Switching Theory and Digital Design

Deptt. Of CS&E, IIT Madras

Digital vs Analog

- Digital Systems – More Accuracy and Reliability
- Analog Systems – Errors introduced by Noise

Digital Systems

- **System Design**
- **Logic Design**
- **Circuit Design**

Digital Systems

- **System Design**
  - Issues in breaking the overall system into sub-systems and specifying the characteristics of each sub-system (e.g. Memory, CPU, I/O Bus control).
- **Logic Design**
  - Determine how to interconnect basic logic building blocks to perform a specific function. E.g. Connect logic gates and Flip-Flops
- **Circuit Design**
  - Specify the interconnection of/and specific components. E.g. Resistors, Diodes, Transistors etc. to form a gate and other logic building blocks

Combinational and Sequential

- **Combinational Circuits**.
  - Operation depends on the current state of the input or *Present State* but not the past state

- **Sequential Circuits**
  - Output depends on the previous and the present stages. Hence it has some memory (not required in Combinational).

Combinational Circuits

- Design of Combinational Network involves the interconnection of logic gates

- Output input relationship can be described mathematically using the *Boolean algebra*

- CLN (Combinational Logic n/w) be designed as
  - Derive a table or algebraic logic Equation
  - Simplify using the K-Maps or Q-M procedures
  - Use different gates to realize the reduced logic equation

Sequential Circuits

- Use memory elements – Flip-Flops (F/F)
- F/F are interconnected with gates to form counters and Registers
- General Sequential Circuits designed using *Timing Diagrams.*
- Combinational and Sequential circuit design technique are used to build ADDER, SUB, MULT and DIV
- *Asynchronous Seq. N/w are most difficult.*

Binary Arithmetic

- I/p and o/p of switching devices assume only different and distinct values

- So Binary Number Systems is used in all digital systems

- Decimal System

$(938.75)_{10} = 9x10^{2} + 3x10^{1} + 8x10^{0} + 7x10^{-1} +5x10^{-2}$

- Binary System

$(1101.01)_{2} = 1x2^{3} + 1x2^{2} + 0x2^{1} + 1x2^{0} + 0x2^{-1} +1x2^{-2}$

- For any number system, Radix or base (R) is a +ve integer. (Here R =10 for decimal, 2 for binary)
- For base R, (R-1) digits are used for representing the number
- $M = (a_n\ a_{n-1}\ ..\ a_0.\ a_{-1}\ a_{-2}\ ..\ a_{-m})_R$

  $= a_nR^n + a_{n-1}R^{n-1} + \ldots + a_0R^0 + a_{-1}R^{-1} + a_{-2}R^{-2} \ldots + a_{-m}R^{-m}$

  where, $0 <= a_i <= (R-1)$

- **If R > 10**
  - **0,1,…,9, A, B, C, D, E, F**

- **E.g. $(A2F)_{16} = 10x16^2 + 2x16^1\ 15x16^0 = (2607)_{10}$**

Any base to decimal:

Binary 1 1 1 1 1 0 1 0 0 0 1

$1\times2^{10}+1\times2^{9}+1\times2^{8}+1\times2^{7}+1\times2^{6}+0\times2^{5}+1\times2^{4}+0\times2^{3}+0\times2^{2}+0\times2^{1}+1\times2^{0}$

$1024+512+256+128+64+0+16+0+0+0+1$

Octal 3 7 2 1

$3\times8^{3}+7\times8^{2}+2\times8^{1}+1\times8^{0}$

$1536+448+16+1$

Decimal 2 0 0 1

$2\times10^{3}+0\times10^{2}+0\times10^{1}+1\times10^{0}$

$2000+0+0+1$

Hexadecimal 7 D 1

$7\times16^{2}+13\times16^{1}+1\times16^{0}$

$1792+208+1$

Read about conversion from any base to another.

$M = (a_n\ a_{n-1}\ ..\ a_0\ .\ a_{-1}\ a_{-2}\ ..\ a_{-m})_R$

$= a_nR^n + a_{n-1}R^{n-1} + ... + a_0R^0 + a_{-1}R^{-1} + a_{-2}R^{-2} .... + a_{-m}R^{-m}$

where, $0 <= a_i <= (R-1)$

Steps to convert from any base to another:

- First Convert to decimal
- Convert decimal output to new base

Solve:

