CS203 - Digital Logic Design

Indian Institute of Technology Ropar

September 11, 2020

Contents

Chapter 1

Introduction to CS203

Digital means discrete in nature(values as well as time). For example, our computer handle discrete data only, therefore are digital computers.
Analog means continuous in nature(values as well as time). For example, the atmospheric variables, our senses, etc are analog.

1.1 Binary Classification

Classification into **two** groups if called *binary classification*. It is easy. For example, in computers, low voltage is classified as 0 and high voltage is classified as 1.

With N bits(binary digits), we can represent 2^N states.

Some rounding off-

- 10 bits - 2^{10} states $= 1024 \approx 10^3$
- 20 bits - 2^{20} states $= 2^{10} \times 2^{10} \approx 10^6$
- 30 bits - 2^{30} states $= 2^{10} \times 2^{10} \times 2^{10} \approx 10^9$ $\sim$ Population of India

1.2 Discretization

The process of discretization of analog signal involves setting discrete levels in values as well as time. Figure 1.1 gives some idea.
We lose some information when discretization occurs. To minimize the loss, we can use **Nyquest Criteria**. Also, simply increasing the number of levels will reduce the error.

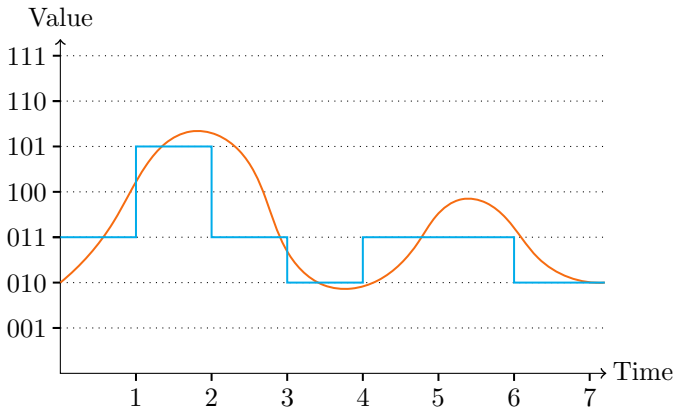
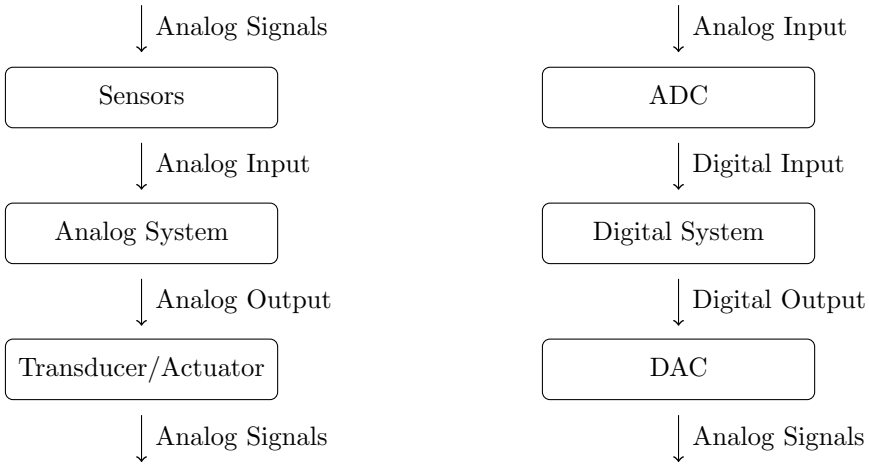


Figure 1.1: Discretization of analog signal

1.3 Analog vs Digial System

Our world is analog but our devices are digital. Figure 1.2 shows working of analog and digital systems.

ADC - Analog-to-digital converter
DAC - Digital-to-analog converter



(a) Analog System (b) Digital System

Figure 1.2: Analog and digial systems

1.3.1 Advantages of Digital System

- **Precision**
  - Small changes/error in signals does not affect the value. Even if errors occur in digital signal, it is easy to record the error and revert it.
  - Digital signals will produce same output for same input most of the time.
- **Programmability**
  - Each analog system is created for specific use case. But digital systems have generic gates which allow reprogramming.
- **Maintainability**
  - Digital signals are robust to change and can last longer(for years).
- **Design automation**

1.3.2 Disadvantages of Digital System

- **Area/Cost**
  - Cost of digital system is more than analog.
- **Power**
- **Performance**
  - Digital signals are slower.
- **Bandwidth**
- **High Frequency Operations**

1.3.3 Conclusion

Most systems are going towards digital. But whenever we need very specific solution that required high bandwidth and frequency, analog systems are used. Radio recievers, transmitters, etc. are mostly analog.

