	Z1	
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

Decimal Adder

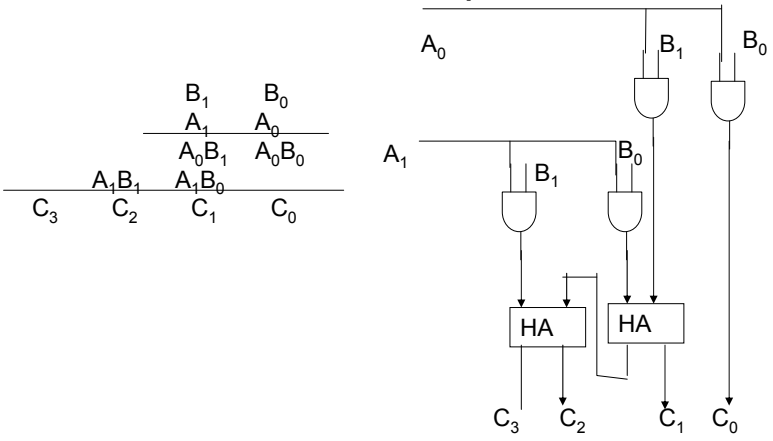
- The output of the BCD could be anything between 0 and 19 (9+9+carry).
- From the truth table, it is clear that there is a carry (BCD carry) if any one of the following occurs
  1. K =1 1 (if the number is greater than 16).
  2. Z_8=1 and Z_4=1
  3. If Z_8=1 and Z_2=1
- If there is a carry, we must add 6 to the binary number to get the BCD code.
- That leads to the following circuit

Decimal Adder



Binary Multiplier

- First, consider 2 bit multiplier



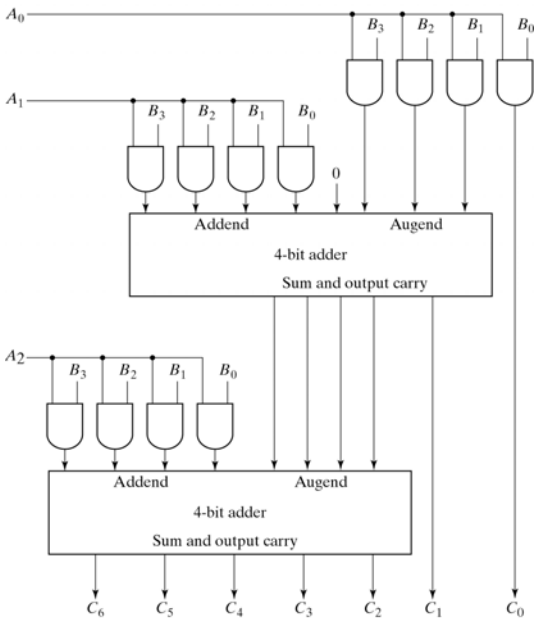


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

Magnitude Comparator

- Assume that we want to design a magnitude comparator for 2 4-bit numbers.
- Direct implementation of this requires a truth table with 2^8=256 entries.
- It is easier to understand the algorithm by which we compare two numbers, that leads to a much less complicated design process.
- Assume the 2 numbers are represented as $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$
- The algorithm works as follows

Magnitude Comparator

- The two numbers are equal iff all the bits in the 2 numbers are equal. That leads to a design of 4EX-OR followed by inverter (actually ex-nor) and an or gate.
- For A to be greater than B, we must have $A_j > B_j$ and $A_i=B_i$ $j>i$.
- So, we start at digit 3, compare A_3 and B_3, either one is greater or they are equal we move to A_2 and B_2 and so on.
- If we define x_i to be the ex-nor of A_i and B_i i.e. x_i is 1 if $A_i=B_i$

Magnitude Comparator

- In this case,
- $(A=B)=x_3x_2x_1x_0$
- $(A>B)=A_3B'_3+x_3A_2B'_2 + x_3x_2A_1B'_1+x_3x_2x_1A_0B'_0$
- $(A<B)$ replace the prime from B to A.
- Circuit is shown in Figure 4-17

