CPE/EE 422/522
Advanced Logic Design
L02

Electrical and Computer Engineering
University of Alabama in Huntsville

Outline

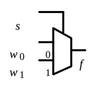
- What we know
  - Laws and Theorems of Boolean Algebra
  - Simplification of Logic Expressions
    - Using Laws and Theorems of Boolean Algebra or Using K-maps
  - Design Using only NAND or only NOR gates
  - Tri-state buffers
  - Basic Combinational Building Blocks
    - Multiplexers, Decoders, Encoders, ...
- What we do not know
  - Hazards in Combinational Networks
  - How to implement functions using ROMs, PLAs, and PALs
  - Sequential Networks (if time)

Review: Combinational-Circuit Building Blocks

- Multiplexers
- Decoders
- Encoders
- Code Converters
- *Comparators*
- *Adders/Subtractors*
- *Multipliers*
- *Shifters*

Multiplexers: 2-to-1 Multiplexer

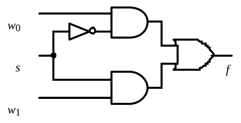
- Have number of data inputs, one or more select inputs, and one output
  - It passes the signal value on one of data inputs to the output



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table



(c) Sum-of-products circuit

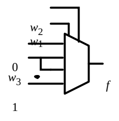
$f = s' w_0 + s w_1$

Review: Synthesis of Logic Functions Using Muxes

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

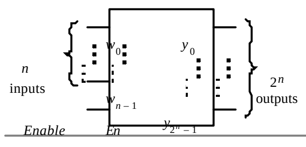
(a) Modified truth table



(b) Circuit

Decoders: n-to-2ⁿ Decoder

- Decode encoded information: n inputs, 2ⁿ outputs
- If En = 1, only one output is asserted at a time
- One-hot encoded output
  - m-bit binary code where exactly one bit is set to 1



$y_0 = w_{n-1}' .. w_1' w_0' En$

$y_1 = w_{n-1}' .. w_1' w_0 En$

$y_2 = w_{n-1}' .. w_1 w_0' En$

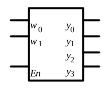
…

$y_{2^n-1} = w_{n-1} .. w_1 w_0 En$

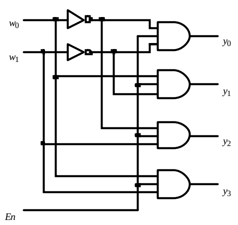
Decoders: 2-to-4 Decoder

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table



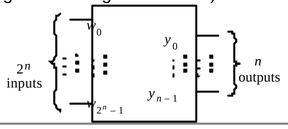
(b) Graphic symbol



(c) Logic circuit

Encoders

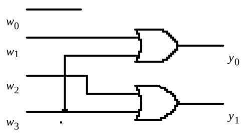
- Opposite of decoders
  - Encode given information into a more compact form
- Binary encoders
  - 2^n inputs into n-bit code
  - Exactly one of the input signals should have a value of 1, and outputs present the binary number that identifies which input is equal to 1
- Use: reduce the number of bits (transmitting and storing information)



Encoders: 4-to-2 Encoder

w_3	w_2	w_1	w_0	y_1	y_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) Truth table



(b) Circuit

Encoders: Priority Encoders

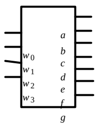
- Each input has a priority level associated with it
- The encoder outputs indicate the active input that has the highest priority

(a) Truth table for a 4-to-2 priority encoder

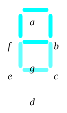
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Code Converters

- Convert from one type of input encoding to a different output encoding
  - E. g., BCD-to-7-segment decoder



(a) Code converter



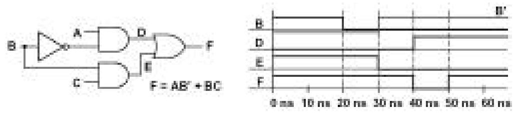
(b) 7-segment display

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) Truth table

Hazards in Combinational Networks

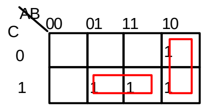
- What are hazards in CM?
  - Unwanted switching transients at the output (glitches)
- Example
  - ABC = 111, B changes to 0
  - Assume each gate has propagation delay of 10ns

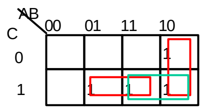


Hazards in Combinational Networks

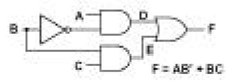
- Occur when different paths from input to output have different propagation delays
- Static 1-hazard
  - a network output momentarily go to the 0 when it should remain a constant 1
- Static 0-hazard
  - a network output momentarily go to the 1 when it should remain a constant 0
- Dynamic hazard
  - if an output change three or more times, when the output is supposed to change from 0 to 1 (1 to 0)

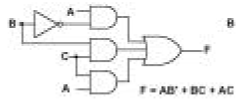
Hazards in Combinational Circuits





$f = AB' + BC$ $f = AB' + BC + AC$





To avoid hazards:
every par of adjacent 1s should be covered by a 1-term

Hazards in Combinational Circuits

Why do we care about hazards?