1.4 Factors pushing the growth story

- **Moore's law**
  - The number of transistors on a unit area of circuit doubles every 18 months.
  - It is not a law but has been pretty accurate till now.
- **Technology**
- **Compute Requirements**
- **Design Automation**

Chapter 2

Binary Representation

Historically, different bases(10, 12, 15, 16, 20) have been used. There were two kinds of number system

- **Positional**: Position of a number determines its value. For example, arabic, indic number systems.
- **Non-positional**: Value is largely decided by what symbol is used. For example, roman number system.

Finally, position based decimal system was accepted worldwide because it makes it easy to do calculations.

2.1 Positional Number System

Value of symbol depends on its position and radix/base.

$$N = (a_n a_{n-1} \dots a_0)_R = \sum_0^n a_i R^i$$

The definition can be easily extended to fractions.

$$N = (a_n a_{n-1} \dots a_0 a_{-1} \dots a_{-m+1} a_{-m})_R = \sum_0^n a_i R^i$$

Few examples

- $(101101)_2 = 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 45$
- $(101.101)_2 = 1 \times 2^2 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3} = 5.625$

Decimal to base-R conversion

For converting the number N in base 10 to a number in base R

1. Divide N by R, the remainder is a_0 and quotient is Q
2. Set N as Q and repeat the above process to get subsequent digits $a_1, a_2, \dots$
3. Stop when N becomes zero

Example: $(24)_{10} = (11000)_2$

R	N	Q	
2	24	12	$a_0 = 0$
2	12	6	$a_1 = 0$
2	6	3	$a_2 = 0$
2	3	1	$a_3 = 1$
2	1	0	$a_4 = 1$
2	0		

For converting fractional part F to base R

1. Multiply F with R, the non-fractional part is a_{-1} and fractional part is F'
2. Set F as F' and repeat the above process to get subsequent digits $a_{-2}, a_{-3}, \ldots$
3. Stop when F becomes zero

Example: $(0.7)_{10} = (0.10110011001100110011\cdots)_2$

R	F	F'	
2	0.7	0.4	$a_{-1} = 1$
2	0.4	0.8	$a_{-2} = 0$
2	0.8	0.6	$a_{-3} = 1$
2	0.6	0.2	$a_{-4} = 1$
2	0.2	0.4	$a_{-5} = 0$
2	0.4	0.8	$a_{-6} = 0$
2	0.8	0.6	$a_{-7} = 1$
2	0.6	0.2	$a_{-8} = 1$
$\vdots$	$\vdots$	$\vdots$	

2.2 Hexadecimal Number System

Hexadecimal system requires 16 symbols which are $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$. Hex numbers are generally prefixed by `0x`.

Conversion from binary to hex

To convert a binary number to hex, group the binary number into groups of four(nibbles) from right to left. Then, convert each nibble to hex using a lookup table or simple conversion.

$$(\texttt{10011110001})_2 \longrightarrow \texttt{100 1111 0001} \longrightarrow \texttt{0x4F1}$$

Conversion from hex to binary

To convert a hex to binary number, convert each digit to binary and make its length 4 by adding zeros as prefix. Now concatenate these so formed nibbles.

$$\texttt{0xBAD} \longrightarrow \texttt{1011 1010 1101} \longrightarrow (\texttt{101110101101})_2$$

2.3 Negative Numbers

2.3.1 Sign Magnitude Negative Numbers

We can set one bit to represent the sign. Generally, we choose the most-significant bit as the sign bit.

Example

$$(1010\ 0011)_2 = (-35)_{10}$$
$$(0111\ 1111)_2 = (127)_{10}$$

So the range of N-bit number is $-(2^{N-1}-1)$ to $(2^{N-1}-1)$.
Also, such representation has two zeros - positive and negative.

