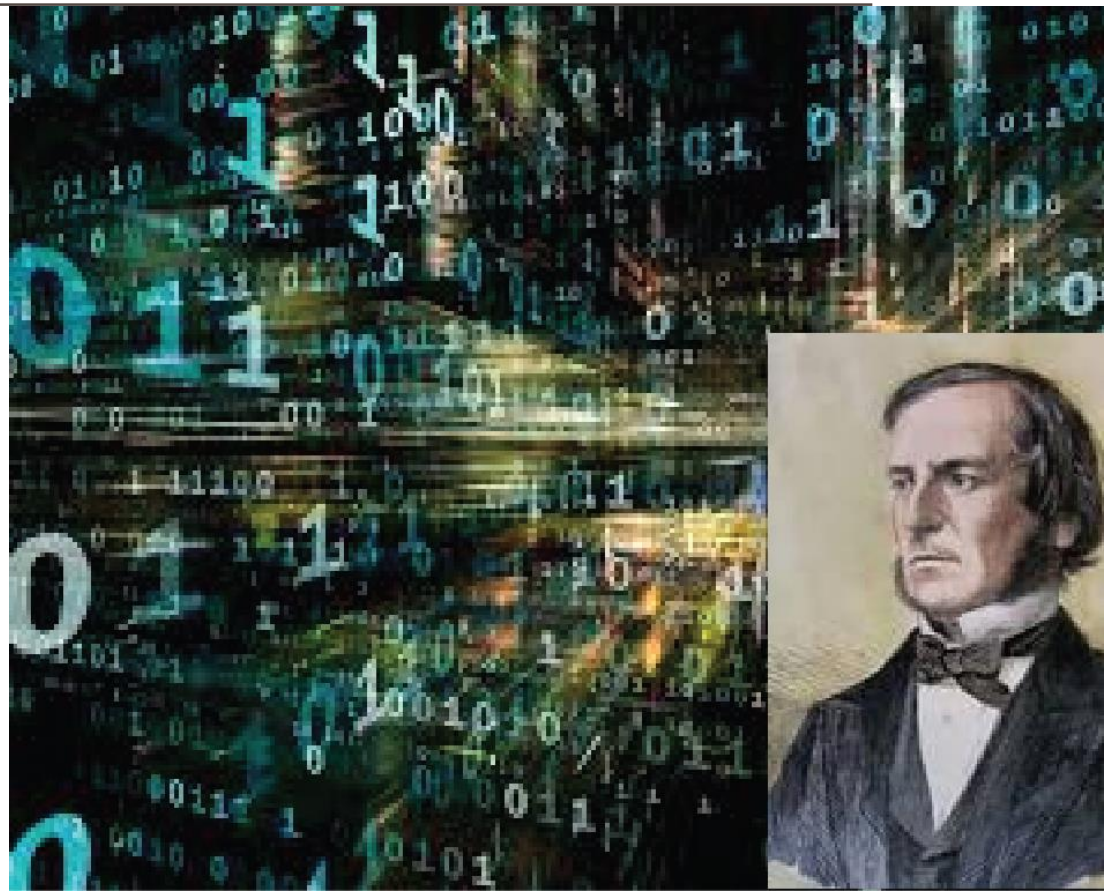
DIGITAL CIRCUITS

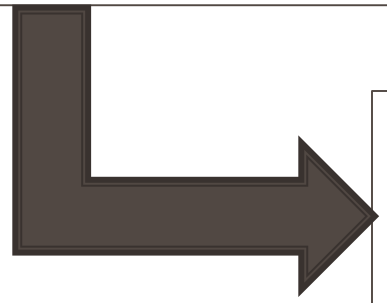


Week-6, Lecture-2
Codes

Sneh Saurabh
5th September, 2018

Digital Circuits: Announcements/Revision

Digital Circuits



Binary Codes

Binary codes: Introduction

- Binary codes are patterns/group of zeros and ones
- Any discrete information that is distinct within a group can be represented using binary codes
- Binary codes merely change the symbol representing data: the meaning of data is not changed
- Motivation for using binary code is ease of manipulation, storage or transmission of data

Binary codes: Number of bits

- An $n - bit$ binary code is a group of $n - bits$ that assumes up to 2^n combination of zeroes and ones
- Each combination represent one element of the set that is being encoded: code should be unique for each element
- Though minimum number of bits required to represent 2^n elements is n, there is no maximum number of bits that can be used in encoding (some bits may be unused)

Binary Coded Decimal (BCD)

Table 5.3 Binary-coded decimal digits.

