Digital Logic Design

Chapter 4

Combinational Circuits

Logic Circuit Analysis

- Given a Logic Circuit, determine its Truth Table or Boolean Expression
- Procedure:
  - Label all gate outputs with arbitrary symbols
  - Determine Boolean expressions/Truth tables for each output in terms of either the inputs or the intermediate variables

Design Procedure

- The design of a combinational circuit involves the following steps:
  - Specification: How the circuit operates is clearly expressed
  - Formulation: Derivation of the truth table or the Boolean equations that define the relationship between inputs and outputs
  - Optimization: Algebraic or K-map optimization of the truth table and drawing the corresponding logic diagram
  - Technology Mapping: Tranform the logic diagram to a new diagram using the available implementation technology
  - Verification: Verify the correctness of the final design

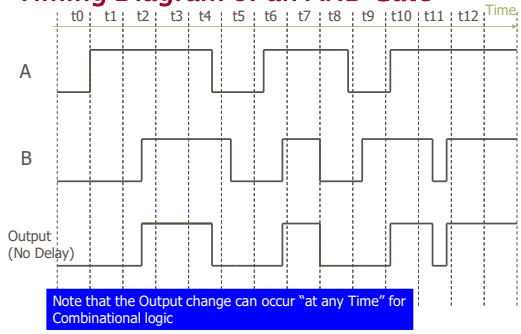
Combinational Logic

- Outputs, "at any time", are determined by the input combination
- When input change, output change immediately
  - Real circuits is imperfect and have "propagation delay"
- A combinational circuit
  - Performs logic operations that can be specified by a set of Boolean expressions
  - Can be built hierarchically

Timing Diagram

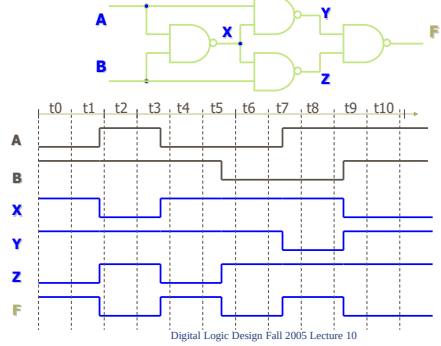
- Describe the functionality of a logic circuit across time
- Represented by a waveform
- For combinational logic, Output is a function of inputs

Timing Diagram of an AND Gate

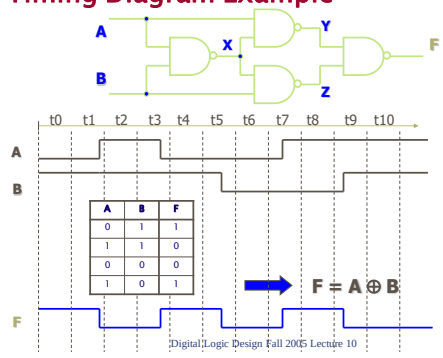


Note that the Output change can occur "at any Time" for Combinational logic

Timing Diagram Example



Timing Diagram Example



Combinational Logic

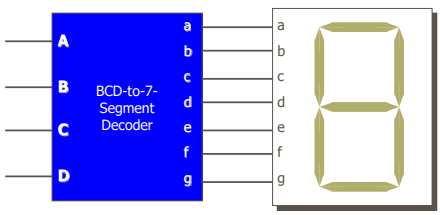
- Outputs, "at any time", are determined by the input combination
- We will discuss
  - Decoders / Encoders
  - Multiplexers / De-Multiplexers
  - Comparators
  - Parity Checkers / Generators
  - Binary Adders / Subtractors
  - Multipliers

Code Converters

- Convert from one code to another. Examples include
  - BCD to Seven_Segment_Decoder
  - Binary to Excess 3 Code Converter
- Design process:
  - Every bit of the output code is represented as a function of the input code bits, all such functions are reduced and implemented

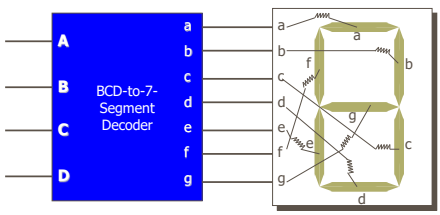
BCD-to-7-Segment Converter

- A kind of decoder



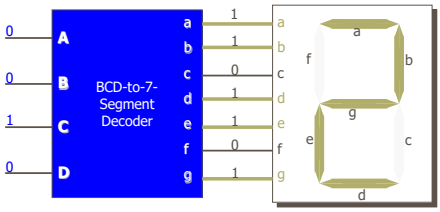
BCD-to-7-Segment Converter

- A kind of decoder



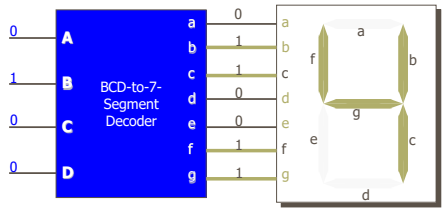
BCD-to-7-Segment Converter

- Decode "2" and show 2



BCD-to-7-Segment Converter

- Decode "4" and show 4

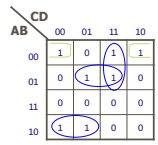


BCD-to-7-Seg. Converter Truth Table

	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	0	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	0	0	1	1	9
>10	All other inputs				0	0	0	0	0	0	0	