Addition and subtraction

Addition and subtraction becomes difficult.

- We have to look at the signs and then choose what operation will be performed.
- Two zeros create confusion.
- Either operation may result in addition or subtraction
- Complex hardware implementation
  - Requires both adder and subtractor
  - Requires controller that determine which hardware to use
  - Separate handling of sign

2.3.2 1's Complement

For obtaining $-u$ flip all the bits of u. Here also, N^{th} bit represents sign and $N-1$ bits represent magnitude.

Example

$$(+13)_{10} = (0000\ 1101)_2 \qquad (+69)_{10} = (0010\ 0101)_2$$
$$(-13)_{10} = (1111\ 0010)_2 \qquad (-69)_{10} = (1101\ 1010)_2$$

So the range of N-bit number is $-(2^{N-1}-1)$ to $(2^{N-1}-1)$.

Mathematical representation

We know $2^N - 1 = 111\ldots1$

$$\therefore -u = \sim |u| = 2^N - 1 - |u|$$

Addtion and subtraction

Addition and subtraction is difficult.

- In *end-around carry* condition, the wrapped bit must be added to the right-most bit.
- In *end-around borrow* condition, the wrapped bit must be subtracted from the right-most bit.
- There are two representation of zeros.

2.3.3 Bias Representation

Let $F(u)$ be the value of binary representation of u. In bias represent

$$F(u) = u + bias$$

Example (with bias = 127)

$$(1)_{10} = (1000\ 0000)_2$$
$$(-127)_{10} = (0000\ 0000)_2$$
$$(128)_{10} = (1111\ 1111)_2$$

Problems

- Bias should be adjusted while adding two numbers

$$F(u + v) = F(u) + F(v) - bias$$

- Bias should be standardized

2.3.4 2's Complement

When $u >= 0$

$$F(u) = |u|$$

When $u < 0$

$$F(u) = 2^N - |u| = \sim |u| + 1$$

Properties

- Single zero
- Most-significant bit represents sign(except in case of zero)
- $F(-u) = 2^N - F(u)$
- Range is from -2^{N-1} to $(2^{N-1} - 1)$

Arithmetic

- Addition

$$F(u + v) = F(u) + F(v)$$

- Subtraction

$$F(u - v) = F(u) + F(-v)$$

- Multiplication (assume no overflow)

$$F(u \times v) = F(u) \times F(v)$$

Overflow and Underflow

- If sign of both operands are same and result if of opposite sign or result is 0, overflow/underflow has occured.
- If sign of both operands are different, overflow cannot occur.

Converting N-bit number to M-bit number

To convert to N-bit number to M-bit number($M \geq N$) keep adding the sign bit as prefix until the size becomes M.
Example (Converting 4-bit number to 8-bit number)

$$(3)_{10} = (1101)_2 = (1111\ 1101)_2$$
$$(5)_{10} = (0101)_2 = (0000\ 0101)_2$$

2.3.5 Number Circle

An efficient way to see different representations in action.

- To add x to u, move x steps in clockwise direction from u.
- To subtract x from u, move x steps in counter-clockwise direction from u.
- Crossing dotted lines will result in underflow/overflow.

Visit https://thesis.laszlokorte.de/demo/number-circle.html to experiment.

2.4 Representing Decimal Numbers and Characters

We have to find a one-to-one mapping between binary combination and corresponding decimal value. *Is there any option better than positional number system?*.

2.4.1 Binary Coded Decimal (BCD) Number System

In BCD, each decimal digit is mapped to a nibble of its value. These nibbles are concatenated to get the binary representation.

8421 BCD number are numbers where the bits of nibbles have weights 8, 4, 2 and 1. It is the *default* BCD representation. **Example**

$$(5682)_{10} \longrightarrow (0101\ 01110\ 1000\ 0010)_2$$

Advantages

- No complex procedure for conversion from decimal representation required.
- BCD numbers are intuitive i.e. one can look at the binary number and grasp the value quickly.

Initial computer (e.g. IBM System/360) used BCD numbers.

Disadvantages

- During arithemtic, we have to take care that value of no nibble exceeds 9.
  In case, when value exceeds 9 (i.e. the value is not a valid BCD number), then add $(0110)_2$ to the result.
  Why did we add 6?
  [1] Because there are 6 invalid states in BCD numbers. To skip those states, we added 6.

