EECS-140/141 Introduction to Digital Logic Design
Lecture 2: Combinational Logic Basics

I. INTRODUCTION

I.A Major Classes of Digital Logic Circuits

I.A.1 Combinational Logic

In this class, outputs are *combinations* of input values at any given time (the present).

Past inputs do not

I.A.2 Sequential Logic

I.B Switches, Variables, and Functions

I.B.1 Simple Switch Circuit

To begin with, think of a simple flashlight ckt, which represents the *simplest binary* ckt.

Position of switch *controls* whether light is off or on.

Switch Position	*Light*	*Current*

I.B.2 Variables and Function

In digital logic terms, switch position is known as an *input variable* and the light status (on/off) can be considered as an *output variable* or a *function* of the input variable(s).

Now assign symbols and numerical values to *variables* and *function*:

Switch (x) *Light* ($y = I(x)$)

We have just defined the simplest logic function: the *identity* function.

It is not a very interesting function, but (as with all functions) it can be summarized by a *Truth Table.*

II. THREE FUNDAMENTAL LOGIC FUNCTIONS

II.A Logical AND Function

II.A.1 Word Example

If you are *enrolled* in this class (at the end of the semester) and get a *passing* grade, then you will get *credit* for the course.

Define variables and assign values:

II.A.2 Circuit Model

Note similarity to a *series* connection of switches:

II.A.3 Truth Table for Logical AND

e	p	$c = e \cdot p$
0	0	
0	1	
1	0	
1	1	

II.B Logical OR Function

II.B.1 Word Example

If you *drop* this class or *withdraw* or your payment *bounces*, you will be *disenrolled*.

II.B.2 Circuit Model (Parallel Switches)

II.B.3 Truth Table

d	w	b	$s = d + w + b$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

II.C Logical NOT (or Complement or Inversion)

II.C.1 Word Example

Clearly, if you *are* enrolled, then you are *not* disenrolled.

II.C.2 Circuit Model

II.C.3 Truth Table (TT)

III. LOGIC NETWORKS AND EXPRESSIONS

III.A Three Fundamental Logic Gates

III.B Logic Networks

Network: Interconnection of Elements (Gates)

III.B.1 Example

Result: 4-input, 1-output logic network.

III.B.2 Analysis of Logic Networks

Brute-force method: Enumeration: Consider all possible input combinations, trace each through the network to get the output value. Summarize results in TT.

d	*w*	*b*	*p*	*c*
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

III.C Logic Expressions

For a given network (or TT), we can also write *a* corresponding logic expression.

III.C.1 Example: Read directly from network above

III.C.2 Expressions are NOT unique!!

From above truth table, note that $c = 1$ only when:

Equivalent way to say this:

Since they have the *same* TT, this expression is equivalent to our original expression:

Note that the 2nd expression results in a *different* (but equivalent) logic network:

So, logic networks are not unique, either! This highlights the importance of *design*.

III.D Some Questions

1. What other useful rules are there besides:
2. How can I find equivalent expression/networks for a given TT?
3. Of the many expression/networks for a given TT, which is the *simplest*?

We will answer questions 1 and 2 in the next section; we will consider question 3 later.

IV. BOOLEAN ALGEBRA

IV.A Intro/Motivation

We need something to help us answer questions 1 and 2 above. Boolean Algebra is the mathematical tool that we need.

Algebra:

Boolean Algebra (George Boole: 1849!):

Boolean Algebra (BA) is *not* like "normal" (arithmetic) algebra, even though *some* properties/rules are similar.

IV.B Axioms of Boolean Algebra

These are the "givens" or assumptions that are true by definition.

We list them in pairs:

1a)

1b)

2a)

2b)

3a)

3b)

4a)

4b)

Note that the "a" parts of 1-3 define the AND function.

Note that the "b" parts of 1-3 define the OR function.

Note that 4a and 4b define the NOT function.

These are the basic *operations* of the algebra.

IV.C Single Variable Theorems

Theorem:

- These deal with a single binary variable (x)
- Again list them in pairs
- Continue numbering (start with 5).

5a)

5b)

6a)

6b)

7a)

7b)

8a)

8b)

9)

So far, we can easily prove each by considering the 2 possible values for *x* (this is all of the possible values), and then applying the axioms. In general, when we consider all *combinations* of all input variables, this method is known as *perfect induction* or the TT method.

Question: which of 5-7 are true in "regular" algebra (where + means add and · means multiply)?

IV.D Some Simplifications for Complicated Expressions

IV.D.1 Precedence

This is the order in which operations are performed.

For BA, the precedence is:

1. parentheses
2. NOT
3. AND
4. OR

IV.D.2 Shorthand

As with arithmetic multiply, $a \cdot b$ can be shortened to *ab IF* there is no confusion.

Also:

So, the above can be written as:

but not as:

Important Note:

a	b	$\overline{a \cdot b}$	$\bar{a} \cdot \bar{b}$
0	0	1	
0	1	1	
1	0	1	
1	1	0	

IV.E Two and Three Variable Theorems (Properties)

These involve multiple variables.

10: Commutative

a)

b)

11: Associative

a)

Proof by TT

x	y	z	$y \cdot z$	$x \cdot (y \cdot z)$	$x \cdot y$	$(x \cdot y) \cdot z$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

b)

You prove with TT as above.