$(231.3)_4 = (\ ??\ )_7$

$(231.3)_4 = (45.75)_{10}$

$(45.75)_{10}$ =>

- DIV(45,7) => Q1 = 6; Rem1 = 3;
- DIV(Q1,7) => Q2 = 0; Rem2 = 6;
- Mult(.75 * 7) => INT1 = 5; Frac1 = 0.25 ;
- Mult(.25 * 7) => INT1 = 1; Frac1 = 0.75 ;
- Mult(.75 * 7) => INT1 = 5; Frac1 = 0.25 ;
- Mult(.25 * 7) => INT1 = 1; Frac1 = 0.75 ; and so on

ANS:

$(63\ .\ \underline{51}\ \underline{51}.....)_7$

A few special cases of base conversion:

Binary to Octal : $(11\ 010\ 111\ 110\ .\ 001\ 1)_2$

$(3\quad 2\quad 7\quad 6\ .\ 1\quad 4)_8$

Binary to Hexadecimal : $(100\quad 1101\ .\ 0101\quad 11)_2$

$(4\quad D\ .\ 5\quad C)_{16}$

What do you think of the floating point number representation:

$$M.\beta^E$$

Convert a Base 3 number into a Base 5 number in one step?

Each subsequent digit starting from the right will be the remainder when you divide your current working total by ($(12)_3$ = (5 in base 3).

The only tricky parts here are keeping track of what base you're in and remembering to convert to base 5 for each digit at the end

Let's demonstrate with an example:

Convert 12210 base 3 (156 in dec) to base 5:

All math is in base 3:

Another example:

Convert 10211 base 3 (103 in dec) to base 5:

Binary Codes

- Number representation Internal to the computer – *Binary*
- For human beings – Decimal
- So, any I/O interface must *Convert* from Decimal to Binary
- Finally the Binary bits are transmitted in terms of binary signals

Binary Codes

Binary	Decimal Value	
	Unsigned	-N Signed Mag.
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	- 0
1001	9	- 1
1010	10	- 2
1011	11	- 3
1100	12	- 4
1101	13	- 5
1110	14	- 6
1111	15	- 7

Binary Codes

N decimal	N binary	–N signed mag.	–N 1's compl.	–N 2's compl.
1	00000001	10000001	11111110	11111111
2	00000010	10000010	11111101	11111110
3	00000011	10000011	11111100	11111101
4	00000100	10000100	11111011	11111100
5	00000101	10000101	11111010	11111011
6	00000110	10000110	11111001	11111010
7	00000111	10000111	11111000	11111001
8	00001000	10001000	11110111	11111000
9	00001001	10001001	11110110	11110111
10	00001010	10001010	11110101	11110110
20	00010100	10010100	11101011	11101100
30	00011110	10011110	11100001	11100010
40	00101000	10101000	11010111	11011000
50	00110010	10110010	11001101	11001110
60	00111100	10111100	11000011	11000100
70	01000110	11000110	10111001	10111010
80	01010000	11010000	10101111	10110000
90	01011010	11011010	10100101	10100110
100	01100100	11011010	10011011	10011100
127	01111111	11111111	10000000	10000001
128	Nonexistent	Nonexistent	Nonexistent	10000000

BCD

- BCD format needs 4 bits for each decimal digit
- Is a way to represent binary numbers in decimal format
- BCD is not the same as binary representation !
- $1234_{10} = 1 \quad 2 \quad 3 \quad 4_{10} =$
- $0001 \quad 0010 \quad 0011 \quad 0101_{BCD} =$
- $= 1001000110101_{BCD} =$
- $= 10011010010_2 =$
- $= 4D2_{16}$

Excess-3 code

- Excess-3 code: Given a decimal digit *n*, its corresponding excess-3 codeword is binary code $(n+3)_2$

- Example:

  $n = 5 \rightarrow n+3 = 8 \rightarrow 1000_{\text{excess-3}}$

  $n = 0 \rightarrow n+3 = 3 \rightarrow 0011_{\text{excess-3}}$

Grey-Code

- Grey-code is one where only one bit changes at a time.
- Codes of successive decimal digits differ in exactly one bit.

The following tables show the difference between three-bit Binary numbers and Gray-coded numbers

Decimal	Binary	Gray-code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	100
6	110	101
7	111	111

Decimal	Binary	Gray-code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Problem of uniqueness in representation

2-out-of-5 code

- 2-out-of-5 code
  - Exactly *2 out of the 5 bits are 1* for every valid code combination
  - Good error checking properties

- The Gray code and 2-out-of-5 code are non-weighted codes. The decimal values of a coded digit cannot be computed by a simple formula, in case of non-weighted codes

Table of Binary Codes

Decimal	BCD	6-3-1-1	Excess 3	2-out of 5	Gray code
0	0000	0	11	00011	0000
1	0001	1	100	00101	0001
2	0010	11	101	00110	0011
3	0011	100	110	01001	0010
4	0100	101	111	01010	0110
5	0101	111	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 Addition