- Combinational networks
  - don't care – the network will function correctly
- Synchronous sequential networks
  - don't care - the input signals must be stable within setup and hold time of flip-flops
- Asynchronous sequential networks
  - hazards can cause the network to enter an incorrect state
  - circuitry that generates the next-state variables must be hazard-free
- Power consumption is proportional to the number of transitions

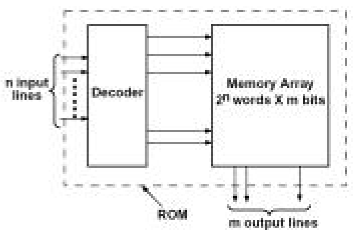
Programmable Logic Devices

- Read Only Memories (ROMs)
- Programmable Logic Arrays (PLAs)
- Programmable Array Logic Devices (PALs)

Read-Only Memories

- Store binary data
  - data can be read out whenever desired
  - cannot be changed under normal operating conditions
- n input lines, m output lines => array of 2^n m-bit words
  - Input lines serve as an address to select one of 2^n words
- Use ROM to implement logic functions?
  - n variables, m functions

Basic ROM Structure

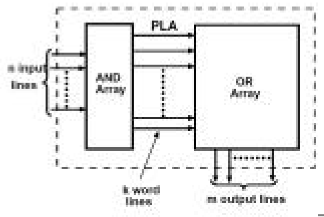


ROM Types

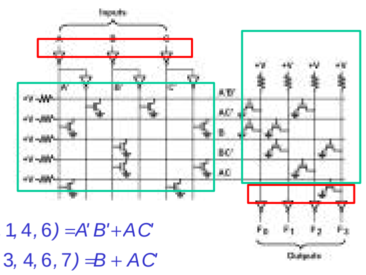
- Mask-programmable ROM
  - Data is permanently stored (include or omit the switching elements)
  - Economically feasible for a large quantity
- EPROM (Erasable Programmable ROM)
  - Use special charge-storage mechanism to enable or disable the switching elements in the memory array
  - PROM programmer is used to provide appropriate voltage pulses to store electronic charges
  - Data is permanent until erased using an ultraviolet light
  - EEPROM – Electrically Erasable PROM
    - erasure is accomplished using electrical pulses (can be reprogrammed typically 100 to 1000 times)
  - Flash memories - similar to EEPROM except they use a different charge-storage mechanism
    - usually have built-in programming and erase capability, so the data can be written to the flash memory while it is in place, without the need for a separate programmer

Programmable Logic Arrays (PLAs)

- Perform the same function as a ROM
  - *n* inputs and *m* outputs – *m* functions of *n* variables
  - AND array – realizes product terms of the input variables
  - OR array – ORs together the product terms



PLA: 3 inputs, 5 p.t., 4 outputs



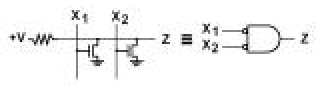
$F_0 = \sum m(0,1,4,6) = A'B' + AC'$

$F_1 = \sum m(2,3,4,6,7) = B + AC'$

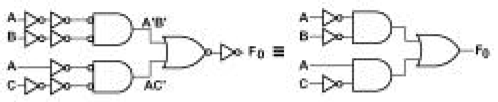
$F_2 = \sum m(0,1,2,6) = A'B' + BC'$

$F_3 = \sum m(2,3,5,6,7) = AC + B$

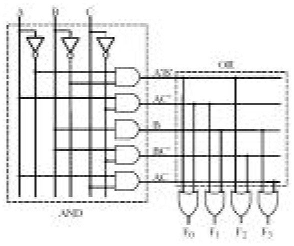
nMOS NOR Gate



$F_0 = \sum m(0,1,4,6) = A'B' + AC'$



AND-OR Array Equivalent



Modified Truth Table for PLA

$F_0 = \sum m(0,1,4,6) = A'B' + AC'$

$F_1 = \sum m(2,3,4,6,7) = B + AC'$

$F_2 = \sum m(0,1,2,6) = A'B' + BC'$

$F_3 = \sum m(2,3,5,6,7) = AC + B$

- 0 – variable is complemented
- 1 – variable is not complemented
- - – not present in the term