12: Distributive

a)

You prove with TT as above.

b)

You prove as Problem 2.1 on Assignment 2 *using algebraic manipulation.* (We will do an example with 13a).

13: Absorption

a)

Prove with *algebraic manipulation.*

Can start with left-hand side (LHS) and manipulate to get RHS or vice versa.

Here, start with LHS:

Use a similar process to prove 12b on Assignment 2. You *may* use 13a as a step.

Here are the first few steps:

b)

14: Combining

a)

b)

You will prove in Problem 2.2 of Assignment 2: you *must* use algebraic manipulation!!

15: DeMorgan's Theorem

a)

Proof via TT (Figure 2.11 in text).

b)

You prove via TT.

16: (no name)

a)

b)

17: Consensus

a)

Proof is Problem 2.3 of Assignment 2.

b)

IV.F Venn Diagrams

IV.F.1 Introduction

Venn diagrams are a graphical means of representing sets.

The "universe" is represented by a box. For BA, the universe is the set {0,1}.

A subset of the universe is represented by a labeled closed contour (e.g. a circle). Inside the contour, the labeled value=1; outside, the value=0.

Shading: the expression being represented has value=1. No shading means the expression being represented has value=0.

Examples:

IV.F.2 BA Proofs

Similar to perfect induction, we can use Venn diagrams to prove identities.

Example: DeMorgan (15a)

You try this method to prove 15b.

V. SYNTHESIS WITH AND/OR/NOT

V.A Preliminaries

V.A.1 Synthesis

Process of finding a logic network implementation of a given function, as specified by a TT (or logic expression).

V.A.2 Terminology

At the risk of confusing BA with arithmetic algebra, we will refer to "sums" and "products":

"sum" =

"product" =

We will use these terms, even though *logical* + and · are quite different from arithmetic + and ·.

V.A.3 Cost

We will use a simple measure of the "cost" or complexity of a logic circuit:

except inverting input variables with NOT gates is "free" (zero cost).

V.B Sum of Products Synthesis

V.B.1 Intro

This is one of the easiest synthesis methods: Find all input *product* (AND) combinations that produce a 1 output, then OR (sum) them all together.

V.B.2 Example

Consider the following 2-input TT:

a	b	f
0	0	1
0	1	0
1	0	1
1	1	1

Here, *one* condition that forces $f = 1$ is if $a = 0$ and $b = 0$.

Equivalently, if $\bar{a} = 1$ and $\bar{b} = 1$.

Equivalently, if $\bar{a} \cdot \bar{b} = 1$.

Another condition is $a = 1$ and $b = 0$ or $a \cdot \bar{b} = 1$.

Final condition is $a \cdot b = 1$.

Now OR them together:

Implement this expression directly as a logic network in Sum of Products (SoP) form:

V.B.3 Minterms

Note that each product term in f above contains *every* input variable exactly once, in either "native" or inverted form. Such a product term is called a *minterm*.

Note also that each minterm corresponds to a *row* of the TT, which we can number in counting order (starting with 0).

As shorthand, we say $\bar{a} \cdot \bar{b}$ is m_0, $\bar{a} \cdot b$ is m_1, etc. Using this shorthand, an equivalent expression for the function f in the previous example is:

Note that minterm *number* is *very* dependent on the *order* of the input variables, so we can *emphasize* that by writing:

V.B.4 Canonical SoP Form

A SoP logic network/expression in which *every* product term is a minterm is called the *canonical* SoP form (CSoP form).

V.B.4.a Expression for Cost of Canonical SoP Form

Let i be the number of inputs for a function f and let m be the number of $f = 1$ minterms in the CSoP form for f. (Note that m is also the number of 1's in the output column of the TT).

Then the CSoP implementation will have:

Gates:

and

Inputs:

for a total "cost" of:

V.B.5 Simpler SoP Implementations

Look again at the example TT:

a	b	f
0	0	1
0	1	0
1	0	1
1	1	1

Note that $f = 1$ whenever $a = 1$ *or* $a = 0$ and $b = 0$.

So, $f =$

Could also note that $f = 1$ whenever $b = 0$ *or* $a = 1$ and $b = 1$.

But the simplest implementation is:

Note: could do the simplifications with BA:

Systematic method for simplification will come later.

V.C Product of Sums Form

V.C.1 Background

If there is a SoP synthesis, is there also a Product of Sums (PoS) synthesis? Yes!

Consider $\bar{f}$ of the previous example:

a	b	f
0	0	1
0	1	0
1	0	1
1	1	1

V.C.2 Maxterms

For every minterm, there is a corresponding Maxterm:

$$M_k = \overline{m_k}$$

Above:

Note: maxterms are *i*-term sums (*i* is number of inputs)

V.C.3 Example

x	*y*	*z*
0	0	0
0	1	1
1	0	1
1	1	0

V.D Design Examples

V.D.1 3-way Light Control

Read/study section 2.8.1 in text.

V.D.2 Multiplexer (Mux)

We often want to *select* one of several *data* inputs to "pass through" to a single output. Such a device is a Multiplexer (or Mux).

Simplest case:

s	x_1	x_0	y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

CSoP:

Compact TT:

Common logic symbol for mux:

V.D.3 AND/OR Circuit

It might be useful to have a ckt that can function as either an AND gate or an OR gate for 2 data inputs under control of a third *select* input *s*:

s	x_1	x_2	y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

CSoP for AND/OR ckt:

CPoS:

Both have cost: