Digital Logic I

CMSC 313
Sections 01, 02

Introduction to Digital Logic

Chapter 3 Objectives

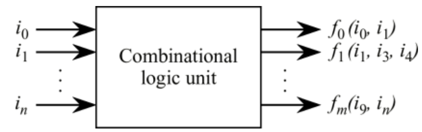
- Understand the relationship between Boolean logic and digital computer circuits.
- Learn how to design simple logic circuits.
- Understand how digital circuits work together to form complex computer systems.

Some Definitions

- Combinational logic: a digital logic circuit in which logical decisions are made based only on combinations of the inputs. e.g. an adder.
- Sequential logic: a circuit in which decisions are made based on combinations of the current inputs as well as the past history of inputs. e.g. a memory unit.
- Finite state machine: a circuit which has an internal state, and whose outputs are functions of both current inputs and its internal state. e.g. a vending machine controller.

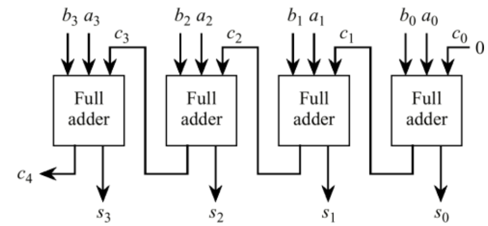
The Combinational Logic Unit

- Translates a set of inputs into a set of outputs according to one or more mapping functions.
- Inputs and outputs for a CLU normally have two distinct (binary) values: high and low, 1 and 0, 0 and 1, or 5 V. and 0 V. for example.
- The outputs of a CLU are strictly functions of the inputs, and the outputs are updated immediately after the inputs change. A set of inputs $i_0 - i_n$ are presented to the CLU, which produces a set of outputs according to mapping functions $f_0 - f_m$.



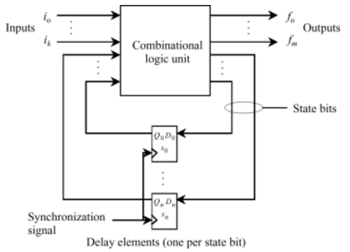
Ripple Carry Adder

- Two binary numbers *A* and *B* are added from right to left, creating a sum and a carry at the outputs of each full adder for each bit position.

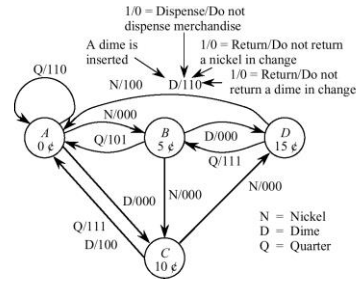


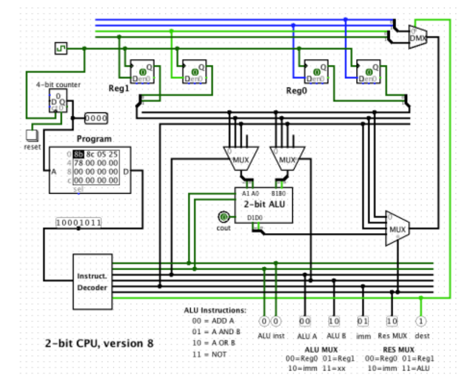
Classical Model of a Finite State Machine

- An FSM is composed of a combinational logic unit and delay elements (called *flip-flops*) in a feedback path, which maintains state information.



Vending Machine State Transition Diagram





3.2 Boolean Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are "true" and "false."
  - In digital systems, these values are "on" and "off," 1 and 0, or "high" and "low."
- Boolean expressions are created by performing operations on Boolean variables.
  - Common Boolean operators include AND, OR, and NOT.

3.2 Boolean Algebra

- A Boolean operator can be completely described using a truth table.
- The truth table for the Boolean operators AND and OR are shown at the right.
- The AND operator is also known as a Boolean product. The OR operator is the Boolean sum.

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

3.2 Boolean Algebra

- The truth table for the Boolean NOT operator is shown at the right.
- The NOT operation is most often designated by a prime mark (X'). It is sometimes indicated by an overbar ($\bar{X}$) or an "elbow" ($\neg X$).