Product Term	Inputs			Outputs			
	A	B	C	F0	F1	F2	F3
A'B'	0	0	-	1	0	1	0
AC'	1	-	0	1	1	0	0
B	0	1	-	0	1	0	1
BC'	-	1	0	0	0	1	0
AC	1	-	1	0	0	0	1

Using PLA: An Example

$F_1 = \sum m(2,3,5,7,8,9,10,11,13,15)$

$F_2 = \sum m(2,3,5,6,7,10,11,14,15)$

$F_3 = \sum m(6,7,8,9,13,14,15)$

$F_1 = bd + b'c + ab'$

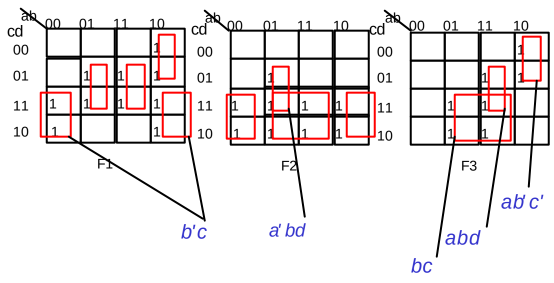
$F_2 = c + a'bd$

$F_3 = bc + ab'c' + abd$

Eight different product terms are required!?

For PLA we want to minimize
the total number of product terms,
not the number of product terms for each function separately!

Using PLA: An Example



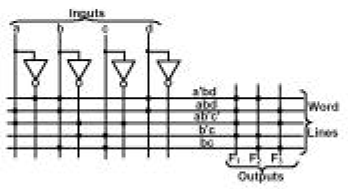
Using PLA: An Example

a	b	c	d	F_1	F_2	F_3
0	1	–	1	1	1	0
1	1	–	1	1	0	1
1	0	0	–	1	0	1
–	0	1	–	1	1	0
–	1	1	–	0	1	1

$F_1 = a'bd + abd + ab'c' + b'c$

$F_2 = a'bd + b'c + bc$

$F_3 = abd + ab'c' + bc$

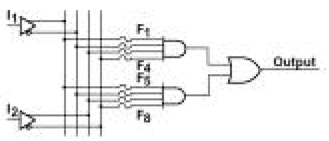


Programmable Array Logic (PALs)

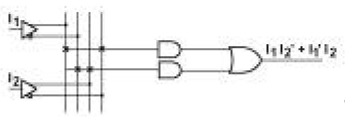
- PAL is a special case of PLA
  - AND array is programmable and OR array is fixed
- PAL is
  - less expensive
  - easier to program

Programmable Array Logic (PALs)

Unprogrammed



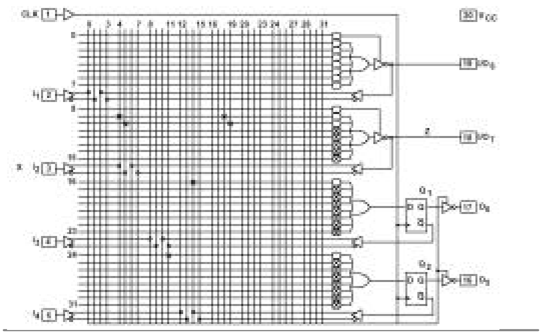
Programmed



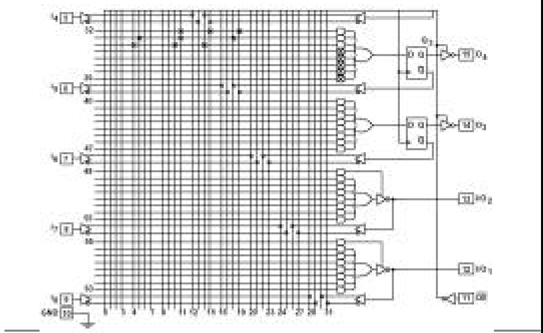
PALs

- Typical PALs have
  - from 10 to 20 inputs
  - from 2 to 10 outputs
  - from 2 to 8 AND gates driving each OR gate
  - often include D flip-flops