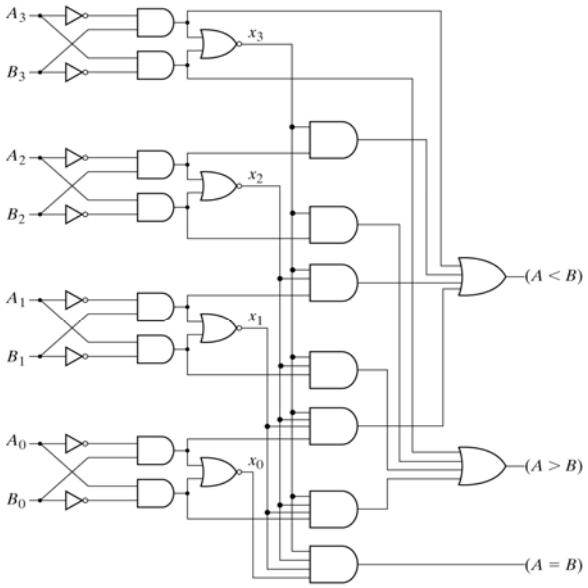


Fig. 4-17 4-Bit Magnitude Comparator

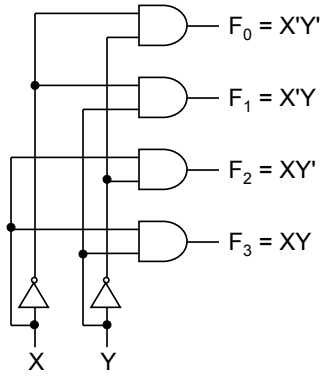
Decoders

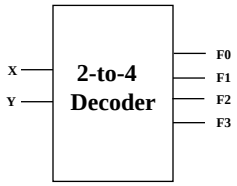
- A *decoder* is a combinational circuit that converts binary information from *n* inputs to a maximum of 2^n outputs.
- A decode is called *n*-to-*m* decoder, where $m \leq 2^n$
- Consider 3-8 decoder, truth table with 3 inputs *x,y,z* and 8 outputs D_7 .. D_0 the circuit is shown in Figure 4-18

Decoders

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

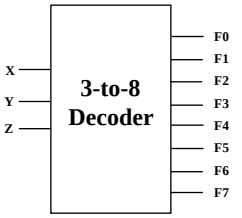
- From truth table, circuit for 2x4 decoder is:
- Note: Each output is a 2-variable minterm (**X'Y', X'Y, XY'** or **XY**)

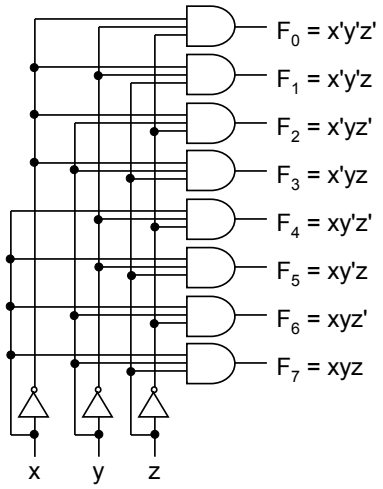




Decoders

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_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





Decoders

- Some decoders are implemented using NAND gates, in this case it will be more economical to produce the output in their complemented form.
- 2-4 decoder
- Circuit 4-19
- When E is 1, non

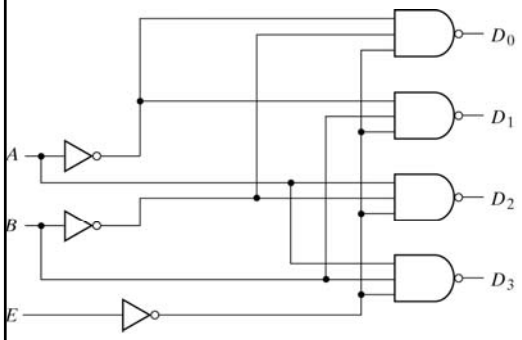
Of the outputs I 0

- Decoder may be

Activated With E=0 or 1

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

Decoders



(a) Logic diagram

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

(b) Truth table

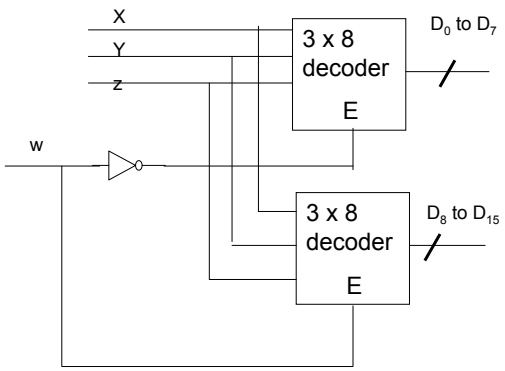
Fig. 4-19 2-to-4-Line Decoder with Enable Input