Design Each Output Individually —"a"

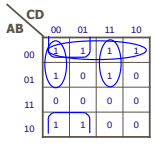
	A	B	C	D	a
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
>10	All other inputs				0



$$a = \overline{A}\overline{B}\overline{D} + \overline{A}CD + \overline{A}B\overline{D} + A\overline{B}\overline{C}$$

Design Each Output Individually —"b"

	A	B	C	D	b
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
>10	All other inputs				0



$$b = \overline{B}\overline{C} + \overline{A}\overline{B} + \overline{A}\overline{C}\overline{D} + \overline{A}CD$$

Decoders and Encoders

- Binary Decoders
- Binary Encoders
- Priority Encoders

Decoders

- n inputs-to-2^n outputs
  - –Each state corresponds to a unique input combination

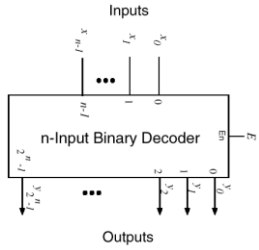
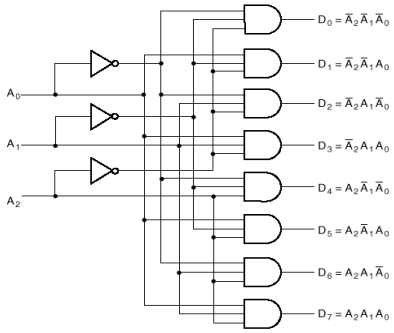


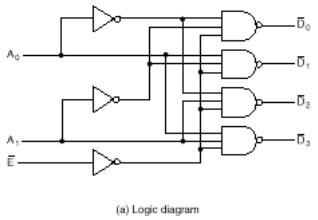
TABLE 3-4
Truth Table for 3–to–8-Line Decoder

Inputs			Outputs							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

3-to-8 Line Decoder



A 2-to-4-Line Decoder



(a) Logic diagram

$\bar{E}$	A_1	A_0	$\bar{D}_0$	$\bar{D}_1$	$\bar{D}_2$	$\bar{D}_3$
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
1	X	X	1	1	1	1

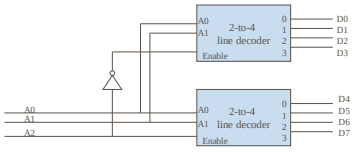
(b) Truth table

$\bar{D}_0 = \bar{E}\,\bar{A}_1\,\bar{A}_0$
$\bar{D}_1 = \bar{E}\,\bar{A}_1\,A_0$
$\bar{D}_2 = \bar{E}\,A_1\,\bar{A}_0$
$\bar{D}_3 = \bar{E}\,A_1\,A_0$

(c) Logic Equations

Large Sized Decoders

- It is possible to connect smaller sized decoders to form a larger decoder



4-input Tree Decoder

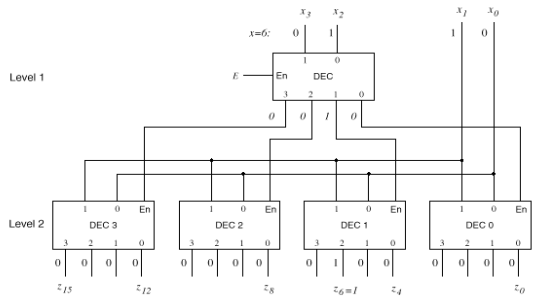
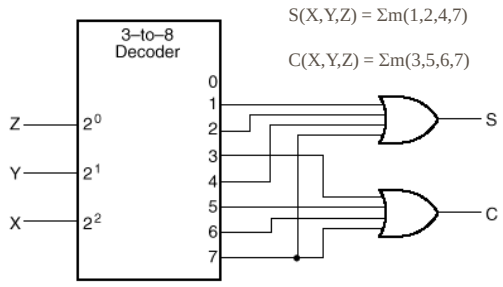


Figure 9.8. 4- input tree decoder

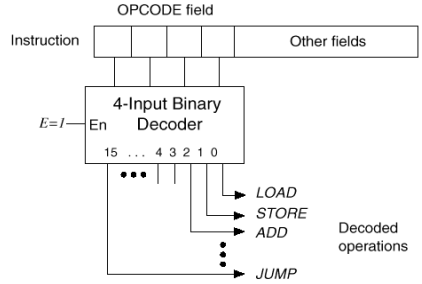
Implementing a Binary Adder a Using Decoder