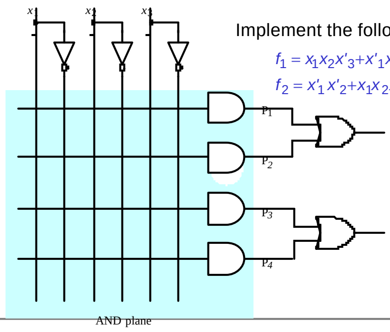
Logic Diagram for 16R4 PAL



Logic Diagram for 16R4 PAL



Using PALs: An Example

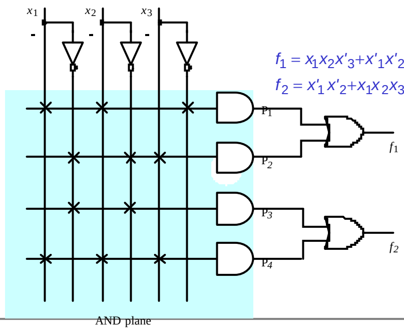


Implement the following:

$f_1 = x_1x_2x'_3+x'_1x'_2\,x_3$

$f_2 = x'_1\,x'_2+x_1x_2x_3$

Using PALs: An Example

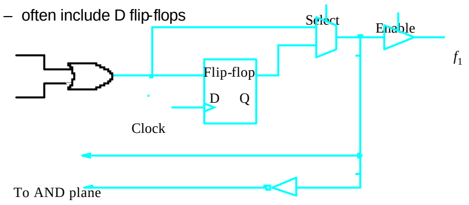


$f_1 = x_1x_2x'_3+x'_1x'_2\,x_3$

$f_2 = x'_1\,x'_2+x_1x_2x_3$

Typical PALs

- Typical PALs have
  - from 10 to 20 inputs
  - from 2 to 10 outputs
  - from 2 to 8 AND gates driving each OR gate
  - often include D flip-flops



MUX output is "fed back" to the AND plane. Why?

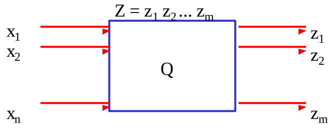
To Do

- Read
  - Textbook chapters 1.5, 3.1, 3.2, 3.3

Sequential Networks

- Have memory (state)
  - Present state depends not only on the current input, but also on all previous inputs (history)
  - Future state depends on the current input and state

$X = x_1\ x_2 \ldots\ x_n$
$Q = Q_1\ Q_2 \ldots\ Q_k$
$Z = z_1\ z_2 \ldots\ z_m$

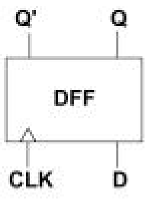


$Z(t) = F(X(t), Q(t))$

$Q(t^+) = G(X(t), Q(t))$

Flip-flops are commonly used as storage devices: D-FF, JK-FF, T-FF

Clocked D Flip-Flop with Rising-edge Trigger



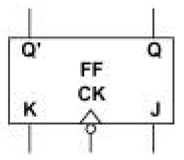
D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

$Q^+ = D$

Next state

The next state in response to the rising edge of the clock is equal to the D input before the rising edge

Clocked JK Flip-Flop

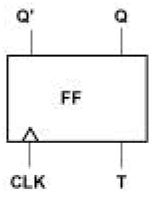


J	K	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Next state $Q^+ = JQ' + K'Q$

JK = 00 => no state change occurs
JK = 10 => the flip-flop is set to 1, independent of the current state
JK = 01 => the flip-flop is always reset to 0
JK = 11 => the flip-flop changes the state $Q^+ = Q'$

Clocked JK Flip-Flop

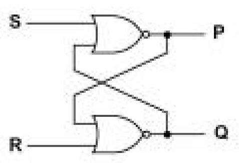


T	Q	Q^+
0	0	0
0	1	1
1	0	1
1	1	0

Next state $Q^+ = QT' + Q'T = Q \oplus T$

T = 1 => the flip-flop changes the state $Q^+ = Q'$
T = 0 => no state change

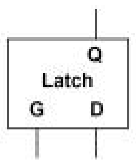
S-R Latch



S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	–
1	1	1	–

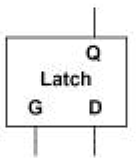
$Q^+ = S + R'Q$

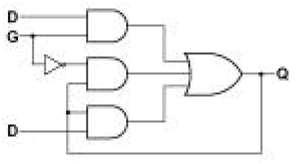
Transparent D Latch



G	D	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Transparent D Latch

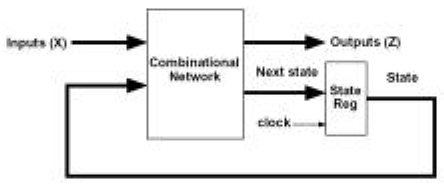




$Q^+ = DG + G'Q + (DQ)$

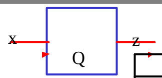
Mealy Sequential Networks

General model of Mealy Sequential Network



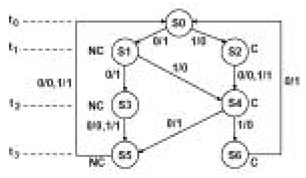
(1) X inputs are changed to a new value
(2) After a delay, the Z outputs and next state appear at the output of CM
(3) The next state is clocked into the state register and the state changes