Decoders

- A *decoder* with an *Enable* input can function as a de*multiplexer*
- A demultiplexer is a circuits that receives data from a single line, and direct it to a possible of 2^n lines (example sharing a communication line).
- The decoder in the previous slide can function as a demultiplexer if we consider E to be the data and A, and B to be the input selection.
- Verify this by assuming selection 10 and determine the output (always equal to E).

Decoders

- Decoders with enable can be connected together to form a larger decoder

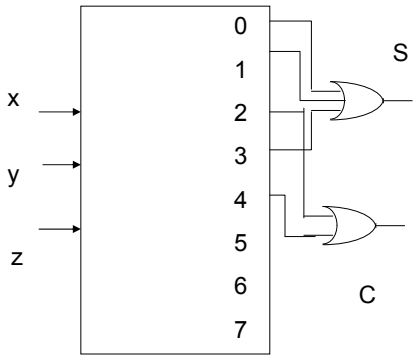


Decoders

- Any n-variable logic function can be implemented using a single n-to-2^n decoder to generate the minterms
  - OR gate forms the sum.
  - The output lines of the decoder corresponding to the minterms of the function are used as inputs to the or gate.
- Any combinational circuit with *n* inputs and *m* outputs can be implemented with an n-to-2^n decoder with *m* OR gates.
- Suitable when a circuit has many outputs, and each output function is expressed with few minterms.

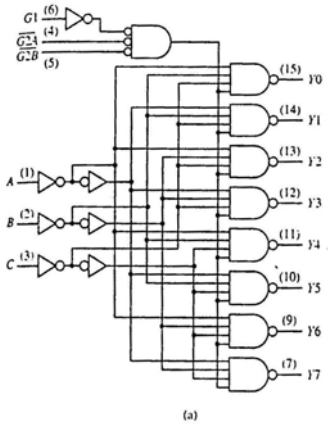
Decoders

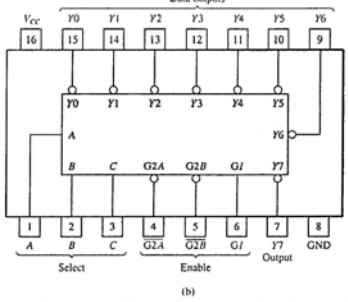
- Decoders can be used to implement logic functions.
- Consider the function S(x,y,z)=Σ(0,1,3) and C(x,y,z)= Σ(4,2)
- Verify this by stating what will be the output if the input is any of the different combination in the truth table



Standard MSI Binary Decoders Example

74138 (3-to-8





Inputs					Outputs							
Enable		Select										
G1	$\overline{G2}$*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L
×	H	×	×	×	H	H	H	H	H	H	H	H
L	×	×	×	×	H	H	H	H	H	H	H	H

$\overline{G2}$* = $\overline{G2A}$ + $\overline{G2B}$

(c)

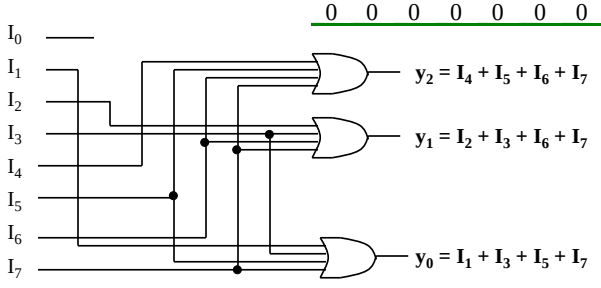
(a) Logic circuit.
(b) Package pin configuration.
(c) Function table.