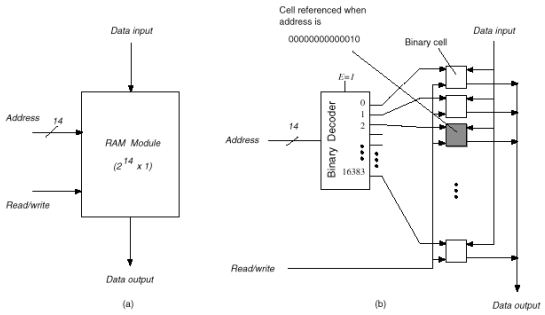
$S(X,Y,Z) = \Sigma m(1,2,4,7)$

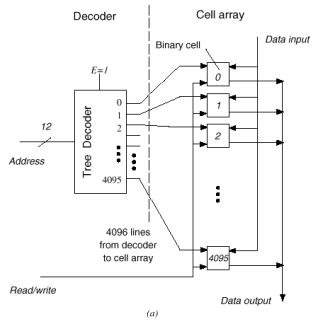
$C(X,Y,Z) = \Sigma m(3,5,6,7)$

Decoder uses



Decoder uses





Decoders and Encoders

- Binary Decoders
- Binary Encoders
- Priority Encoders

Encoders

- Binary: 2^n inputs–to–n outputs
  - It is assumed that only one input is TRUE or ONE at one time: Most of the input combinations are Don't Care
  - Implementation is using OR gates only; one gate is needed for each output

$A0 = D1 + D3$

$A1 = D2 + D3$

INPUTS				OUTPUTS	
D3	D2	D1	D0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Binary Encoders

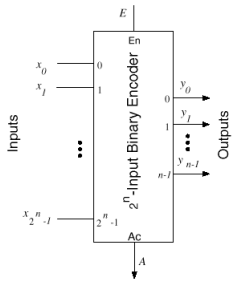
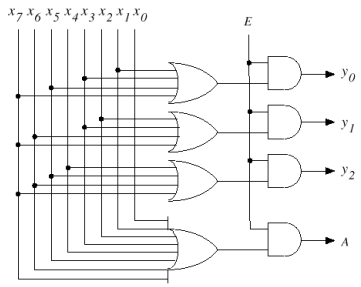


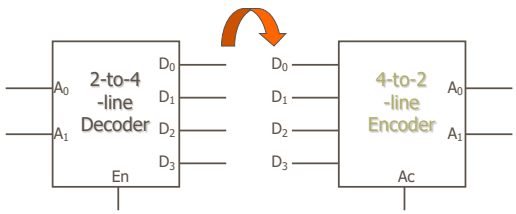
TABLE 3-5
Truth Table for Octal-to-Binary Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$



M-to-N-Line Encoder ($M \leq 2^N$)



8-to-3 Encoder

D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	**1**	0	0	0	1
0	0	0	0	0	**1**	0	0	0	1	0
0	0	0	0	**1**	0	0	0	0	1	1
0	0	0	**1**	0	0	0	0	1	0	0
0	0	**1**	0	0	0	0	0	1	0	1
0	**1**	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Since Dx=1 only in one column at a time

A0 = D1 + D3 + D5 + D7
A1 = D2 + D3 + D6 + D7
A2 = D4 + D5 + D6 + D7

8-to-3 Priority Encoder

D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0	Active
0	0	0	0	0	0	0	**0**	0	0	0	0
0	0	0	0	0	0	0	**1**	0	0	0	1
0	0	0	0	0	0	**1**	**X**	0	0	1	1
0	0	0	0	0	**1**	**X**	**X**	0	1	0	1
0	0	0	0	**1**	**X**	**X**	**X**	0	1	1	1
0	0	0	**1**	**X**	**X**	**X**	**X**	1	0	0	1
0	0	**1**	**X**	**X**	**X**	**X**	**X**	1	0	1	1
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	0	1
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	1	1

4-to-2 Priority Encoder

D3	D2	D1	D0	A1	A0	Active
0	0	0	**0**	0	0	0
0	0	0	**1**	0	0	1
0	0	**1**	**X**	0	1	1
0	**1**	**X**	**X**	1	0	1
1	**X**	**X**	**X**	1	1	1

For A1

D3 D2 \ D1 D0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$A1 = D2 + D3$$

Or using simplification property

$$A1 = D3 + \overline{D3}D2 = D3 + D2$$

4-to-2 Priority Encoder

D3	D2	D1	D0	A1	A0	Active
0	0	0	**0**	0	0	0
0	0	0	**1**	0	0	1
0	0	**1**	**X**	0	1	1
0	**1**	**X**	**X**	1	0	1
1	**X**	**X**	**X**	1	1	1