2.4.2 Excess 3 Code

It is another BCD representation where value of each digit is 3 more in binary than in decimal.
Note: Be careful while adding numbers in *Excess 3 code*.

$$\text{Excess3Add}(u, v) = \text{Excess3}(u) + \text{Excess3}(v) - (0011)_2$$

Decimal	Excess 3 Code
0	0011
1	0100
2	0101
3	0110
⋮	⋮
8	1011
9	1100

Table 2.1: Excess 3 code mapping

Advantages

- **Self Complementing**
  9's complement can be obtained by inverting all the bits.

$$\text{Excess3}(9 - x) = \sim \text{Excess3}(x)$$

  It helps in doing subtration by addition.
  Consider the addition of two digits d_1 and d_2 (digits are represented in excess 3 code).

$$\text{Excess3Subtraction}(d_2, d_1) = \sim \text{Excess3Add}(\sim d_2, d_1)$$

2.4.3 Two-out-of-five Code

It is another BCD representation where each decimal digit is mapped to a group of 5 bits. Each digit's binary representation contains **exactly** two 1s.

Decimal	Two-out-of-five Code
0	00011
1	00101
2	00110
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

Table 2.2: Two-out-of-five code mapping

Advantages

- **Error Resilient**
  If any or several bits flip, there is a high chance that it will not contain 2 ones. In that case, we can know that the value is incorrect and we can redo the computation.

2.4.4 Gray Code

It is another BCD representation. It is a low power code because the trasititions between adjacent numbers is minimum.

Decimal	Gray Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	1110
6	1010
7	1011
8	1001
9	1000

Table 2.3: Gray code mapping

Advantages

- **Low Power Consumption**
  Since the transition between adjacent numbers is minimum, power consumption is minimized.

Gray Code Sequence

To generate Gray code sequence, do the following[2]:

1. Commence with the simplest Gray code possible; that is, for a single bit.
2. Create a mirror image of the existing Gray code below the original values.
3. Prefix the original values with 0s and the mirrored values with 1s.
4. Repeat step 2 and 3 until the desired width is achieved.

Start →	Mirror →	Prefix →	Mirror →	Prefix
0	0	00	00	000
1	1	01	01	001
	1	11	11	011
	0	10	10	010
			10	110
			11	111
			01	101
			00	100

Figure 2.1: Gray code generation upto 3 bits

2.5 Representing Characters

- **ASCII**
  Using 8-bit number for each commonly used character
- **UTF-8**
  - Compatible with ASCII
  - Uses 1 to 6 bytes
- **UTF-16**

2.6 Real Numbers

How would we represent a decimal point in binary?

2.6.1 Fixed Point Numbers

In this kind of numbers, we would fix some bits for only the fractional part.

Example (Assuming 4 decimal digits can be represented and decimal point is after 2 digits)

2300 would represent 23.00
123 would represent 1.23
1 would represent 0.01
100 would represent 1.00

2.6.2 Floating Point Numbers

To represent very large and very small number, we use scientific notation.

$$\text{In decimal} : 129 = 1.29 \times 10^2$$
$$\text{In binary} : 1000001 = 1.000001 \times 2^6$$

Such numbers have 3 parts:

1. **Significand**(s): The digit and sign before point
   Can only be $+1$ or -1
2. **Mantissa**(m): The fractional part of number
   Can only be positive
3. **Exponent**(e): The power of 2
   Can be positive or negative

Typical representation

- 1 bit is reserved for significand
- Next few bits for exponent
- Remaining bits for mantissa

2.6.3 IEEE 754 Single Precision Floating Point Numbers

32-bit number (single precision)

- 1 bit is reserved for significand (s)
- 8 bits for exponent (e)
  - Bias representation (with $bias = 127$)
    Bias method is sufficient because only addition is need in exponent
  - $(00000000)_2$ and $(11111111)_2$ are reserved for special purpose
- 23 bits for mantissa (m)
- $N = (-1)^s \times (1.m) \times 2^e$

[3] Visit https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/ for more information.
[4] https://www.h-schmidt.net/FloatConverter/IEEE754.html is an online converter.