NOT X

X	X′
0	1
1	0

3.2 Boolean Algebra

- A Boolean function has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is also a member of the set {0,1}.

Now you know why the binary numbering system is so handy in digital systems.

3.2 Boolean Algebra

- The truth table for the Boolean function:

$$F(x,y,z) = xz' + y$$

is shown at the right.

- To make evaluation of the Boolean function easier, the truth table contains extra (shaded) columns to hold evaluations of subparts of the function.

$F(x,y,z) = xz' + y$

x	y	z	z'	xz'	xz' + y
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

3.2 Boolean Algebra

- As with common arithmetic, Boolean operations have rules of precedence.
- The NOT operator has highest priority, followed by AND and then OR.
- This is how we chose the (shaded) function subparts in our table.

$F(x,y,z) = xz' + y$

x	y	z	z'	xz'	xz' + y
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

3.2 Boolean Algebra

- Digital computers contain circuits that implement Boolean functions.
- The simpler that we can make a Boolean function, the smaller the circuit that will result.
  - Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
- With this in mind, we always want to reduce our Boolean functions to their simplest form.
- There are a number of Boolean identities that help us to do this.

16

3.2 Boolean Algebra

- Most Boolean identities have an AND (product) form as well as an OR (sum) form. We give our identities using both forms. Our first group is rather intuitive:

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$xx' = 0$	$x + x' = 1$

17

3.2 Boolean Algebra

- Our second group of Boolean identities should be familiar to you from your study of algebra:

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

18

3.2 Boolean Algebra

- Our last group of Boolean identities are perhaps the most useful.
- If you have studied set theory or formal logic, these laws are also familiar to you.

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$(xy)' = x' + y'$	$(x+y)' = x'y'$
Double Complement Law	$(x)'' = x$	

3.2 Boolean Algebra

- We can use Boolean identities to simplify:

$$F(x,y,z) = xy + x'z + yz$$

$$\begin{aligned}
F(x,y,z) &= xy + x'z + yz \\
&= xy + x'z + yz(1) && \text{(Identity)} \\
&= xy + x'z + yz(x + x') && \text{(Inverse)} \\
&= xy + x'z + (yz)x + (yz)x' && \text{(Distributive)} \\
&= xy + x'z + x(yz) + x'(zy) && \text{(Commutative)} \\
&= xy + x'z + (xy)z + (x'z)y && \text{(Associative twice)} \\
&= xy + (xy)z + x'z + (x'z)y && \text{(Commutative)} \\
&= xy(1 + z) + x'z(1 + y) && \text{(Distributive)} \\
&= xy(1) + x'z(1) && \text{(Null)} \\
&= xy + x'z && \text{(Identity)}
\end{aligned}$$

3.2 Boolean Algebra

- Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly.
- DeMorgan's law provides an easy way of finding the complement of a Boolean function.
- Recall DeMorgan's law states:

$(xy)' = x' + y'$ and $(x + y)' = x'y'$

3.2 Boolean Algebra

- DeMorgan's law can be extended to any number of variables.
- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs.
- Thus, we find the the complement of:

$$F(x,y,z) = (xy)+(x'y)+(xz')$$

is:

$$\begin{aligned} F'(x,y,z) &= ((xy)+(x'y)+(xz'))' \\ &= (xy)'(x'y)'(xz')' \\ &= (x'+y')(x+y')(x'+z) \end{aligned}$$

3.2 Boolean Algebra

- Through our exercises in simplifying Boolean expressions, we see that there are numerous ways of stating the same Boolean expression.
  - These "synonymous" forms are *logically equivalent*.
  - Logically equivalent expressions have identical truth tables.
- In order to eliminate as much confusion as possible, designers express Boolean functions in *standardized* or *canonical* form.

