Logic Design (Part 2)
Combinational Logic Circuits (Chapter 3 + Notes)

1

Next

- Download Set1 of Cedar Logic or LogiSim examples
  - From lectures webpage
- Designing combinational logic circuits
  - Cedar Logic or LogiSim
- Intro to Combinational logic devices
  - Building complex devices using the basic gates
  - Adders, Multiplexers, Decoders,…..

2

Digital Logic Design – Introduction

- MOS transistors used as switches to implement basic logic gates (boolean operations)
  - (NAND, NOT, AND, OR,..).
- Boolean logic functions operate on boolean variables
  - expressed with AND, OR, and NOT
  - (A AND B) OR (NOT A AND C)
    - AND, OR, NOT notation replaced by . + ‘
    - (A.B) + (A’ . C)

- Boolean function represented as:
  - Truth table
  - logic circuit

3

Digital Logic Circuits

- We saw how we can build the simple logic gates using transistors
- Can build any boolean function using these gates
- Use these gates as building blocks to build more complex combinational circuits
  - Multiplier, Multiplexer, Decoder, …..
  - …any boolean function
- Develop a theoretically sound approach to designing boolean functions and circuits….Boolean Algebra

4

Definition: Combinational and Sequential Logic Circuits

- A circuit is a collection of devices that are physically connected by wires
  - Combinational circuit
  - Sequential circuit
- In Combinational circuit the input determines output
- In sequential circuit, the input and the previous 'state' (previous values) determine output and next 'state'
  - Need to 'remember' previous value – need memory device
  - Need circuit to implement concept of storage
- Start with design of Combinational Logic circuits

5

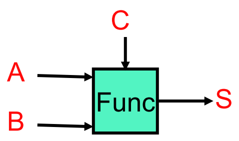
Boolean Function..Recall from Discrete Math!

- Output(s) is a function on input boolean variables
- *z = f(x,y,…)*
- *x,y are boolean variables (0 or 1)*
- *z=f(x,y) is a boolean output*
- The "operators" used are any of the boolean logic operators
  - AND, OR, NOT, NAND, etc.
- If integers are represented using binary notation, then all functions over integers are boolean functions
- How do we represent boolean functions?

6

Boolean Functions

- A function can be thought of as a mapping from inputs to outputs.
  - Think of a black box with n binary inputs and 1 binary output
- We can express the action of this function in terms of a truth table which says what the output should be for every input pattern.
- This function implements a binary adder!



Block Diagram

Truth table
(describes behavior)

A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

7

Boolean Algebra

- George *Boole* – Famous Mathematician/Logician
  - *Boolean* Algebra – branch of Algebra, where variables can only have values of true (1) or false (0)
  - Boolean operators: AND(.), OR(+), NOT(~ or !)
- With Boolean Algebra:
  - We create “functions” using boolean variables and operators
  - Any logical function can be expressed in terms of the three elementary operations: AND, OR and NOT
  - Boolean functions can be rearranged and sometimes simplified by applying algebraic identities
  - DeMorgan’s laws: allow conversion from AND to OR (using NOT)
- Big idea – you can write a logical function as a boolean algebraic expression and then use various identities to rewrite that function in an equivalent (usually simpler) form.

8

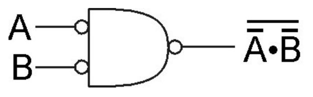
DeMorgan's Law: Converting AND to OR with help from NOT

- NOT(A AND B) = (NOT A) OR (NOT B);
- NOT(A OR B) = (NOT A) AND (NOT B);
- In C syntax: ~(A&B) = ~A|~B   ~(A|B) = ~A&~B

A	B	~A	~B	~A&~B	(A&B)	~A|~B	(A|B)	~(A|B)	~(A&B)
0	0	1	1	1	0	1	0	1	1
0	1	1	0	0	0	1	1	0	1
1	0	0	1	0	0	1	1	0	1
1	1	0	0	0	1	0	1	0	0

DeMorgan's Law

- Converting AND to OR (with some help from NOT)
- Consider the following gate:



From DeMorgan's laws:
NOT ((NOT A) AND (NOT B))=
NOT (NOT A) OR NOT(NOT B)=
A OR B

A	B	$\bar{A}$	$\bar{B}$	$\bar{A}\cdot\bar{B}$	$\overline{\bar{A}\cdot\bar{B}}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Same as A+B

To convert AND to OR (or vice versa), invert inputs and output.