Example

Converting 24.25 to IEEE 754 floating point number	
Decimal number	$24.25 = 16 + 8 + 0.25$
Binary number	11000.01
Scientific form	1.100001×2^4
Significand	0
Mantissa	100 0010 0000 0000 0000 0000
Exponent	$(127 + 4)_{10} = (1000\ 0011)_2$
Floating Point Number	0100 0001 1100 0010 0000 0000 0000 0000
In Hex	0x41920000

Converting -0.625 to IEEE 754 floating point number	
Decimal number	$-0.625 = -1 \times (0.5 + 0.125)$
Binary number	-0.101
Scientific form	-1.01×2^{-1}
Significand	1
Mantissa	010 0000 0000 0000 0000 0000
Exponent	$(127 + 0)_{10} = (1111\ 1111)_2$
Floating Point Number	1111 1111 1010 0000 0000 0000 0000 0000
In Hex	0xFFA00000

2.6.4 Denormal/subnormal Numbers

In a normal floating-point value, there are no leading zeros in the significand; rather, leading zeros are removed by adjusting the exponent (for example, the number 0.0123 would be written

Converting 0xC2630000 to decimal	
In Hex	0xC2630000
Floating Point Number	1100 0010 0110 0011 0000 0000 0000 0000
Significand	1
Exponent	$(100\ 0010\ 0)_2 = (127 + 5)_{10}$
Mantissa	110 0011 0000 0000 0000 0000
Scientific form	1.1100011×2^5
Binary number	111000.11
Decimal number	56.75

as 1.23×10^{-2}). Denormal numbers are numbers where this representation would result in an exponent that is below the smallest representable exponent (the exponent usually having a limited range). Such numbers are represented using leading zeros in the significand. [5]

2.6.5 Special Values

IEEE reserves exponent field values of all 0s and all 1s to denote special values in the floating-point scheme. [6]

Denormalized

If the exponent is all 0s, then the value is a *denormalized number* (section 2.6.4), which now has an assumed leading 0 before the binary point. Thus,

$$N = (-1)^s \times 0.m \times 2^{-126}$$

As denormalized numbers get smaller, they lose precision as left bits of the mantissa become zeros.

Zero

Zero is a denormalized number, with all bits in exponent and mantissa set to 0. If significand is 1, it represents -0, else +0. Also,

$$+0 = -0$$

Infinity

When exponent is all 1s and mantissa is all 0s, it represents *infinity*. If significand is 1, it represents $-\infty$, else $+\infty$.

Not a Number(NaN)

The value *NaN* is used to represent a value that does not represent a real number. Nan's are represented by a bit pattern with exponent of all 1s and a non-zero mantissa. There are two types on NaN's

1. **Quiet NaN (QNan)**
   A QNaN is a NaN with the most-significant mantissa bit set to 1. It represents *indeterminate operations*.

2. **Signalling NaN (SNan)**
   A SNaN is a NaN with the most-significant mantissa bit set to 0. It represents *invalid operations*.

2.6.6 IEEE 754 Half Precision Floating Point Numbers

16-bit number (half precision)

- 1 bit is reserved for significand (s)
- 5 bits for exponent (e)
  - Bias representation (with $bias = 15$)
  - $(00000)_2$ and $(11111)_2$ are reserved for special purpose
- 10 bits for mantissa (m)

2.6.7 IEEE 754 Double Precision Floating Point Numbers

64-bit number (double precision)

- 1 bit is reserved for significand (s)
- 11 bits for exponent (e)
  - Bias representation (with $bias = 1023$)
  - $(00000000000)_2$ and $(11111111111)_2$ are reserved for special purpose
- 52 bits for mantissa (m)

2.6.8 Issues with Floating Point Numbers

- **Quantization Issues**
  - Different rounding can lead to different results
  - Different formats have different precision
  - Accumulation of errors
  - Non-terminating numbers cannot be represented
- **Hardware/Computational Costs**

Chapter 3

Boolean Algebra

Boolean algebra is algebra on boolean values - *true* and *false*. These boolean values can be represented by bits - 0 for *false* and 1 for *true*.
A *boolean function* is a function that takes boolean inputs, performs only boolean operations and returns a boolean output. A *truth table* is a table listing outcomes of a function for every combination of input variables.

