EE224 Short Notes

Aditya Byju

Course Professor: Prof. Siddharth Tallur
Ref: Prof's slides
Relax, it's only $1s$ and $0s$

Digital Systems

September 2021

Contents

Introduction

- Logic is the study of valid reasoning. It tries to establish criteria to decide whether some piece of reasoning is valid or invalid.
- A piece of reasoning is valid if the statements that are claimed to follow from previous ones do indeed follow from those. Otherwise, the reasoning is said to be invalid.
- Logic is a system based on propositions. A proposition is a statement that is either true or false (not both).
- In Propositional Logic (a.k.a Propositional Calculus or Sentential Logic), the objects are called propositions. We usually denote a proposition by a letter: p, q, $r, \ldots$
- The value of a proposition is called its truth value. Opinions, interrogative, and imperative are not propositions.
- Operators/Connectives are used to create a compound proposition from two or more propositions:
  - negation (denoted by $\neg$ or ! or $\sim$)
  - AND or logical conjunction (denoted by $\wedge$ or $\cdot$)
  - OR or logical disjunction (denoted by $\vee$ or $+$)
  - XOR or exclusive OR (denoted by $\oplus$)
  - implication (denoted by $\Rightarrow$ or $\rightarrow$)
  - biconditional (denoted by $\Leftrightarrow$ or $\leftrightarrow$)
- The implication of $p \rightarrow q$ can also be read as:
  - if p than q
  - p implies q
  - if p, q
  - p only if q
  - q if p
  - q when p
  - q whenever p
  - q follows from p
  - p is a sufficient condition for q (p is sufficient for q)
  - q is a necessary condition for p (q is necessary for p)
- Truth table of some common operators:

p	q	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	0
1	1	1	1	0	1	1

- Positional number systems are represented by a positive radix. A number with radix r is represented by a string of digits:

$$A_{n-1}A_{n-2}\dots A_1A_0.A_{-1}A_{-2}\dots A_{-m+1}A_{-m}$$

in which $0 \geq A_i < r$ and . is the radix point.

- Number systems:

		General	Decimal	Binary
Radix (Base)		r	10	2
Digits		0 => r - 1	0 => 9	0 => 1
Powers of Radix	0	r^0	1	1
	1	r^1	10	2
	2	r^2	100	4
	3	r^3	1000	8
	4	r^4	10,000	16
	5	r^5	100,000	32
	-1	r^{-1}	0.1	0.5
	-2	r^{-2}	0.01	0.25
	-3	r^{-3}	0.001	0.125
	-4	r^{-4}	0.0001	0.0625
	-5	r^{-5}	0.00001	0.03125

- Special powers of 2:
  - 2^{10} (1024) is Kilo, denoted by "K"
  - 2^{20} (1048576) is Mega, denoted by "M"
  - 2^{30} (1073741824) is Giga, denoted by "G"
- Difficulties with signed integers:
  - sign and magnitude bits should be differently treated in arithmetic operations
  - addition and subtraction require different logic circuits
  - overflow is difficult to detect
  - "zero" has two representations
- In 1's complement to form a negative number, complement each bit in the given number. In 2's complement to form a negative number, start with the given number, subtract one, and then complement each bit, or first complement each bit, and then add 1
- Positive and negative numbers of equal magnitudes are complements of each other
- **Overflow rule:** if two numbers with the same sign bit (both positive or both negative) are added, the overflow occurs if and only if the result has the opposite sign
- General format of floating point numbers (IEEE 754 floating point standard):

$$\pm 1.bbbb_{two} \times 2^{eeee} \quad or \quad (-1)^S \times (1+F) \times 2^E$$

where S = sign, 0 for positive, 1 for negative, F = fraction (or mantissa) as a binary integer, $1+F$ is called significand, E = exponent as a binary integer, positive or negative (two's complement)

