Unit 1

Introduction
Number Systems and Conversion

Logic Circuits (Spring 2022)

Digital and Analog Systems

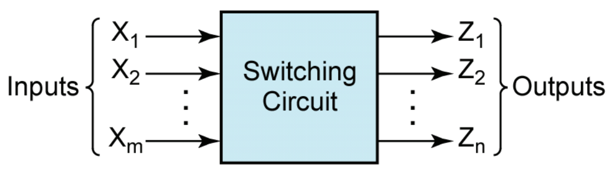
- Analog system
  - Physical quantities or signals can vary continuously over a specified range
  - The output of an analog system might have an error (ranging from a fraction of one percent to a few percents) because these systems do not work with discrete quantities
- Digital systems
  - Used extensively in computation and data processing, control systems, communications, and measurement
  - Physical quantities can only assume discrete values
  - In many cases, digital systems can be designed so that for a given input, the output is exactly correct.

Design of Digital Systems

- System design
  - Breaking the overall system into subsystems
  - Specifying the characteristics of each subsystem
  - Example: specifying the number and type of memory units, arithmetic units, I/O devices et al. and their interconnections
- Logic design
  - Determining how to interconnect basic logic building blocks to perform a specific function
  - Example: determining the interconnection of logic gates and flip-flops
- Circuit design
  - Specifying the interconnection of specific hardware components to form a gate, flip-flop or other logic building block
  - Hardware components: resistors, diodes, and transistors
  - Example: integrated circuits on a chip of silicon

Switching Circuit

- *Switching circuit*
  - One or more inputs and one or more outputs
  - Both inputs and outputs take on discrete values.



- *Combinational circuits*
  - Output values only depend on the present values of inputs
  - The past input values do not affect the output values
- *Sequential circuits*
  - Output values depend on both past and present input values
  - Sequential circuits are said to have "memory", because they must remember past sequence of inputs

Building Blocks of Switching Circuits

- *Logic gate*
  - Basic building block of combinational circuits
  - The relationship between the input and output signals can be described mathematically using *Boolean algebra*
- *Flip-flop*
  - Basic memory element of sequential circuits
- Switching devices
  - Usually two-state devices
  - The output can only assume one of two discrete values
  - Examples: relays, diodes, and transistors

Decimal and Binary Numbers

- Decimal (base 10) numbers
  - We use a positional notation
  - Each digit is multiplied by an appropriate power of 10 depending on its position in the number

$$953.78_{10} = 9 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$$

- Binary (base 2) numbers
  - Each digit is multiplied by an appropriate power of 2 on its position in the number

$$\begin{aligned} 1011.11_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\ &= 8 + 0 + 2 + 1 + \tfrac{1}{2} + \tfrac{1}{4} = 11\tfrac{3}{4} = 11.75_{10} \end{aligned}$$

  - It is natural to use binary numbers internally in digital systems
    ← the outputs of switching devices assume only two different values

Power Series Expansion

- Number system of *base* (*radix*) R
  - If the base is R, then R digits (0, 1, … $R-1$) are used
- A number can be expanded in a power series in R

$$
\begin{aligned}
N &= (a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3})_R \\
&= a_4 \times R^4 + a_3 \times R^3 + a_2 \times R^2 + a_1 \times R^1 + a_0 \times R^0 \\
&\qquad + a_{-1} \times R^{-1} + a_{-2} \times R^{-2} + a_{-3} \times R^{-3}
\end{aligned}
$$

  - a_i: the coefficient of R^i
  - $0 \le a_i \le R-1$
- Example

$$
\begin{aligned}
147.3_8 &= 1 \times 8^2 + 4 \times 8^1 + 7 \times 8^0 + 3 \times 8^{-1} = 64 + 32 + 7 + \tfrac{3}{8} \\
&= 103.375_{10}
\end{aligned}
$$

Conversion: Decimal Integer $\Rightarrow$ Base R

$$N = (a_n a_{n-1} \cdots a_2 a_1 a_0)_R = a_n R^n + a_{n-1} R^{n-1} + \cdots + a_2 R^2 + a_1 R^1 + a_0$$

If we divide N by R, the remainder is a_0:

$$\frac{N}{R} = a_n R^{n-1} + a_{n-1} R^{n-2} + \cdots + a_2 R^1 + a_1 = Q_1, \text{ remainder } a_0$$