Encoders

- Encoders perform the reverse operation of a decoder.
- An encoder has up to 2^n input lines, and n output lines.
- The encoder generates the binary code corresponding to the active input line.
- Truth table with 8 variables and three outputs, only need 1 in every row
- $x=D_1+D_3+D_5+D_7$; $y = D_2+D_3+D_6+D_7$; $Z=D_4+D_5+D_5+D_7$

Encoders

Inputs								Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0
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



Priority Encoders

- Priority encoders are encoders with a certain priority scheme.
- If more than one input is active, the one with the higher priority is encoded.
- The following figure shows the truth table for a priority encoder.
- Note than there is a valid bit. The valid bit indicates if the output is valid or not, if non of the input is active, the V bit is 0, means nothing is active

Priority Encoders

- **What if more than one input line has a value of 1?**
- **Ignore "lower priority" inputs.**
- **Idle indicates that no input is a 1.**
- **Note that polarity of Idle is opposite from Table 4-8 in Mano**

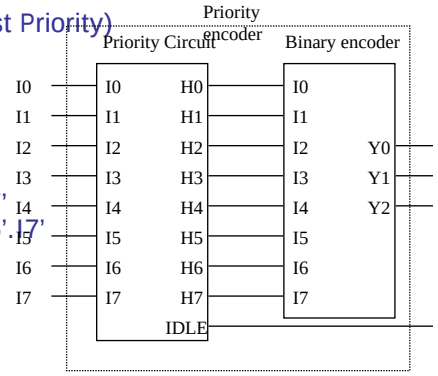
Inputs								Outputs			
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0	Idle
0	0	0	0	0	0	0	0	x	x	x	1
1	0	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	X	X	1	1	1	1	0

Priority encoders

- Assign priorities to the inputs
- When more than one input are asserted, the output generates the code of the input with the highest priority
- Priority Encoder :

  H7=I7 (Highest Priority)
  H6=I6.I7'
  H5=I5.I6'.I7'
  H4=I4.I5'.I6'.I7'
  H3=I3.I4'.I5'.I6'.I7'
  H2=I2.I3'.I4'.I5'.I6'.I7'
  H1=I1. I2'.I3'.I4'.I5'.I6'.I7'
  H0=I0.I1'. I2'.I3'.I4'.I5'.I6'.I7'
  IDLE= I0'.I1'. I2'.I3'.I4'.I5'.I6'.I7'
- Encoder

  Y0 = I1 + I3 + I5 + I7
  Y1 = I2 + I3 + I6 + I7
  Y2 = I4 + I5 + I6 + I7



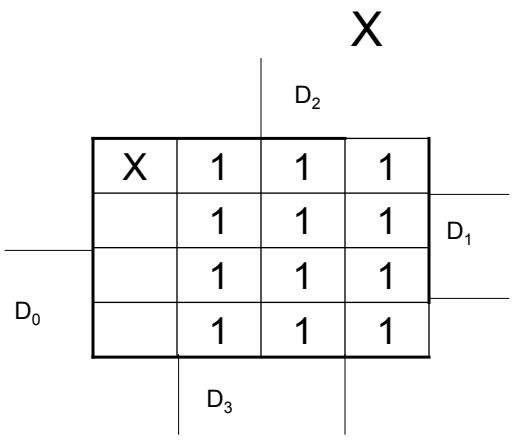
Priority Encoders

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	z
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	0	1

$X=D_2+D_3$

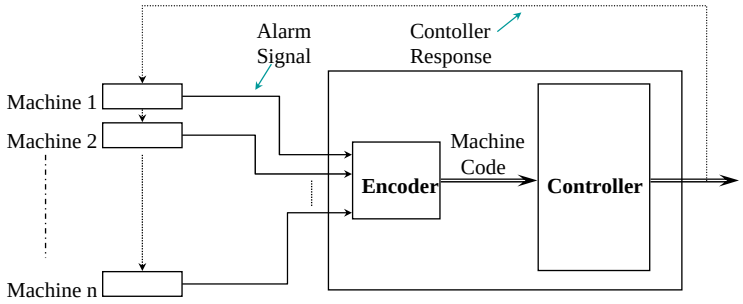
$Y=D_3+D_1D'_2$

$V=D_0+D_1+D_2+D_3$



Priority Encoders

- **Encoder identifies the requester and encodes the value**
- **Controller accepts digital inputs.**



Summary

- Decoder allows for generation of a single binary output from an input binary code
  - For an n-input binary decoder there are 2^n outputs
- Decoders are widely used in storage devices (e.g. memories)
  - We will discuss these in a few weeks
- Encoders all for data compression
- Priority encoders rank inputs and encode the highest priority input

Multiplexers

- A multiplexer is a combinational circuit that accepts binary information from one of many input lines and directs it to the output line.
- Which input to accept information from is selected by the selection lines.
- Usually there are n selection lines and 2^n input lines.
- A multiplexer can be combined with a common selection to select multiple bit selection, and Enable to control the operation. Figure 4-26 shows a quadruple 2-1 line multiplexer.