For A0

D3 D2 \ D1 D0	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$A0 = D3 + \overline{D2}D1$$

Or using simplification property

$$A0 = D3 + \overline{D3}\,\overline{D2}D1 = D3 + \overline{D2}D1$$

4-to-2 Priority Encoder

D3	D2	D1	D0	A1	A0	Active
0	0	0	**0**	0	0	0
0	0	0	**1**	0	0	1
0	0	**1**	**X**	0	1	1
0	**1**	**X**	**X**	1	0	1
1	**X**	**X**	**X**	1	1	1

For Active

D3 D2 \ D1 D0	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$Active = D3 + D2 + D1 + D0$$

8-to-3 Priority Encoder (A2)

D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0	Active
0	0	0	0	0	0	0	**0**	0	0	0	0
0	0	0	0	0	0	0	**1**	0	0	0	1
0	0	0	0	0	0	**1**	**X**	0	0	1	1
0	0	0	0	0	**1**	**X**	**X**	0	1	0	1
0	0	0	0	**1**	**X**	**X**	**X**	0	1	1	1
0	0	0	**1**	**X**	**X**	**X**	**X**	1	0	0	1
0	0	**1**	**X**	**X**	**X**	**X**	**X**	1	0	1	1
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	0	1
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	1	1

$$A2 = \overline{D7}D6\overline{D5}D4 + D7\overline{D6}D5 + D7\overline{D6} + D7$$
$$= \overline{D6}\,\overline{D5}D4 + \overline{D6}D5 + D6 + D7$$
$$= \overline{D5}D4 + D5 + D6 + D7$$
$$= D4 + D5 + D6 + D7$$

8-to-3 Priority Encoder (A1)

D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0	Active
0	0	0	0	0	0	0	**0**	0	0	0	0
0	0	0	0	0	0	0	**1**	0	0	0	1
0	0	0	0	0	0	**1**	**X**	0	0	1	1
0	0	0	0	0	**1**	**X**	**X**	0	1	0	1
0	0	0	0	**1**	**X**	**X**	**X**	0	1	1	1
0	0	0	**1**	**X**	**X**	**X**	**X**	1	0	0	1
0	0	**1**	**X**	**X**	**X**	**X**	**X**	1	0	1	1
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	0	1
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	1	1

$$A1 = \overline{D7}\,\overline{D6}D5\overline{D4}\overline{D3}D2 + \overline{D7}D6\overline{D5}D4\overline{D3} + \overline{D7}D6 + D7$$
$$= \overline{D6}\,\overline{D5}D4\overline{D3}D2 + \overline{D6}D5\overline{D4}D3 + D6 + D7$$
$$= \overline{D5}D4\overline{D3}D2 + \overline{D5}D4D3 + D6 + D7$$
$$= \overline{D5}D4(\overline{D3}D2 + D3) + D6 + D7$$
$$= \overline{D5}D4D2 + \overline{D5}D4D3 + D6 + D7$$

8-to-3 Priority Encoder (A0)

D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0	Active
0	0	0	0	0	0	0	**0**	0	0	0	0
0	0	0	0	0	0	0	**1**	0	0	0	1
0	0	0	0	0	0	**1**	**X**	0	0	1	1
0	0	0	0	0	**1**	**X**	**X**	0	1	0	1
0	0	0	0	**1**	**X**	**X**	**X**	0	1	1	1
0	0	0	**1**	**X**	**X**	**X**	**X**	1	0	0	1
0	0	**1**	**X**	**X**	**X**	**X**	**X**	1	0	1	1
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	0	1
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	1	1

$$A0 = \overline{D7}\,\overline{D6}\,\overline{D5}D4\overline{D3}\overline{D2}D1 + \overline{D7}\,\overline{D6}\,\overline{D5}D4\overline{D3} + \overline{D7}\,\overline{D6}D5 + D7$$
$$= \overline{D6}\,\overline{D5}D4\overline{D3}\overline{D2}D1 + \overline{D6}\,\overline{D5}D4\overline{D3} + \overline{D6}D5 + D7$$
$$= \overline{D6}(\overline{D5}D4\overline{D3}\overline{D2}D1 + \overline{D5}D4\overline{D3} + D5) + D7 = \overline{D6}(\overline{D4}\,\overline{D3}\,\overline{D2}D1 + \overline{D4}D3 + D5) + D7$$
$$= \overline{D6}(\overline{D4}(\overline{D3}\,\overline{D2}D1 + D3) + D5) + D7 = \overline{D6}(\overline{D4}(\overline{D2}D1 + D3) + D5) + D7$$
$$= \overline{D6}D4\overline{D2}D1 + \overline{D6}\,\overline{D4}D3 + \overline{D6}D5 + D7$$

8-to-3 Priority Encoder (All)