Then we divide the quotient Q_1 by R:

$$\frac{Q_1}{R} = a_n R^{n-2} + a_{n-1} R^{n-3} + \cdots + a_3 R^1 + a_2 = Q_2, \text{ remainder } a_1$$

Next we divide Q_2 by R:

$$\frac{Q_2}{R} = a_n R^{n-3} + a_{n-1} R^{n-4} + \cdots + a_3 = Q_3, \text{ remainder } a_2$$

Conversion: Decimal Integer ⇒ Base R

Example Convert 53_{10} to binary.

$$
\begin{array}{rll}
2 \,\big/\, 53 & & \\
2 \,\big/\, 26 & \text{rem.} = 1 = a_0 & \\
2 \,\big/\, 13 & \text{rem.} = 0 = a_1 & \\
2 \,\big/\, 6 & \text{rem.} = 1 = a_2 & 53_{10} = \\
2 \,\big/\, 3 & \text{rem.} = 0 = a_3 & \\
2 \,\big/\, 1 & \text{rem.} = 1 = a_4 & \\
0 & \text{rem.} = 1 = a_5 &
\end{array}
$$

Conversion: Decimal Fraction ⇒ Base R

$$F = (.a_{-1}a_{-2}a_{-3}\cdots a_{-m})_R = a_{-1}R^{-1} + a_{-2}R^{-2} + a_{-3}R^{-3} + \cdots + a_{-m}R^{-m}$$

Multiplying by R yields

$$FR = a_{-1} + a_{-2}R^{-1} + a_{-3}R^{-2} + \cdots + a_{-m}R^{-m+1} = a_{-1} + F_1$$

where F_1 represents the fractional part of the result and a_{-1} is the integer part. Multiplying F_1 by R yields

$$F_1R = a_{-2} + a_{-3}R^{-1} + \cdots + a_{-m}R^{-m+2} = a_{-2} + F_2$$

Next, we multiply F_2 by R:

$$F_2R = a_{-3} + \cdots + a_{-m}R^{-m+3} = a_{-3} + F_3$$

This process is continued until we have obtained a sufficient number of digits.

Conversion: Decimal Fraction $\Rightarrow$ Base R

Example

Convert 0.625_{10} to binary.

$$
\begin{array}{ccc}
F = \ .625 & F_1 = \ .250 & F_2 = \ .500 \\
\times \quad 2 & \times \quad 2 & \times \quad 2 \\
\underline{1.250} & \underline{0.500} & \underline{1.000} \\
(a_{-1} = 1) & (a_{-2} = 0) & (a_{-3} = 1)
\end{array}
$$

$.625_{10} =$

Conversion: Decimal Fraction $\Rightarrow$ Base R

Example

Convert 0.7_{10} to binary.

$$
\begin{array}{r}
.7 \\
\underline{\quad 2} \\
(1).4 \\
\underline{\quad 2} \\
(0).8 \\
\underline{\quad 2} \\
(1).6 \\
\underline{\quad 2} \\
(1).2 \\
\underline{\quad 2} \\
(0).4 \\
\underline{\quad 2} \\
(0).8
\end{array}
$$

$\longleftarrow$ process starts repeating here because 0.4 was previously obtained

$0.7_{10} = 0.1\underline{0110}\,\underline{0110}\,\underline{0110}\ \ldots_2 \ =$

Conversion Between Any Two Bases

Example Convert 231.3_4 to base 7.

$$231.3_4 = 2 \times 16 + 3 \times 4 + 1 + \tfrac{3}{4} = 45.75_{10}$$

$$\begin{array}{lll} 7\,/\underline{45} & & \\ 7\,/\underline{\ 6} & \text{rem. } 3 & \\ \quad 0 & \text{rem. } 6 & \end{array} \qquad \begin{array}{r} .75 \\ \underline{7} \\ (5).25 \\ \underline{7} \\ (1).75 \\ \underline{7} \\ (5).25 \\ \underline{7} \\ (1).75 \end{array} \qquad 45.75_{10} =$$

Conversion: Binary $\Rightarrow$ Hexadecimal

- Rule of thumb
  - Each hexadecimal digit corresponds to exactly four binary digits
  - Starting at the binary point, the bits are divided into groups of four
  - Each group is replaced by a single hexadecimal digit

$$1001101.010111_2 = \underbrace{0100}_{4}\ \underbrace{1101}_{D}\ .\ \underbrace{0101}_{5}\ \underbrace{1100}_{C} =$$

Binary Arithmetic

- Addition table for binary numbers

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0 \quad \text{and carry 1 to the next column}$$

Carrying 1 to a column is equivalent to adding 1 to that column.