3.1 Basic Operations

There are three basic operations in boolean algebra.

1. **AND**
   A AND B is *true* iff both A and B are *true*. This is also called **product** in boolean algebra.

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.1: Truth table for AND

2. **OR**
   A OR B is *true* iff at least one of A and B is *true*. This is also called **sum** in boolean algebra.

A	B	$A \cdot B$
0	0	0
0	1	1
1	0	1
1	1	1

Table 3.2: Truth table for OR

3. **OR**
   NOT A is *true* iff A is *false*.

A	A'
0	1
1	0

Table 3.3: Truth table for NOT

3.2 Basic Theorems

- **Operation with 0**

$$X + 0 = X$$
$$X \cdot 0 = 0$$

- **Operation with 1**

$$X + 1 = 1$$
$$X \cdot 1 = X$$

- **Idempotent Law**

$$X + X = X$$
$$X \cdot X = X$$

- **Involution Law**

$$(X')' = X$$

- **Law of Complementarily**

$$X + X' = 1$$
$$X \cdot X' = 0$$

- **Commutative Law**

$$X + Y = Y + X$$
$$X \cdot Y = Y \cdot X$$

- **Associative Law**

$$(X + Y) + Z = X + (Y + Z) = X + Y + Z$$
$$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$$

- **Distributive Law**

$$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$$
$$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$$

- **DeMorgans's Law**

$$(X + Y)' = X' \cdot Y'$$
$$(X \cdot Y)' = X' + Y'$$

3.3 Duality Principle

0 is identity for OR. 1 is identity for AND.

Laws of boolean algebra remain same if operators and identity elements are interchanged.

In simple words, if in a boolean equation, all 0s and 1s are exchanged and all OR operators and all AND operators are exchanged with each other, the boolean equation continues to hold its validity.

3.4 Simplification Theorems

1. **Uniting**

$$X \cdot Y + X \cdot Y' = X$$
$$(X + Y) \cdot (X + Y') = X$$

2. **Absorption**

$$X + X \cdot Y = X$$
$$X \cdot (X + Y) = X$$

3. **Elimination**

$$X + (X' \cdot Y) = X + Y$$
$$X \cdot (X' + Y) = X \cdot Y$$

4. **Consensus**

$$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$$
$$(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$$

3.5 Standard Boolean Expressions

A, A', B, etc. are called *literals* and $AB, BC'A$, etc. are called *terms*.

1. **Sum of products (SOP)**
   EXAMPLES:
   - $AB' + BCD + A'C$
   - $A' + B' + C' + DE$
2. **Products of sum (POS)**
   EXAMPLES:
   - $(A + B')(B + C + D)(A' + C)$
   - $A'BC'(D + E)$

3.6 Minimizing Boolean Function

To reduce cost and improve performance, boolean functions are minimized. We will try to develope a systematic and procedural approach.

3.6.1 Canonical/Standard Forms

There can be several equivalent boolean expressions. Therefore, we need to have a standard form.

Minterm

We get 4 boolean terms by combining two variables x and y with logical AND operation. These boolean product terms are called *minterms* or *standard product terms.*[7]

Minterm

We get 4 boolean terms by combining two variables x and y with logical OR operation. These boolean product terms are called *maxterms* or *standard sum terms.*[7] There are 2 cannoical forms

i	**x**	**y**	**Minterm** (m_i)	**Maxterm** (M_i)
0	0	0	$x'y'$	$x+y$
1	0	1	$x'y$	$x+y'$
2	1	0	xy'	$x'+y$
3	1	1	xy	$x'+y'$

Table 3.4: Minterms and maxterms for two variables x and y

1. **Cannonical SoP - Sum of product** Each term should contain all the variables - either in normal or complemented form i.e. each term in SoP is a minterm. So a function in SoP form can be written as

$$F = m_{i_1} + m_{i_2} + \cdots = \sum m(i_1, i_2, \ldots)$$

2. **Cannonical PoS - Product of sum** Each sum term should contain all the variables - either in normal or complemented form i.e. each term in PoS is a maxterm. So a function in PoS form can be written as

$$F = M_{i_1} \cdot M_{i_2} \cdot \cdots = \prod M(i_1, i_2, \ldots)$$

Properties

- Cannonical PoS and SoP forms are complementary.

$$F = \prod M(0,1,2,4,6) = \sum m(3,5,7)$$

- DeMorgen's law

$$F = \sum m(3,5,7) = a'bc + ab'c + abc$$
$$F' = \prod M(3,5,7) = (a+b'+c')(a'+b+c')(a'+b'+c')$$

The expressions in cannonical form can be simplified using *uniting theorem*(theorem 1 in section 3.4).