D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0	Active
0	0	0	0	0	0	0	**0**	0	0	0	0
0	0	0	0	0	0	0	**1**	0	0	0	1
0	0	0	0	0	0	**1**	**X**	0	0	1	1
0	0	0	0	0	**1**	**X**	**X**	0	1	0	1
0	0	0	0	**1**	**X**	**X**	**X**	0	1	1	1
0	0	0	**1**	**X**	**X**	**X**	**X**	1	0	0	1
0	0	**1**	**X**	**X**	**X**	**X**	**X**	1	0	1	1
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	0	1
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	1	1	1	1

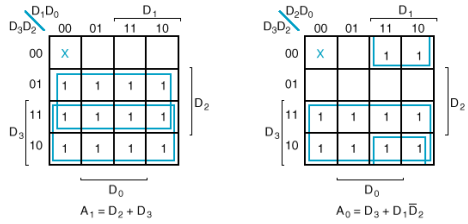
$A2 = \overline{D4} + \overline{D5} + \overline{D6} + D7$

$A1 = \overline{D5}\,\overline{D4}D2 + \overline{D5}\,\overline{D4}D3 + D7$

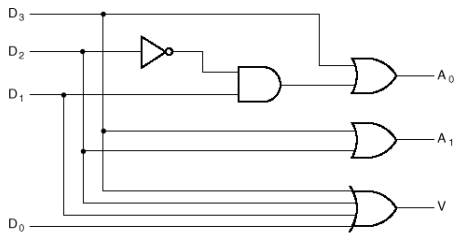
$A0 = \overline{D6}\,\overline{D4}\,\overline{D2}D1 + \overline{D6}\,\overline{D4}D3 + \overline{D6}\,\overline{D5} + D7$

$Active = D1 + D2 + D3 + D4 + D5 + D6 + D7$

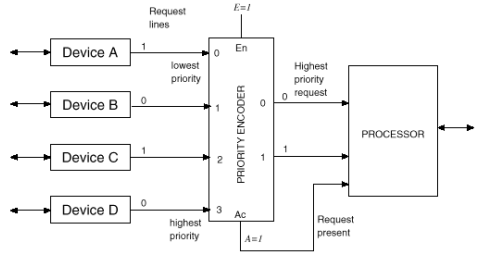
Maps of Priority Encoder



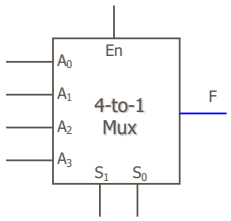
Logic Diagram of a 4-Input Priority Encoder



Uses of priority encoders

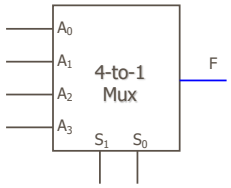


Multiplexers (Mux)



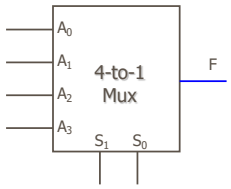
- Functionality: Selection of a particular input
- Route 1 of N inputs (A) to the output F
- En(able) bit can disable the route and set F to 0

Multiplexers (Mux) w/out Enable



S1	S0	A3	A2	A1	A0	F
0	0	X	X	X	0	0
0	1	X	X	0	X	0
1	0	X	0	X	X	0
1	1	0	X	X	X	0
0	0	X	X	X	1	1
0	1	X	X	1	X	1
1	0	X	1	X	X	1
1	1	1	X	X	X	1

Multiplexers (Mux) w/out Enable

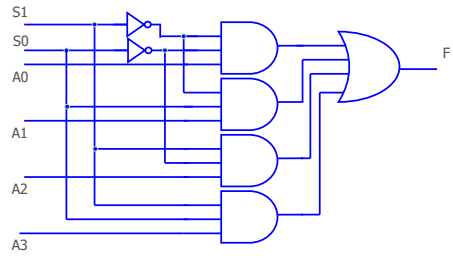


S1	S0	F
0	0	A0
0	1	A1
1	0	A2
1	1	A3

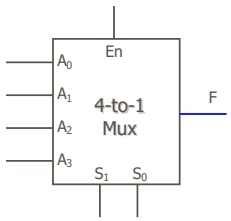
$$F = \overline{S_1}\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3$$

Logic Diagram of a 4-to-1 Mux

$$F = \overline{S_1}\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3$$



Multiplexers (Mux) w/ Enable

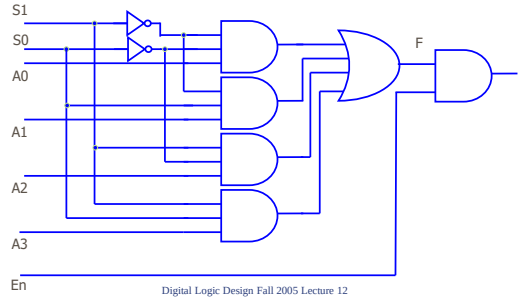


En	S1	S0	F
0	X	X	0
1	0	0	A0
1	0	1	A1
1	1	0	A2
1	1	1	A3

$$F = En \cdot (\overline{S_1}\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3)$$
$$= En\overline{S_1}\overline{S_0}A_0 + En\overline{S_1}S_0A_1 + EnS_1\overline{S0}A_2 + EnS_1S_0A_3$$