Logic functions

- Truth tables are unique, expressions and logic diagrams are not
- Output column of truth table has length 2^n for n input variables. It can be arranged in 2^{2^n} ways for n variables
- Switching devices:
  - electromechanical relays (1940s)
  - vacuum tubes (1950s)
  - bipolar transistors (1960 -1980)
  - field effect transistors (1980 -)
  - integrated circuits (1970 -)
- Enabling function permits an input signal to pass through an output
- Disabling function blocks an input signal from passing through to an output, replacing it with a fixed value
- Decoding function converts n-bit input to m-bit output (given $n \geq m \geq 2^n$). Circuits that perform decoding are called decoders
- Encoding function does the opposite of what the decoding function does. It converts m-bit input to n-bit output. Circuits that perform encoding are called encoders
- Selection function selects a single input from a set of inputs. Circuits that perform selecting are called multiplexers
- A set of elements is any collection of objects having common properties
- A binary operator $*$ defined on a set S of elements is a rule that assigns each pair from S to a unique pair from S
- An Axiom or Postulate is a self-evident or universally recognized truth
- Postulates:
  - commutative law: an operator $*$ on S is commutative if:
    $$a * b = b * a, \quad \forall a, b \in S$$
  - associative law: an operator $*$ is associative if:
    $$a * (b * c) = (a * b) * c, \quad \forall a, b, c \in S$$
  - identity element: with respect to an operator $*$ on S is, if there exists an element e such that:
    $$e * a = a * e = a, \quad \forall a \in S$$
  - inverse: for every $a \in S$, if there exists a $b \in S$ such that:
    $$a * b = e$$
  - distributive law: with respect to two operators $*$ and $+$ if:
    $$a * (b + c) = (a * b) + (a * c)$$
    then $*$ is said to be distributes over $+$

Boolean algebra

- Boolean algebra is defined with a set of elements $\{0, 1\}$, a set of operators $\{+, ., \sim\}$ and a number of postulates
- 5-tuple of Boolean algebra:
$$\{B, +, ., \sim, 0, 1\}$$
- **The duality principle:** each postulate of Boolean algebra contains a pair of expressions or equations such that one is transformed into the other and vice-versa by interchanging the operators, $+ \leftrightarrow .$, and identity elements, $0 \leftrightarrow 1$. The two expressions are called the duals of each other.
- **Idempotency/Invariance theorem:** for all elements a in B, $a + a = a$ and $a.a = a$
- **Annulment theorem:** $a + 1 = 1$ and $a.0 = 0$
- **Involution theorem:** $\overline{\overline{a}} = a$
- **Absorption theorem:** $a + a.b = a$ and $a.(a + b) = a$
- **Adsorption theorem:** $a + \overline{a}.b = a + b$ and $a.(\overline{a} + b) = ab$
- **Uniting theorem:** $a.b + a.\overline{b} = a$ and $(a + b)(a + \overline{b}) = a$
- **De-Morgan's theorem:** $\overline{a + b} = \overline{a}.\overline{b}$ and $\overline{a.b} = \overline{a} + \overline{b}$
- **Consensus theorem:** $ab + \overline{a}c + bc = ab + \overline{a}c$ and $(a + b)(\overline{a} + c)(b + c) = (a + b)(\overline{a} + c)$
- Smallest number of literals means smallest number of complemented and uncomplemented variables
- NAND and NOR are universal operators
- Common canonical forms:
  - truth table
  - sum of minterms (SOM)
  - product of maxterms (POM)
  - binary decision diagram (BDD)
  - Reed Muller representation
- Minterms are AND terms with every variable present in either true or complemented form. Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n minterms for n variables.
- Functions are specified as sum of minterms using full minterms, or minterm indices. A function's complement includes all minterms not included in the original.
- Maxterms are OR terms with every variable in true or complemented form. Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n maxterms for n variables.
- Functions are specified as product of maxterms using full maxterms, or maxterm indices. A function's complement includes all maxterms not included in the original.
- Any Boolean function can be expressed as a sum of minterms

- The complement of a function expressed as a sum of minterms is constructed by selecting the minterms missing in the sum of minterms canonical forms. Alternatively, the complement of a function expressed by a sum of minterms form is simply the product of maxterms with the same indices.
- **Shannon's expansion theorem:**

$$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i.f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n) + \overline{x_i}.f(x_1, x_2, \ldots, x_i = 0, \ldots, x_n)$$