An Example: 8421 BCD to Excess3 BCD Code Converter



X (inputs)				Z (outputs)			
t3	t2	t1	t0	t3	t2	t1	t0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

State Graph and Table for Code Converter



	NS		Z	
PS	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	–	1	–

State Assignment Rules

I. States which have the same next state (NS) for a given input should be given adjacent assignments (look at the columns of the state table).

II. States which are the next states of the same state should be given adjacent assignments (look at the rows).

III. States which have the same output for a given input should be given adjacent assignments.

I. (1,2) (3,4) (5,6) (in the X=1 column, S_1 and S_2 both have NS S_4;
in the X=0 column, S_3 & S_4 have NS S_5, and S_5 & S_6 have NS S_0)

II. (1,2) (3,4) (5,6) (S_1 & S_2 are NS of S_0; S_3 & S_4 are NS of S_1;
and S_5 & S_6 are NS of S_4)

III. (0,1,4,6) (2,3,5)

Q2 Q3 \ Q1	0	1
00	S0	S1
01		S2
11	S5	S3
10	S6	S4

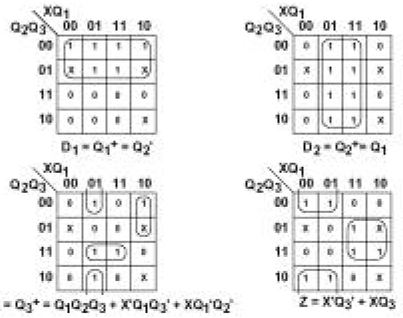
Transition Table

	NS		Z	
PS	X=0	X=1	X=0	X=1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	–	1	–

	Q1+Q2+Q3+		Z	
Q1Q2Q3	X=0	X=1	X=0	X=1
000	100	101	1	0
100	111	110	1	0
101	110	110	0	1
111	011	011	0	1
110	011	010	1	0
011	000	000	0	1
010	000	xxx	1	x
001	xxx	xxx	x	x

S0 = 000, S1 = 100, S2 = 101, S3 = 111, S4 = 110, S5 = 011, S6 = 010

K-maps



$Q_2Q_3 \backslash XQ_1$	00	01	11	10
00	1	1	1	1
01	x	1	1	x
11	0	0	x	0
10	0	0	x	x

$D_1 = Q_1^+ = Q_2'$

$Q_2Q_3 \backslash XQ_1$	00	01	11	10
00	0	1	1	0
01	x	1	1	x
11	0	1	1	0
10	0	1	1	x

$D_2 = Q_2^+ = Q_1$

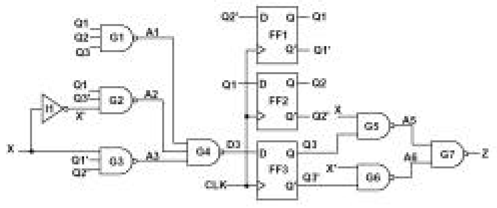
$Q_2Q_3 \backslash XQ_1$	00	01	11	10
00	0	1	0	1
01	x	0	x	x
11	0	1	1	x
10	x	1	x	x

$D_3 = Q_3^+ = Q_1Q_2Q_3 + X'Q_1Q_3' + XQ_1'Q_2'$

$Q_2Q_3 \backslash XQ_1$	00	01	11	10
00	1	1	0	x
01	x	0	1	x
11	0	0	1	1
10	1	1	x	x

$Z = X'Q_3' + XQ_3$

Realization


Q1
Q2
Q3
G1
A1
Q1
Q3'
X'
G2
A2
H
X
Q1'
Q2'
G3
A3
G4
D3
Q2'
D
Q
Q1
FF1
Q'
Q1'
Q1
D
Q
Q2
FF2
Q'
Q2'
D
Q
Q3
FF3
Q'
Q3'
CLK
X
G5
A5
X'
G6
A6
G7
Z