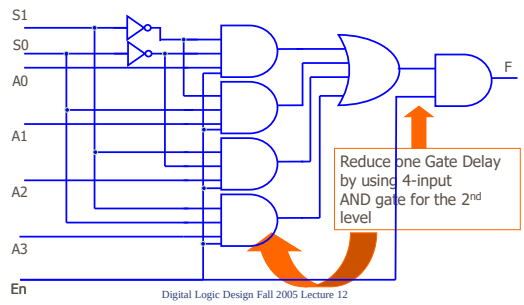
4-to-1 Mux w/ Enable Logic

$$F = En \cdot (\overline{S_1}\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3)$$

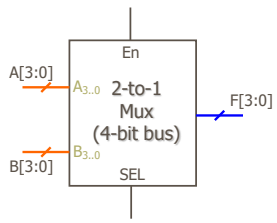


4-to-1 Mux w/ Enable Logic

$$F = En\overline{S_1}\overline{S_0}A_0 + En\overline{S_1}S_0A_1 + EnS_1\overline{S0}A_2 + EnS_1S_0A_3$$

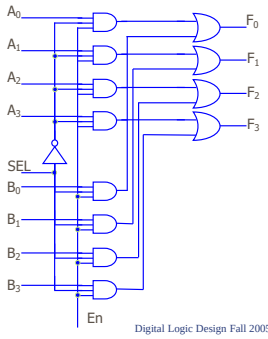


Quadruple 2-to-1 Line Mux



En	SEL	F[3:0]
0	X	0000
1	0	A[3:0]
1	1	B[3:0]

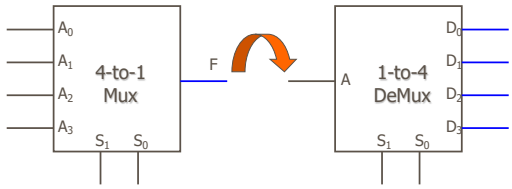
Quadruple 2-to-1 Line Mux



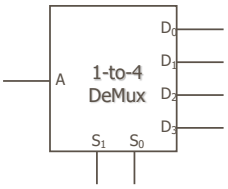
$F_x = A_x \cdot En \cdot \overline{SEL} + B_x \cdot En \cdot SEL$

En	SEL	F[3:0]
0	X	0000
1	0	A[3:0]
1	1	B[3:0]

Demultiplexers (DeMux)



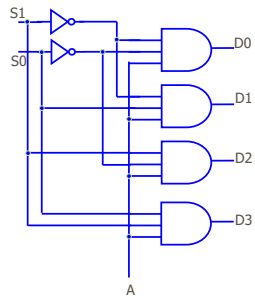
DeMux Operations



S1	S0	D3	D2	D1	D0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

$$D_0 = \overline{S_1}\,\overline{S_0}A$$
$$D_1 = \overline{S_1}S_0A$$
$$D_2 = S_1\overline{S_0}A$$
$$D_3 = S_1S_0A$$

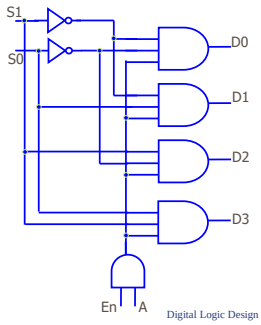
DeMux Operations



S1	S0	D3	D2	D1	D0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

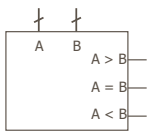
$$D_0 = \overline{S_1}\,\overline{S_0}A$$
$$D_1 = \overline{S_1}S_0A$$
$$D_2 = S_1\overline{S_0}A$$
$$D_3 = S_1S_0A$$

DeMux Operations w/ Enable



En	S1	S0	D3	D2	D1	D0
0	X	X	0	0	0	0
1	0	0	0	0	0	A
1	0	1	0	0	A	0
1	1	0	0	A	0	0
1	1	1	A	0	0	0

1-bit Magnitude Comparator

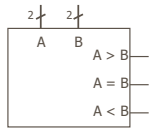


A	B	A?B
0	0	A=B
0	1	A<B
1	0	A>B
1	1	A=B

Single bit comparison

$$A > B \rightarrow A\overline{B}$$
$$A < B \rightarrow \overline{A}B$$
$$A = B \rightarrow \overline{A \oplus B}$$

2-bit Magnitude Comparator (unsigned)