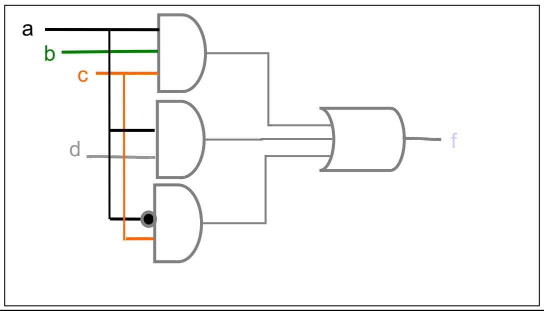
Representation of Boolean Logic Functions

✧A logic function can be represented as

1. a truth table or
2. a logic expression or
3. a logic circuit

✧Example

$$f = a.(b.c + d) + \bar{a}.c = a.b.c + a.d + \bar{a}.c$$



a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

11

Completeness: Very Important Concept

- It can be shown that any truth table (i.e. any binary function of binary variables) can be reduced to combinations of the AND & NOT functions, or of the OR & NOT functions.
  - This result extends also to functions of more than two variables
- In fact, it turns out to be convenient to use a basic set of three logic gates:
  - AND, OR & NOT or NAND, NOR & NOT
  - In fact, can implement all logic functions using just NAND!
- Question to ask: How do we design a good circuit ?

12

Truth Tables to Boolean Function/Circuits

- A truth table can be mapped to a Boolean function
  - In **Disjunctive Normal Form (DNF)** – an OR of AND terms
    - Recall from Discrete 1 (CS1311)
- Each row in the truth table corresponds to a conjunction of literals (i.e., Boolean variables) and is called *minterm*
  - Literal is Boolean variable A or its complement A'
- To derive the Boolean function F:
  - Examine each row where the output = 1
    - Include this conjunctive term as an AND of the literals
  - F = OR of the included terms (minterms)
    - Also called sum of products (OR of minterms)

13

Truth Tables to Boolean Function

- Input variables: A,B,C (Notation: NOT A written as A')
- Output = F
- First examine rows where F=1
- F=1 when A=0 B=1 C=0
  - bool func is (A' AND B AND C')
- F=1 when A=1 B=0 C=0
  - bool func is (A AND B' AND C')
- F=1 when A=1 B=1 C=0
  - bool func is (A AND B AND C')
- F=0 on all other input combinations
- Write F as OR of above minterms
- F= (A'.B.C') OR (A.B'.C') OR (A.B.C')

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	**1**
0	1	1	0
1	0	0	**1**
1	0	1	0
1	1	0	**1**
1	1	1	0

14

Boolean functions, Circuits & Boolean Algebra!

▪Power of abstraction….To build boolean functions, you can work with basic gates..no need to go down to the transistor level !!

▪ what is the theory behind boolean logic design ?

- Boolean Algebra
  - DeMorgan's Law: Convert AND&NOT to OR&NOT
    - (NOT A) AND (NOT B) = NOT (A OR B) (i.e., A NOR B)
    - (NOT A) OR (NOT B) = NOT (A AND B) (i.e, A NAND B)
  - Disjunctive Normal Form (DNF), etc.
- Question: How can we design an "efficient" circuit to implement a Boolean function ?
  - "efficient" = minimum number of logic gates
- Example:
  - Given F= A'BC' + A'B'C + ABC' + AB'C : 4 AND gates & OR gate
  - From Bool Alg., this is equivalent to F = BC' + B'C: 2 AND gates & 1 OR gate

15

Boolean Algebra and Combinational Logic Design

- Boolean Logic is a Boolean Algebra
  - laws of Boolean Algebra can be applied
- How do we design "efficient" circuits ? Is there a methodology we can follow ?
  - Boolean algebra (Karnaugh Maps)
  - Again, recall from CS1311 Discrete 1 !