3.2 Boolean Algebra

- There are two canonical forms for Boolean expressions: sum-of-products and product-of-sums.
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  - For example: $F(x,y,z) = xy + xz + yz$
- In the product-of-sums form, ORed variables are ANDed together:
  - For example: $F(x,y,z) = (x+y)(x+z)(y+z)$

3.2 Boolean Algebra

- It is easy to convert a function to sum-of-products form using its truth table.
- We are interested in the values of the variables that make the function true (=1).
- Using the truth table, we list the values of the variables that result in a true function value.
- Each group of variables is then ORed together.

$F(x,y,z) = xz' + y$

x	y	z	xz'+ y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

25

3.2 Boolean Algebra

- The sum-of-products form for our function is:

$$F(x,y,z) = (x'yz') + (x'yz) + (xy'z') + (xyz') + (xyz)$$

We note that this function is not in simplest terms. Our aim is only to rewrite our function in canonical sum-of-products form.

$F(x,y,z) = xz' + y$

x	y	z	xz'+ y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

26

3.3 Logic Gates

- We have looked at Boolean functions in abstract terms.
- In this section, we see that Boolean functions are implemented in digital computer circuits called gates.
- A gate is an electronic device that produces a result based on two or more input values.
  - In reality, gates consist of one to six transistors, but digital designers think of them as a single unit.
  - Integrated circuits contain collections of gates suited to a particular purpose.

27

3.3 Logic Gates

- The three simplest gates are the AND, OR, and NOT gates.

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	X'
0	1
1	0

- They correspond directly to their respective Boolean operations, as you can see by their truth tables.

3.3 Logic Gates

- Another very useful gate is the exclusive OR (XOR) gate.
- The output of the XOR operation is true only when the values of the inputs differ.

X XOR Y

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Note the special symbol ⊕ for the XOR operation.

3.3 Logic Gates

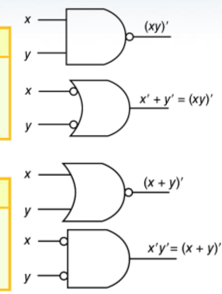
- NAND and NOR are two very important gates. Their symbols and truth tables are shown at the right.

X NAND Y

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

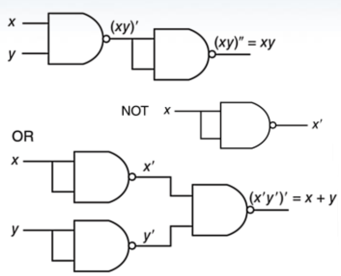
X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0



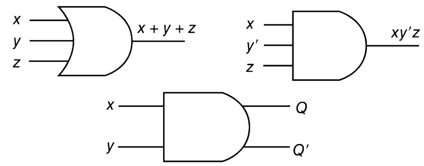
3.3 Logic Gates

- NAND and NOR are known as *universal gates* because they are inexpensive to manufacture and any Boolean function can be constructed using only NAND or only NOR gates.



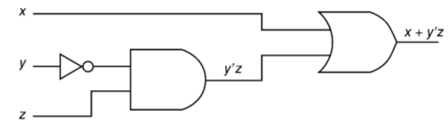
3.3 Logic Gates

- Gates can have multiple inputs and more than one output.
  - A second output can be provided for the complement of the operation.
  - We'll see more of this later.



3.4 Digital Components

- The main thing to remember is that combinations of gates implement Boolean functions.
- The circuit below implements the Boolean function `F(x,y,z) = x + y'z`:

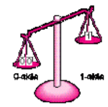


We simplify our Boolean expressions so that we can create simpler circuits.

The Sum-of-Products (SOP) Form

Truth Table for The Majority Function

Minterm Index	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



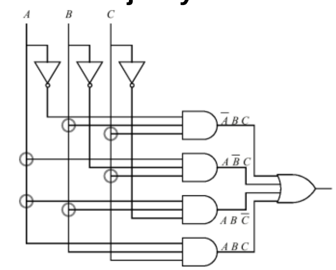
A balance tips to the left or right depending on whether there are more 0's or 1's.