Multiplexers

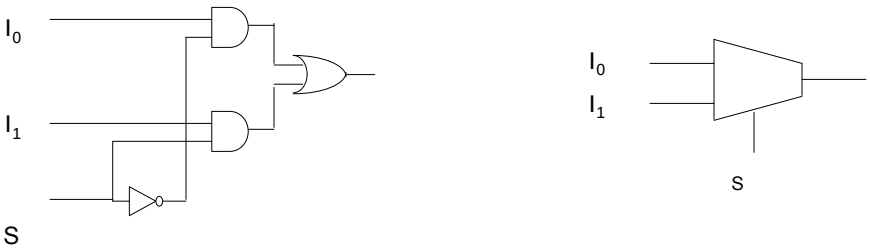
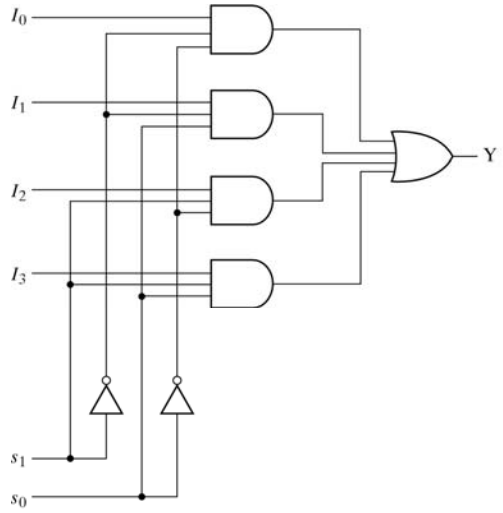


Figure 4-25

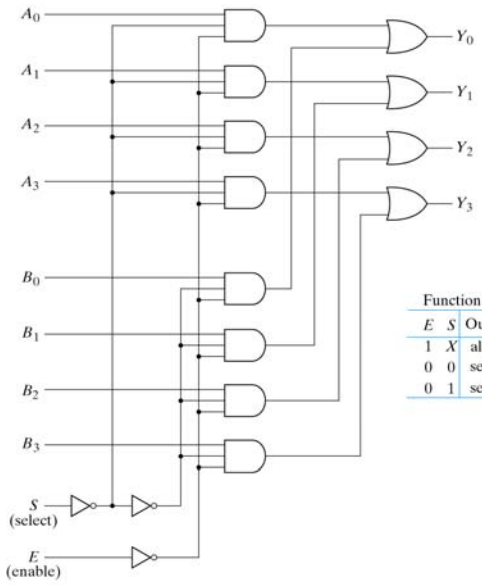


(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Fig. 4-25 4-to-1-Line Multiplexer



Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Fig. 4-26 Quadruple 2-to-1-Line Multiplexer

Function Implementation

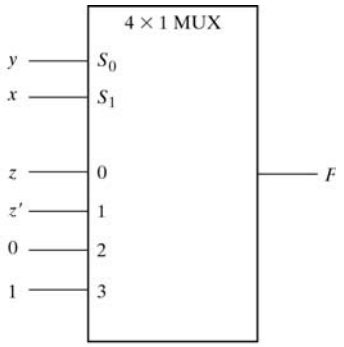
- We can consider the multiplexer to be a decoder that include the OR gates within.
- The OR minterms are generated by the function associated with the selection inputs.
- The rule to implement a function is as follows:

Function Implementation

- Assume that we have n variables
- Choose n-1 of them to be the selection lines of a 2^{n-1}-to-1 multiplexer.
- The selection lines chooses one of 2^{n-1} inputs.
- These inputs corresponds to the the truth table (2^n) entries taken 2 entries at a time.
- Assume the n^{th} variable is Z.
- These 2^{n-1} entries each is Z, Z', 0, or 1
- According to the entry number, the corresponding input is one of these 4 values. 4-27 and 4-28