Adding binary numbers:

1 + 0 = 0 + 1 = 1; 0 + 0 = 0;   1 + 1 = 0, with carry 1

Addend	0	0	1	1
Augend	+0	+1	+0	+1
Sum	0	1	1	0
Carry	0	0	0	1

Input		**Output**	
A	**B**	**C**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Add $\mathbf{109_{10}}$ **to** $\mathbf{136_{10}}$**:**

01101101 + 10001000

= 11110101

= $\boldsymbol{235_{10}}$

Binary Subtraction

- Unsigned numbers: minus sign is not explicitly represented.
- Given 2 binary numbers M and N, find M-N:
  - Case I: M ≥ N, thus, MSB of Borrow is 0

B	0 0 0 1 1 0		
M	1 1 1 1 0	30	
N	-1 0 0 1 1	- 19	Result is Correct
Dif	0 1 0 1 1	11	

  - Case II: N > M, thus MSB of Borrow is 1

B	1 1 1 0 0 0		
M	1 0 0 1 1	19	
N	-1 1 1 1 0	- 30	Result requires correction!
Dif	1 0 1 0 1	21	

Input		Output	
M	N	B	S
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

1-bit Adder unit:

Input		Output	
A1	**A2**	**C**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

1-bit Subtractor unit:

Input		Output	
A1	**A2**	**B**	**S**
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

C – Carry to next column;

B – Borrow from next column

Some more work out examples:

Input		Output	
M	N	B	S
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

				1		
		1	1	1	0	1
	-	1	0	0	1	1
			1	0	1	0

		1	1	1	1	
		1	0	0	0	0
	-				1	1
			1	1	0	1

		1	1	1		
	1	1	1	0	0	1
	-		1	0	1	1
	1	0	1	1	1	0

- In general, if N > M, Dif = $M-N+2^n$,
  where, n = # bits.
- In Case II of the previous example,

  $$\text{Dif} = 19 - 30 + 2^5 = 21.$$

- To correct the magnitude of Dif, which should be N-M, calculate $2^n - (M-N+2^n)$.

  $$= 32 - 21 = 11 \text{ (check out binary)}$$

- This is known as the ***2'***s complement of Dif.

- To subtract two n-bit numbers, M-N, in base 2:
  - Find M-N.
  - If MSB of Borrow is 0, then M ≥ N. Result is positive and correct.
  - If MSB of Borrow is 1, then N > M. Result is negative and its magnitude must be corrected by subtracting it from 2^n (find its 2's complement).

- Given M = 01100100 and N = 10010110, find M-N.

B	1 0 0 1 1 1 1 0 0			
M	0 1 1 0 0 1 0 0	100		
N	-1 0 0 1 0 1 1 0	-150		
Dif	**1 1 0 0 1 1 1 0**	**206**		

Input		Output	
A1	**A2**	**B**	**S**
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

2^n	1 0 0 0 0 0 0 0 0	256
Dif	- 1 1 0 0 1 1 1 0	- 206
	0 0 0 1 1 0 0 1 0	**50**

Check the 2's complement relationship of the two results.

Binary Multiplication

- Example:
  - Multiplier $A=A_1A_0$ and multiplicand $B=B_1B_0$
  - Find C = AxB:

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & \times & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
+ & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

Multiplication rules

0x0 = 1

0x1 = 0

1x0 = 0

1x1 = 1

Example

Multiply $(5)_{10}$ by $(3)_{10}$

```
  101
 * 11
 ----
  101
 101x
 ----
 1111
```

Result $(15)_{10}$ or $(1111)_2$

Multiply $(13)_{10}$ x $(10)_{10}$

```
    1101
  x 1010
  ------
    0000
   1101x
  0000xx
 1101xxx
--------
10000010
```

Result $(130)_{10}$ or $(10000010)_2$

Binary Division

Divide $(59)_{10}$ by $(3)_{10}$

```
11)111011(  1 0 0 1 1
   11
   ---------------
       101
        11
       -------------
        101
         11
        ---------
          10
```

Quotient – **$(10011)_2$ / $(19)_{10}$**

Remainder – **$(10)_2$ / $(2)_{10}$**

REFERENCES

- Fundamentals of Logic design; Charles C. Roth, Jr., 4th Edn., Jaico Pubcln., 1999+.
- Digital Logic and Computer Design; M. Morris Mano; Prentice-Hall India, 1998+.
- Microelectronics; Millman; McGraw-Hill, 2000.