- Transform the function into a two-level AND-OR equation
- Implement the function with an arrangement of logic gates from the set {AND, OR, NOT}
- M is true when A=0, B=1, and C=1, or when A=1, B=0, and C=1, and so on for the remaining cases.
- Represent logic equations by using the sum-of-products (SOP) form

The SOP Form of the Majority Gate

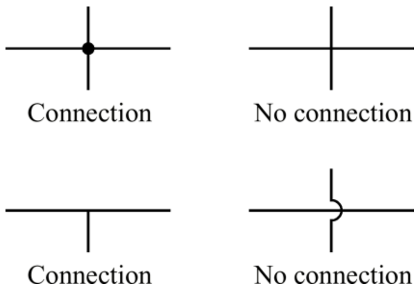
- The SOP form for the 3-input majority gate is:
- $M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC = m3 + m5 + m6 + m7 = \Sigma(3, 5, 6, 7)$
- Each of the 2^n terms are called minterms, running from 0 to $2^n - 1$
- Note the relationship between minterm number and boolean value.
- Discuss: common-sense interpretation of equation.

A 2-Level AND-OR Circuit Implements the Majority Function



The encircled "T" intersections are electrically common (see next slide).

Notation Used at Circuit Intersections



Sum of Products (a.k.a. disjunctive normal form)

- OR (i.e., sum) together rows with output 1
- AND (i.e., product) of variables represents each row
  e.g., in row 3 when $x_1 = 0$ AND $x_2 = 1$ AND $x_3 = 1$
  or when $\overline{x_1} \cdot x_2 \cdot x_3 = 1$
- MAJ3$(x_1, x_2, x_3) = \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 = \sum m(3,5,6,7)$

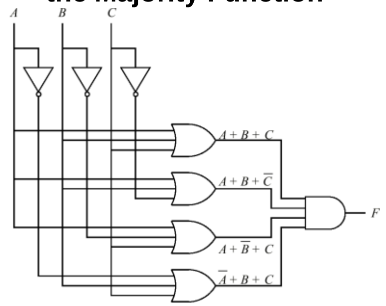
	x_1	x_2	x_3	MAJ3
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Product of Sums (a.k.a. conjunctive normal form)

- AND (i.e., product) of rows with output 0
- OR (i.e., sum) of variables represents negation of each row
  e.g., NOT in row 2 when $x_1 = 1$ OR $x_2 = 0$ OR $x_3 = 1$
  or when $x_1 + \overline{x_2} + x_3 = 1$
- MAJ3$(x_1, x_2, x_3) = (x_1+x_2+x_3)(x_1+x_2+\overline{x_3})(x_1+\overline{x_2}+x_3)(\overline{x_1}+x_2+x_3)$
  $= \prod M(0,1,2,4)$

	x_1	x_2	x_3	MAJ3
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

A 2-Level OR-AND Circuit Implements the Majority Function



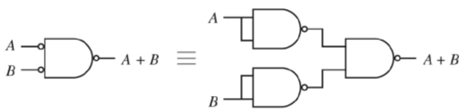
Equivalences

- Every Boolean function can be written as a truth table
- Every truth table can be written as a Boolean formula (SOP or POS)
- Every Boolean formula can be converted into a combinational circuit
- Every combinational circuit is a Boolean function
- Later you might learn other equivalencies:
  finite automata $\equiv$ regular expressions
  computable functions $\equiv$ programs

Universality

- Every Boolean function can be written as a Boolean formula using AND, OR and NOT operators.
- Every Boolean function can be implemented as a combinational circuit using AND, OR and NOT gates.
- Since AND, OR and NOT gates can be constructed from NAND gates, NAND gates are universal.

NAND Gates Can Implement AND and OR Gates

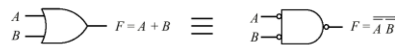


Inverted inputs to a NAND gate are implemented with NAND gates.

DeMorgan's Theorem

$A\ B$	$\overline{AB} = \overline{A} + \overline{B}$		$\overline{A+B} = \overline{A}\,\overline{B}$	
0 0	1	1	1	1
0 1	1	1	0	0
1 0	1	1	0	0
1 1	0	0	0	0

DeMorgan's theorem: $A + B = \overline{\overline{A + B}} = \overline{\overline{A}\,\overline{B}}$



Discuss: Applying DeMorgan's theorem by "pushing the bubbles," and "bubble tricks."