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

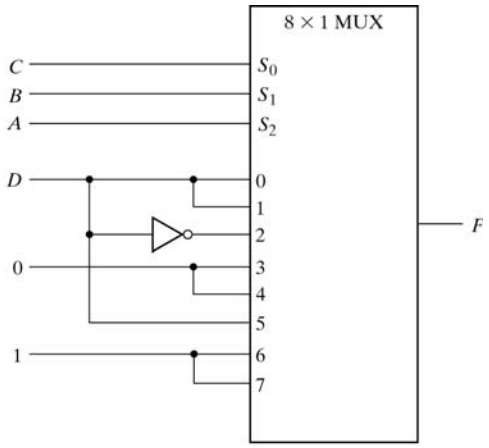
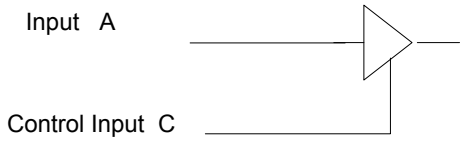


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

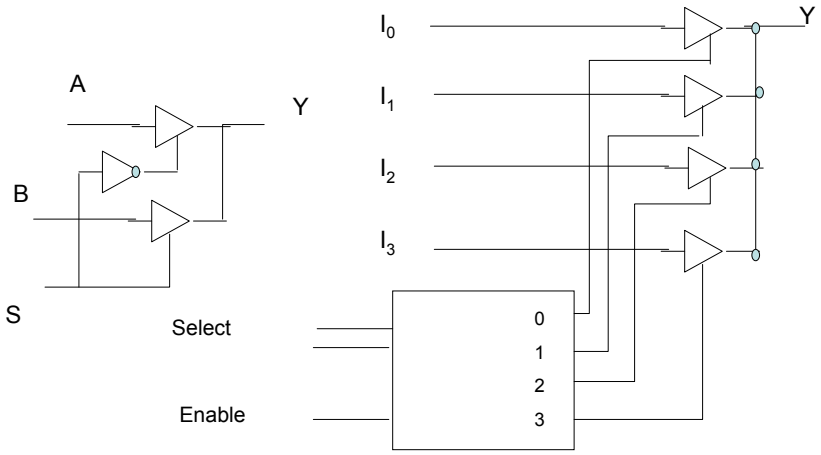
Three States gates

- The figure shows a three state buffer

$$Y = \begin{cases} Y = A & if\ C = 1 \\ Y = High\ Z & if\ C = 0 \end{cases}$$



Multiplexers with three-state gates



HDL for Combinational Circuits

- A module in Verilog can be described in any one of the following modeling techniques
  - Gate-level modeling using instantiation of primitive gates and user-defined modules.
  - Dataflow modeling using continuous assignment statements with **assign**
  - Behavioral modeling using procedural assignment statements with **always**

Verilog (gate-level)

- In gate level we have the following primitive gates (**and, nand, or, nor xor, xnor, not, buf**)
- The system assigns four-valued logic to every gate (0,1,z,x).
- The truth tables for the 4 most used gates is shown in the next slide

Verilog

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

Or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

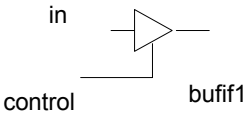
xor	0	1	x	z
0	0	1	0	0
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

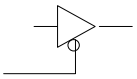
not	Input	output
	0	1
	1	0
	x	x
	z	x

Verilog 2-4 line decoder

```
//gate-level description of a 2-4 line decoder
module decoder_g1(A,B,E,D);
   input      A,B,E;
   output     [0:3]D;
   wireAnot, Bnot, Enot;
   not
       n1(Anot,A),
       n2(Bnot,B),
       n3(Enot,E);
nand
       n1(D[0],Anot,Bnot,Enot),
       n2(D[1],Anot,B,Enot),
       n3(D[2],A,Bnot,Enot),
       n4(D[3],A,B,Enot);
endmodule
```

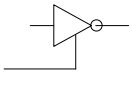
Three-state Gates

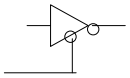




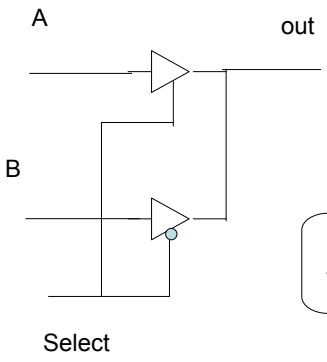
bufif1(OUT,A,control);

notif0(Y,B,enable);