- Subtraction table for binary numbers

$$0 - 0 = 0$$
$$0 - 1 = 1 \quad \text{and borrow 1 from the next column}$$
$$1 - 0 = 1$$
$$1 - 1 = 0$$

Borrowing 1 from a column is equivalent to subtracting 1 from that column.

Binary Arithmetic

- Multiplication table for binary numbers

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Binary Addition

Example

Add 13_{10} and 11_{10} in binary.

$$
\begin{array}{rrl}
 & 1111 & \longleftarrow \text{carries} \\
13_{10} = & 1101 & \\
11_{10} = & \underline{1011} & \\
 & 11000 & = 24_{10}
\end{array}
$$

Binary Subtraction

Examples of Binary Subtraction

(a)
$$
\begin{array}{rl}
1\longleftarrow & \text{(indicates} \\
11101 & \text{a borrow} \\
\underline{-\,10011} & \text{from the} \\
1010 & \text{3rd column)}
\end{array}
$$

(b)
$$
\begin{array}{rl}
1\,1\,1\,1\longleftarrow & \text{borrows} \\
10000 & \\
\underline{-\quad\;\; 11} & \\
1101 &
\end{array}
$$

(c)
$$
\begin{array}{rl}
1\,1\,1\longleftarrow & \text{borrows} \\
111001 & \\
\underline{-\quad 1011} & \\
101110 &
\end{array}
$$

Binary Multiplication

1101	1111	multiplicand
1011	1101	multiplier
1101	1111	first partial product
1101	0000	second partial product
0000	(01111)	sum of first two partial products
1101	1111	third partial product
10001111	(1001011)	sum after adding third partial product
	1111	fourth partial product
	11000011	final product (sum after adding fourth partial product)

Binary Division

```
            1101
1011 ) 10010001
          1011
           1110
           1011
            1101
            1011
              10
```

The quotient is 1101 with a remainder of 10.

Representation of Negative Numbers

- Common methods of representing positive and negative numbers
  - *Sign & magnitude*
  - *2's complement*
  - *1's complement*
- The leftmost bit of a number
  - 0 for positive numbers
  - 1 for negative numbers

Representation of Negative Numbers

- *Sign & magnitude*
  - Most significant bit is the sign
  - $-5_{10} = 1101_2$
- *2's complement*
  - $N^* = 2^n - N$
  - $-5_{10} = 2^4 - 5 = 16 - 5 = 11_{10} = 1011_2$
  - $-5_{10} = 10000_2 - 0101_2 = 1011_2$
- *1's complement*
  - $\overline{N} = (2^n - 1) - N$
  - $-5_{10} = (2^4 - 1) - 5 = 16 - 1 - 5 = 10_{10} = 1010_2$
  - $-5_{10} = 1111_2 - 0101_2 = 1010_2$

Representation of Negative Numbers

	Positive Integers (all systems)		Negative Integers		
$+N$		$-N$	Sign and Magnitude	2's Complement N^*	1's Complement $\overline{N}$
+0	0000	−0	1000	——	1111
+1	0001	−1	1001	1111	1110
+2	0010	−2	1010	1110	1101
+3	0011	−3	1011	1101	1100
+4	0100	−4	1100	1100	1011
+5	0101	−5	1101	1011	1010
+6	0110	−6	1110	1010	1001
+7	0111	−7	1111	1001	1000
		−8	——	1000	——

Addition of 2's Complement Numbers

1. Addition of two positive numbers, sum $< 2^{n-1}$

+3	0011	
+4	0100	
+7		(correct answer)

2. Addition of two positive numbers, sum $\geq 2^{n-1}$

+5	0101	
+6	0110	
		⟵ wrong answer because of overflow (+11 requires 5 bits including sign)

Addition of 2's Complement Numbers

3. Addition of positive and negative numbers (negative number has greater magnitude)

+5	0101	
−6	1010	
−1		(correct answer)

4. Same as case 3 except positive number has greater magnitude

−5	1011	
+6	0110	
+1		⟵ correct answer when the carry from the sign bit is ignored (this is *not* an overflow)

Case 4: $-A + B$ (where $B > A$)

$$A^* + B = (2^n - A) + B = 2^n + (B - A) > 2^n$$

Addition of 2's Complement Numbers

5. Addition of two negative numbers, $|\text{sum}| \le 2^{n-1}$

−3	1101	
−4	1100	
−7		⟵ correct answer when the last carry is ignored (this is *not* an overflow)