- **Decision tree:**
  - vertex represents decision
  - follow green (dashed) line for value 0
  - follow red (solid) line for value 1
  - function value determined by leaf value
  - assign arbitrary total ordering to variables
  - variables must appear in ascending order along all paths
- Binary decision diagram (BDD) reduction rules:
  - merge equivalent leaves
  - merge isomorphic nodes
  - eliminate redundant tests
  - function value determined by leaf value
  - assign arbitrary total ordering to variables
  - variables must appear in ascending order along all paths
- Functions are equal if and only if their ROBDDs are identical
- Selecting good variable ordering:
  - static ordering: fan in heuristic and weight heuristic
  - dynamic ordering: variable swap, window permutation and sifting
- A restriction to a function at $x = d$ denoted by $f|_{x=d}$, where $x \in var(f)$, and $d \in \{0, 1\}$ is equal to f after assigning $x = d$
- Let v_1, v_2 denote root nodes of f_1, f_2 respectively, with $var(v_1) = x_1$ and $var(v_2) = x_2$. If v_1 and v_2 are leafs, f_1 OP f_2 is a leaf node with value $val(v_1)$ OP $val(v_2)$.
- If all nodes are replaced by multiplexers in ROBDD, then cost is directly proportional to the number of nodes, and max delay is $N\tau$ where τ is delay for 1 mux and N is the number of muxes
- Functions decompositions:
  - Shanon's decomposition:

    $$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i.f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n) \oplus \overline{x_i}.f(x_1, x_2, \ldots, x_i = 0, \ldots, x_n)$$

  - Positive Davio decomposition:

    $$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = f(x_1, x_2, \ldots, x_i = 0, \ldots, x_n) \oplus x_i.(f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n) \oplus f(x_1, x_2, \ldots, x_i = 0, .$$

  - Negative Davio decomposition:

    $$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n) \oplus \overline{x_i}.(f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n) \oplus f(x_1, x_2, \ldots, x_i = 0, .$$

- Reed Muller representation is just the positive Davio decomposition. It's generalization is:

  $f(x_1, x_2, \ldots, x_i, \ldots, x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \ldots \oplus a_r x_1 x_2 \oplus \ldots \oplus a_p x_1 x_2 x_3 \oplus \ldots \oplus a_m x_1 x_2 \ldots x_n$

- Common non-canonical forms:
  - sum of product
  - product of sum
  - and invert graph (AIG)
- AIG is a Boolean network with two types of nodes:
  - two-input ANDs, nodes
  - inverters (NOT)
- Any Boolean function can be expressed using AIGs:
  - for many practical functions AIGs are smaller than BDDs
  - efficient graph representation (structural)
  - very good correlation with design size
- AIGs are not canonical:
  - for one function, there may be many structurally-different AIGs
  - functional reduction and structural hashing can make them "canonical enough"
- Converting logic function into AIG graph:
  - for many practical functions AIGs are smaller than BDDs
  - efficient graph representation (structural)
  - very good correlation with design size
  - for many practical functions AIGs are smaller than BDDs
  - efficient graph representation (structural)
- Operations for converting logic functions into AIG graph:
  - inversion: $\overline{a}$
  - conjunction: $a \wedge b$
  - disjunction: $\overline{(\overline{a} \wedge \overline{b})}$
  - implication: $\overline{(a \wedge \overline{b})}$
  - equivalence: $\overline{(a \wedge \overline{b})} \wedge \overline{(\overline{a} \wedge b)}$
  - XOR: $\overline{(\overline{(a \wedge \overline{b})} \wedge \overline{(\overline{a} \wedge b)})}$
- AIG size is measured by using number of AND nodes. AIG depth is measured by using the number of logic levels (number of AND-gates on longest path from a primary input to a primary output). The inverters are ignored when counting nodes and logic levels.
- SAT (Satisfiability) formulas for simple gates:
  - AND gate: $(\overline{c} + a)(\overline{c} + b)(c + \overline{a} + \overline{b})$
  - OR gate: $(c + \overline{a})(c + \overline{b})(\overline{c} + a + b)$
  - NOT gate: $(c + a)(\overline{c} + \overline{a})$
  - NAND gate: $(c + a)(c + b)(\overline{c} + \overline{a} + \overline{b})$

  Here a and b are the inputs and c is the output