3.6.2 Incompletely Specified Functions

An *incompletely specified function* is a boolean function that only define values for a subset of its inputs. The undefined outputs are called **don't care**.[8]

Example:

$$F(a,b) = \sum m(1) + \sum d(0,3)$$

a	**b**	**F(a, b)**
0	0	?
0	1	1
1	0	0
1	1	?

3.6.3 Karnaugh Map

Karnaugh maps are graphical method of minimizing boolean expressions.
Visit https://en.wikipedia.org/wiki/Karnaugh_map to learn about Karnaugh maps. [9]

https://www.charlie-coleman.com/experiments/kmap/ is a popular solver.

3.6.4 Quine-McCluskey Method

Karnaugh map becomes very difficult when number of variables increase. Here, Quine-McCluskey method comes to rescue.
Visit https://en.wikipedia.org/wiki/Quine-McCluskey_algorithm to learn about Quine-McCluskey methods. [10]

http://quinemccluskey.com/ is a popular solver. I think it failed for $f(a,b,c) = \sum m(0,1,2,5,6,7)$.

https://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/code/qmc/ is a good solver.

3.7 Logic Gates

We have already discussed 3 gates in section 3.1. We will see some more gates in this section.

1. **XOR**
   XOR of n-inputs is 1 if number of inputs having value 1 are odd.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.5: Truth table for XOR

A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Table 3.6: Truth table for XNOR

2. **XNOR**
   XNOR of n-inputs is 1 if number of inputs having value 1 are even. It is the inversion of XOR.

3. **NAND**
   NAND is AND followed by NOT.

A	B	$(A \cdot B)'$
0	0	1
0	1	1
1	0	1
1	1	0

Table 3.7: Truth table for NAND

4. **NOR**
   NOR is OR followed by NOT.

A	B	$(A + B)'$
0	0	1
0	1	0
1	0	0
1	1	0

Table 3.8: Truth table for NOR

NAND and NOR gates are universal gates. We can make all other gates using NAND or NOR gates only.

Bibliography

[1] BCD Addition - How to add BCD numbers? Solved Examples. https://www.learnpolytechnic.com/msbte-board/electronics-engineering/digital-techniques/bcd-addition/. Accessed: 2020-08-26.

[2] Gray Code Fundamentals - Part 2 — EE Times. https://www.eetimes.com/gray-code-fundamentals-part-2/. Accessed: 2020-08-24.

[3] IEEE 745 Standard Floating Point Number - GeeksForGeeks. https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/. Accessed: 2020-08-27.

[4] IEEE-745 Floating Point Number. https://www.h-schmidt.net/FloatConverter/IEEE754.html. Accessed: 2020-08-28.

[5] Denormal number - Wikipedia. https://en.wikipedia.org/wiki/Denormal_number. Accessed: 2020-08-28.

[6] IEEE Standard 754 Floating-Point. https://steve.hollasch.net/cgindex/coding/ieeefloat.html. Accessed: 2020-08-28.

[7] Digital Circuits - Cannonical & Standard Form - Tutorialspoint. https://www.tutorialspoint.com/digital_circuits/digital_circuits_canonical_standard_forms.html. Accessed: 2020-09-04.

[8] incompletely specified function - Boolean Algebra - WikiChip. https://en.wikichip.org/wiki/boolean_algebra/incompletely_specified_function. Accessed: 2020-09-04.

[9] Karnaugh map - Wikipedia. https://en.wikipedia.org/wiki/Karnaugh_map. Accessed: 2020-09-04.

[10] Quine-McCluskey algorithm - Wikipedia. https://en.wikipedia.org/wiki/Quine-McCluskey_algorithm. Accessed: 2020-09-11.