Three State Gates



must be tri

```
module muxtri(A,B,select,OUT);
  input   A,B,select
  output  OUT;
  tri     OUT;
  bufif1(OUT,A,select);
  bufif0(OUT,B,select);
endmodule
```

Dataflow Modeling

```
//Dataflow modeling of a 2-4 line decoder
module decoder_df (A,B,E,D);
  input A,B,E;
  output [0:3] D;
  assign     D[0]=~(~A & ~B & ~E),
             D[1]=~(~A &  B & ~E),
             D[2]=~( A & ~B & ~E),
             D[3]=~( A &  B & ~E);
endmodule
```


Dataflow Modeling

```
//Dataflow modeling of a 4-bit adder
module binary_adder (A,B,C_in,SUM,C_OUT);
input [3:0]A,B;
input C_in;
output [3:0] SUM;
output C_out;
assign {C_out,SUM} = A+B;
endmodule
```

Dataflow Modeling

```
//Dataflow Modeling of a 4-bit
  comparator
module magcomp (A,B,ALSB,AGTB,AEQB);
  input [3:0] A,B;
  output ALTB, AGTB,AEQB;
  assign ALTB = (A < B),
         AGTB = (A>B),
         AEQB = (A==B);
endmodule
```


Dataflow Modeling

```
//Dataflow model for a 2-to-1 mux
module mux2x1_df(A,B,select,OUT);
  input A,B,select;
  output OUT;
  assign OUT= select? A : B;
endmodule
```

behavioral description

```
//Behavioral description of a 2-1 line MUX
   module mux2_1 (A,B,select,OUT);
   input A,B,select;
   output OUT;
   reg OUT;
   always @ (select or A or B)
        if (select == 1) OUT = A;
        else OUT=b;
   endmodule
```


Simulation (Test Bench)

- A test bench is a program for applying simulation to an HDL design.
- initial statements are executed at time 0
- always statements are executed always
- test module has no input or outputs
- The signals that are applied to the design module are declared as **reg**.
- The output of the design modules are declared as **wire**.

4-to-1 MUX

- Description of a 4-to-1 MUX using conditional
- operators:

```
module mux4to1 (w0, w1, w2, w3, S, f);
    input w0, w1, w2, w3;
    input [1:0] S;
    output f;
    assign f = S[1] ? (S[0] ? w3 : w2) : (S[0] ? w1 : w0);
endmodule
```



```
module mux4to1 (W, S, f);
input [0:3] W;
input [1:0] S;
output f;
reg f;
        always @(*)
        if (S == 0)
                f = W[0];
        else if (S == 1)
                f = W[1];
        else if (S == 2)
                f = W[2];
        else if (S == 3)
                f = W[3];
endmodule
```

 Description of a 4-to-1 MUX using a case statement:

```
module mux4to1 (W, S, f);
input [0:3] W;
input [1:0] S;
output f;
reg f;
always @(*)
        case (S)
                0: f = W[0];
                1: f = W[1];
                2: f = W[2];
                3: f = W[3];
        endcase
endmodule
```


 Verilog description of a priority encoder:

```
module priority (W, Y, z);
input [3:0] W;
output [1:0] Y;
output z;
reg [1:0] Y;
reg z;
always @(*)
        begin
                z = 1;
                casex (W)
                        4'b1xxx: Y = 3;
                        4'b01xx: Y = 2;
                        4'b001x: Y = 1;
                        4'b0001: Y = 0;
                        default: begin
                                z = 0 ;
                                Y = 2'bxx;
                                end
                endcase
        end
endmodule
```

 Verilog description of a 16-to-1 MUX constructed as a tree of 4-to-1 decoders:

```
module mux16to1 (W, S, f, M);
        input [0:15] W;
        input [3:0] S;
        output f;
        output [3:0] M;
        wire [0:3] M;
        mux4to1 Mux1 (W[0:3], S[1:0], M[0]);
        mux4to1 Mux2 (W[4:7], S[1:0], M[1]);
        mux4to1 Mux3 (W[8:11], S[1:0], M[2]);
        mux4to1 Mux4 (W[12:15], S[1:0], M[3]);
        mux4to1 Mux5 (M[0:3], S[3:2], f);
endmodule
```