- Reading Assignment: Review Boolean algebra and Karnaugh Maps concepts and application to digital logic design
  - Notes posted on website

16

How to design combinational circuit

- Analyze the problem
  - Determine inputs and outputs (they must be binary)
- Determine boolean variables
  - inputs x1,x2,…
  - Outputs y1, y2,…
- Derive truth table
  - Value of each yi for each combination of inputs x1,x2,…
- For simple circuit, find DNF from truth table
- To find 'optimal' (minimum size) 2-level circuit, derive Karnaugh map and find terms

17

Time to test your circuit design&analysis skills…

Analyze Circuits to derive Boolean functions

Design circuits from truth table/function

- In Cedar Logic: Download and open Set1.cdl in CedarLogic
  - Has number of 'pages' of circuits
- In Logisim: Download Set1.zip into folder, it has several Logisim circuits
  - Labelled Set1-Page1.circ, Set1-Page2, etc.
  - Open in Logisim
- Work through the 6 pages of circuits

18

Transistor Circuits…Circuits on Page 1,2,3,4….

- Open in Cedar Logic – view Page 1 circuit
- *UsefulTip (for CedarLogic): Hit space key to center and maximize the circuit in the cedar logic window*
- Circuit on Page 1 is NAND gate (from transistors)
  - Inputs: A,B . can be set to 1 (red) or 0 (black) by clicking on them
  - Output C=1 if at least one of upper two (switches) close
    - This happens when A=0 or B=0 since it is a P-type transistor
  - Output C=0 when A=1 and B=1 since there is a path from C to ground (0) and N type transistor
- Transistors at bottom of page
  - N-type: if input is 1 (red) then transistor close (i.e., short circuit)
  - P-type: if input is 0 (black) then transistor closed (i.e, short circuit)

19

Questions…Circuit on Set1-Page2, Page3, Page4

- Page 2: What gate/function is being implemented?
  - Can you determine this by analyzing the circuit *(instead of deriving truth table)*
  - Note serial structure in upper two transistors = both must close for a path from Power to output, i.e., for output to be = 1
  - Note parallel structure in lower two transistors = at least one must be closed for a path from Ground (0) to output, i.e., for output =0
- Page 3: combination of Page 2 circuit whose output goes to another circuit
- Page 4: What function ? Derive truth table with inputs A,B and 'outputs' C,D, F
  - Is this a standard function ?

20

Circuits with logic gates…Set1 - Pages 5,6,7,8

- Page 5: example using logic gates
- Input specified using the switches
- Output goes to LED (red=1, black=0)
- To determine the function being implemented by a circuit you can "trace" back from output to inputs

21

Questons: Pages 6, 7, 8:

- Page 6: what Boolean functions are being implemented by circuits (a) and (b) ?
- Page 7: Draw circuit to implement the function:
  - F= (A AND (B XOR C) OR (NOT C))
  - Draw by hand first, then implement in simulator
- Page8: Design circuit for truth table shown here
  - Derive function and draw by hand first
    - Implement in simulator

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

22

Logic control for overhead light

- Design logic for light switch in a room based on the position of two switches – switches are at two different entrances to the room and either switch should be able to change the state (on/off) of the light independently.
  - If both switches are in the "down" position (represented by 0) the light must be off (represented by 0)
  - No matter what position the switches are in or the current state of the light, flipping either switch must change the state of the light.

  - Design gate level circuit
  - Complete truth table first

23

Recall our Goal….

- Design a machine that translates from natural language to electrons running around to solve the problem

  - We now have a device that controls how electrons run around
- Next: we want to build a computer
  - First step: Design a collection of logic devices that implement important functions that will be needed to build our computer

- S/W Analogy: When you write your software, you are using a collection of concepts, tools, IDEs and libraries
  - Each has been built, and tested, for you
  - All you have to do is combine them!

24

Next: Combinational Logic Devices

- We saw how we can build the simple logic gates using transistors
- Can build any boolean function using these gates
- Use these gates as building blocks to build more complex combinational circuits
  - Decoder: based on value of n-bit input control signal, select one of 2^N outputs
  - Multiplexer: based on value of N-bit input control signal, select one of 2^N inputs.
  - Adder: add two binary numbers
  - …any boolean function
- SW Analogy: We are building a library of functions
  - To design your solution, you can use any device in the library!

25