CS203 - Digital Logic Design

Indian Institute of Technology Ropar

August 24, 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.

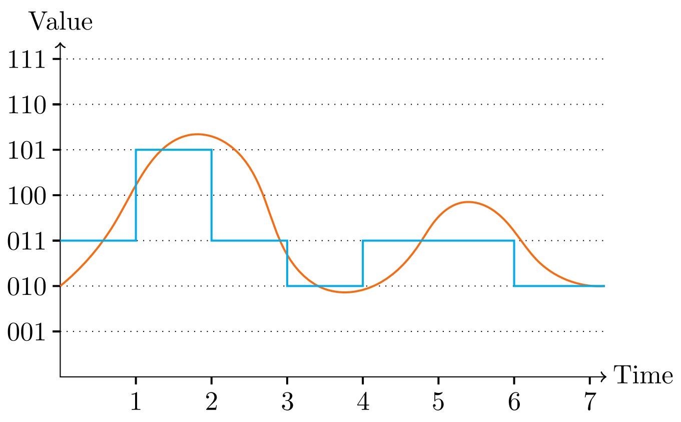
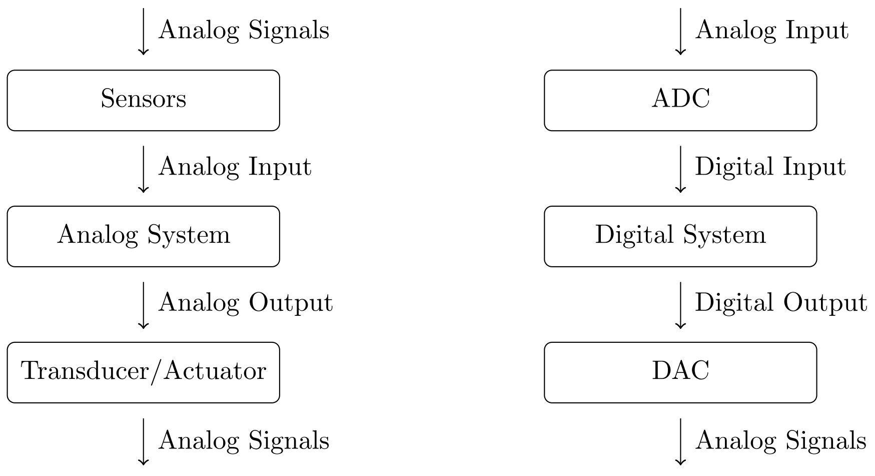


Figure 1.1: Discretization of analog signal

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**

1.5 Binary Representation

1.5.1 Traditional Number Systems

- 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.

1.5.2 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 $a_1, a_2, \ldots$
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	$a_{-8} = 1$
$\vdots$	$\vdots$	$\vdots$	

1.6 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.

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

1.7 Negative Numbers

1.7.1 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

$$(1001\ 0011)_2 = (-35)_{10}$$
$$(0111\ 1111)_2 = (127)_{10}$$

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

1.7.2 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.

1.7.3 Bias Representation

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

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

Example (with bias = 127)

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

Problems

- Bias should be adjusted while adding two numbers

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

- Bias should be standardized

1.7.4 Summary So Far

- Representation should be simple
- Two values of zero are not desirable
- Addition and subtraction should be easy (if possible done by same method)

1.7.5 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)

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

1.7.6 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.

1.8 Representing Decimal Numbers and Characters

1.8.1 Number Encoding Problem

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?.*

1.8.2 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**

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

Decimal	Excess 3 Code
0	0011
1	0100
2	0101
3	0110
$\vdots$	$\vdots$
8	1011
9	1100

Table 1.1: Excess 3 code mapping

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.

1.8.3 Excess 3 Code

It is another BCD representation where value of each digit is 3 more in binary than in decimal.

Advantages

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

  $$9 - x = \sim x$$

  It helps in doing subtration by addition.

1.8.4 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** 2 ones.

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.

1.8.5 Gray Code

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

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 1.2: Two-out-of-five code mapping

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

Table 1.3: 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[1]:

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 1.3: Gray code generation upto 3 bits

1.9 Representing Characters

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

Bibliography

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