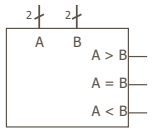


A_1A_0
B_1B_0

Two-bit comparison

$$(A > B) \rightarrow A_1\overline{B_1} + (\overline{A_1 \oplus B_1})A_0\overline{B_0}$$
$$(A < B) \rightarrow \overline{A_1}B_1 + (\overline{A_1 \oplus B_1})\overline{A_0}B_0$$
$$(A = B) \rightarrow (\overline{A_1 \oplus B_1})(\overline{A_0 \oplus B_0})$$

2-bit Magnitude Comparator (unsigned)

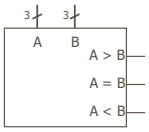


A_1A_0
B_1B_0

Two-bit comparison

$$\text{Assign} \quad X_n = \overline{A_n \oplus B_n}$$
$$(A > B) \rightarrow A_1\overline{B_1} + X_1A_0\overline{B_0}$$
$$(A < B) \rightarrow \overline{A_1}B_1 + X_1\overline{A_0}B_0$$
$$(A = B) \rightarrow X_1X_0$$

3-bit Magnitude Comparator (unsigned)

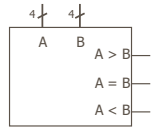


$A_2A_1A_0$
$B_2B_1B_0$

Three-bit comparison

$$\text{Assign} \quad X_n = \overline{A_n \oplus B_n}$$
$$(A > B) \rightarrow A_2\overline{B_2} + X_2A_1\overline{B_1} + X_2X_1A_0\overline{B_0}$$
$$(A < B) \rightarrow \overline{A_2}B_2 + X_2\overline{A_1}B_1 + X_2X_1\overline{A_0}B_0$$
$$(A = B) \rightarrow X_2X_1X_0$$

4-bit Magnitude Comparator (unsigned)



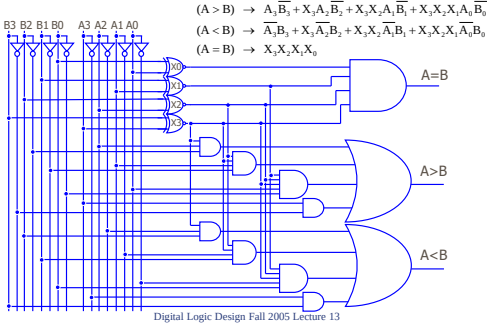
$A_3A_2A_1A_0$
$B_3B_2B_1B_0$

Four-bit comparison

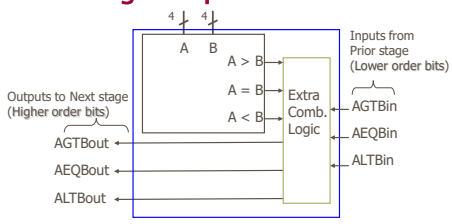
$$\text{Assign} \quad X_n = \overline{A_n \oplus B_n}$$
$$(A > B) \rightarrow A_3\overline{B_3} + X_3A_2\overline{B_2} + X_3X_2A_1\overline{B_1} + X_3X_2X_1A_0\overline{B_0}$$
$$(A < B) \rightarrow \overline{A_3}B_3 + X_3\overline{A_2}B_2 + X_3X_2\overline{A_1}B_1 + X_3X_2X_1\overline{A_0}B_0$$
$$(A = B) \rightarrow X_3X_2X_1X_0$$

4-bit Magnitude Comparator

$$\text{Assign} \quad X_n = \overline{A_n \oplus B_n}$$
$$(A > B) \rightarrow A_3\overline{B_3} + X_3A_2\overline{B_2} + X_3X_2A_1\overline{B_1} + X_3X_2X_1A_0\overline{B_0}$$
$$(A < B) \rightarrow \overline{A_3}B_3 + X_3\overline{A_2}B_2 + X_3X_2\overline{A_1}B_1 + X_3X_2X_1\overline{A_0}B_0$$
$$(A = B) \rightarrow X_3X_2X_1X_0$$

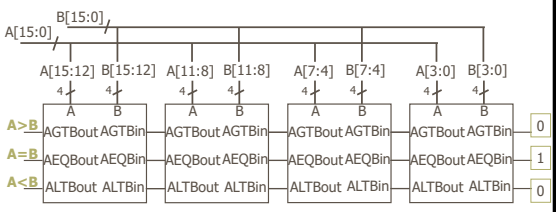


Cascading Comparator



AGTBout = (A>B) + (A=B) · AGTBin
AEQBout = (A=B) · AEQBin
ALTBout = (A<B) + (A=B) · ALTBin