6. Addition of two negative numbers, $|\text{sum}| > 2^{n-1}$

−5	1011	
−6	1010	
		⟵ wrong answer because of overflow (−11 requires 5 bits including sign)

Case 5: $-A - B$ (where $A + B \le 2^{n-1}$)

$$A^* + B^* = (2^n - A) + (2^n - B) = 2^n + 2^n - (A + B)$$

Addition of 2's Complement Numbers

Add −8 and +19 in 2's complement

+8 = 00001000

complementing all bits to the left of the first 1, −8, is represented by 11111000

$$
\begin{array}{rl}
11111000 & (-8) \\
\underline{00010011} & \underline{+19} \\
= & +11
\end{array}
$$

$\uparrow$ (discard last carry)

Binary Codes for Decimal Numbers

- Why binary codes?
  - Although most large computers work internally with binary numbers, the input-output equipment generally uses decimal numbers
  - The decimal numbers must be coded in terms of binary signals, because most logic circuits only accept two-valued signals
- The simplest form of binary code
  - Each decimal digit is replaced by its binary equivalent
  - Example: 937.25 is represented by

9 3 7 . 2 5

1001 0011 0111 . 0010 0101

Binary Codes for Decimal Numbers

Decimal Digit	8-4-2-1 Code (BCD)	6-3-1-1 Code	Excess-3 Code	2-out-of-5 Code	Gray Code
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	1110
6	0110	1000	1001	10001	1010
7	0111	1001	1010	10010	1011
8	1000	1011	1011	10100	1001
9	1001	1100	1100	11000	1000

Binary Codes for Characters – ASCII Code

Character	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Character	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Character	A_6	A_5	A_4	A_3	A_2	A_1	A_0
space	0	1	0	0	0	0	0	@	1	0	0	0	0	0	0	‘	1	1	0	0	0	0	0
!	0	1	0	0	0	0	1	A	1	0	0	0	0	0	1	a	1	1	0	0	0	0	1
“	0	1	0	0	0	1	0	B	1	0	0	0	0	1	0	b	1	1	0	0	0	1	0
#	0	1	0	0	0	1	1	C	1	0	0	0	0	1	1	c	1	1	0	0	0	1	1
$	0	1	0	0	1	0	0	D	1	0	0	0	1	0	0	d	1	1	0	0	1	0	0
%	0	1	0	0	1	0	1	E	1	0	0	0	1	0	1	e	1	1	0	0	1	0	1
&	0	1	0	0	1	1	0	F	1	0	0	0	1	1	0	f	1	1	0	0	1	1	0
’	0	1	0	0	1	1	1	G	1	0	0	0	1	1	1	g	1	1	0	0	1	1	1
(	0	1	0	1	0	0	0	H	1	0	0	1	0	0	0	h	1	1	0	1	0	0	0
)	0	1	0	1	0	0	1	I	1	0	0	1	0	0	1	i	1	1	0	1	0	0	1
*	0	1	0	1	0	1	0	J	1	0	0	1	0	1	0	j	1	1	0	1	0	1	0
+	0	1	0	1	0	1	1	K	1	0	0	1	0	1	1	k	1	1	0	1	0	1	1
,	0	1	0	1	1	0	0	L	1	0	0	1	1	0	0	l	1	1	0	1	1	0	0
–	0	1	0	1	1	0	1	M	1	0	0	1	1	0	1	m	1	1	0	1	1	0	1
.	0	1	0	1	1	1	0	N	1	0	0	1	1	1	0	n	1	1	0	1	1	1	0
/	0	1	0	1	1	1	1	O	1	0	0	1	1	1	1	o	1	1	0	1	1	1	1
0	0	1	1	0	0	0	0	P	1	0	1	0	0	0	0	p	1	1	1	0	0	0	0
1	0	1	1	0	0	0	1	Q	1	0	1	0	0	0	1	q	1	1	1	0	0	0	1
								…	…	…	…	…											
=	0	1	1	1	1	0	1	]	1	0	1	1	1	0	1	}	1	1	1	1	1	0	1
>	0	1	1	1	1	1	0	^	1	0	1	1	1	1	0	~	1	1	1	1	1	1	0
?	0	1	1	1	1	1	1	—	1	0	1	1	1	1	1	delete	1	1	1	1	1	1	1