16-bit Cascading Comparator

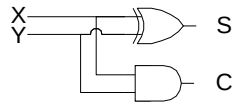


Arithmetic Circuits: Adders

- Half Adder

S = X'Y+Y'X
C = XY

Inputs		Outputs	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

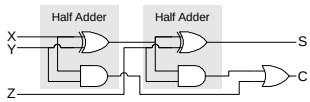


Adders: Full Adder

- Adder

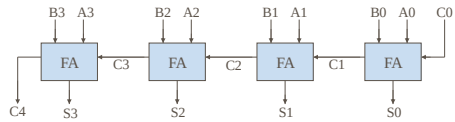
INPUTS			OUTPUTS	
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S = Σm(1,2,4,7)
C = Σm(3,5,6,7)



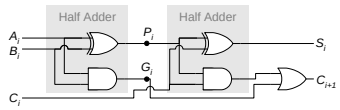
Multi-bit Adders

- Challenge: No of inputs for a 16 bit adder = 32+1
  - Truth Table is not practical; Reduction will take forever
- Solution: Use Single-bit adder and cascade
  - Ripple Carry Adder:



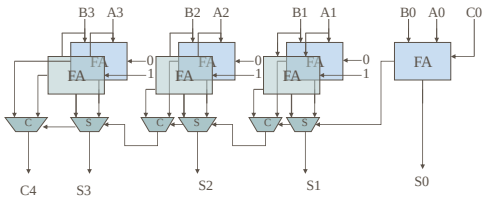
Carry Path Delays and Enhancements

- Carry-path is long: high-speed Adder circuits can not be implemented using ripple carry method
  - No. of Carry Logic Levels for *n*-bit Addition = $2n + 2$
- Delay optimization can be achieved by Logic Manipulation
  - Define $P_i = A_i \oplus B_i$ & $G_i = A_i.B_i$
  - Then $S_i = P_i \oplus C_i$ & $C_{i+1} = G_i + P_i.C_i$

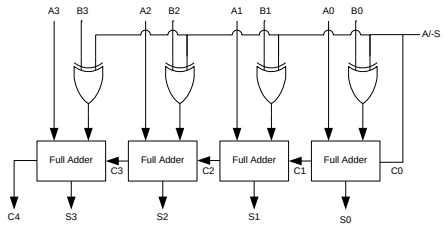


Carry Select Adder

- Evaluate all carry's in parallel and then select the correct one

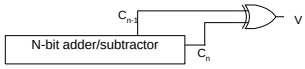


Binary Adder/Subtractor



Handling Overflow in Adders

- If Sum does not fit in *n* bits, we say overflow has happened
  - It is important to know that overflow occurred so that it can be handled appropriately
  - In case of Unsigned number, Overflow = end carry
  - With Signed numbers, overflow can only occur if the two numbers are of the same sign

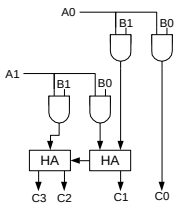


Multipliers

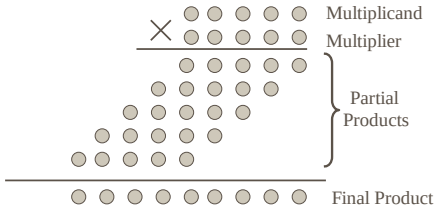
Binary Multiplication

In1	In0	product
0	0	0
0	1	0
1	0	0
1	1	1

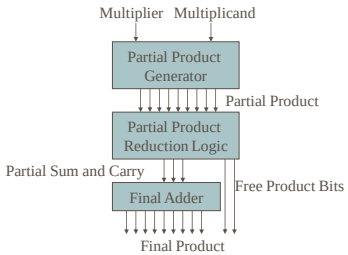
		B1	B0
		A1	A0
		A0B1	A0B0
	A1B1	A1B0	
C3	C2	C1	C0



Multi-bit Multiplication



Multi-bit Parallel Multiplier



Parallel Multipliers

- Consist of Three main components
  - Partial product generator:
    - Implemented with AND Gates
  - Partial Product Reduction Logic
    - Many implementation like
      - Carry Save, Dual Carry Save
      - Wallac Tree, Dadda Tree
      - Vuillemin Reduction
  - Final Adder
    - Ripple Carry, Carry Look-ahead or Carry Save

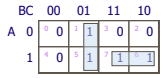
Hazards and Glitches

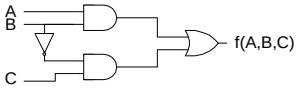
- A hazard is the possibility of an unwanted transient (spike or glitch). In a particular circuit implementation of the function, a glitch may or may not occur depending the actual delays in the circuit
- CAUSE: Different Propagation Delays in paths reaching the input to a gate
- Hazards: The Circuit may not behave as expected
  - Static Hazards
    - The outputs may change state for a brief time and then return to the normal: GLITCH
  - Dynamic Hazards
    - The output may contain glitches when a state change of the output occurs

Hazards Example

F(A,B,C)=Σm(1,5,6,7)

Reduced Implementation





Hazard Covers

- Add additional gates to cover states that can cause glitches. These gates are not required functionally, however, their absence can cause glitches.