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- Each decimal digit is given one binary code
- In BCD, each decimal digit is assigned the corresponding $4-bit$ binary number
- Last 6 combinations of $4-bit$ binary number are not used

Problem:

Represent $(185)_{10}$ in:

a) BCD

b) Binary

Answer:

a) BCD: $(185)_{10} = (0001\ 1000\ 0101)_{BCD}$

b) Binary: $(185)_{10} = (1011\ 1001)_2$

- BCD numbers are not binary numbers
- In general BCD numbers take more number of bits than binary numbers

Other Binary Codes for Decimal Digits

- Decimal digits (10 in numbers) require minimum of 4 bits
- Four bits give 16 bit combinations
- Several binary codes can be generated by choosing any 10 out of 16 bit combinations

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
	1011	0110	0001	0010
Unused bit combi-nations	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

Binary Codes for Decimal Digits: Weighted Codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi-nations	1010	0101	0000	0001
	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

Weighted codes:

- Each bit position is assigned a weighting factor
- Each decimal digit can be evaluated by adding the weights of all the 1's in the binary code

BCD 8421

- Weights are power-of-two values of each bit
- $(7)_{10} = (0111)_{BCD} = 4 + 2 + 1$

2421

- $(7)_{10} = (1101)_{2421} = 2 + 4 + 1$

8,4,-2,-1

- $(7)_{10} = (1001)_{8,4,-2,-1} = 8 - 1$

Self-complementing codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi-nations	1010	0101	0000	0001
	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

Excess-3

- Unweighted code
- Each coded combination is obtained from the corresponding binary value plus 3
- $(7)_{10} = (0111 + 0011)_{Ex-3} = (1010)_{Ex-3}$

- **Self-complementing codes**: The 9's complement of a decimal number is obtained directly by complementing each bit in the pattern.

Excess-3

- Eg.: $(7)_{10} = (1010)_{Ex-3}$ $(2)_{10} = (0101)_{Ex-3}$

2421

- Eg.: $(1)_{10} = (0001)_{2421}$ and $(8)_{10} = (1110)_{2421}$

Gray code: Definition

Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

- **Gray codes**: Each successive number differs by only one bit.

Illustration:

- $(7)_{10} = (0100)_{Gray}$
- Adjacent numbers are $(6)_{10} = (0101)_{Gray}$ and $(8)_{10} = (1100)_{Gray}$: both differs by one bit only from $(7)_{10} = (0100)_{Gray}$
- In contrast, binary code of 7 is $(0111)_{BCD}$ and 8 is $(1000)_{BCD}$: differ by 4 bits

Gray code: Application

Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

- Gray codes are used in applications in which the normal sequence of binary numbers may produce an error during the transition from one number to the next.

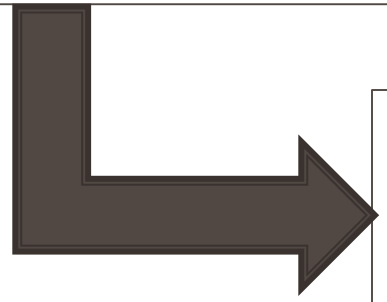
- Assume change in value is from 7 to 8:
  - **Binary code**: change required from 0111 to 1000. If right-most bit takes longer time to transition, then a wrong value (1001) can be inferred.
  - **Gray code**: change required from 0100 to 1100. Either the original or changed value can be inferred. Nothing else.

One-hot code

Decimal	Binary	Gray	One-hot
0	000	000	00000001
1	001	001	00000010
2	010	011	00000100
3	011	010	00001000
4	100	110	00010000
5	101	111	00100000
6	110	101	01000000
7	111	100	10000000

- One-hot code: only one bit is 1
- For coding N elements, N-bits are required in one-hot
- Decoding is easier: just check the bit corresponding to that element
- Used in several situations

Digital Circuits

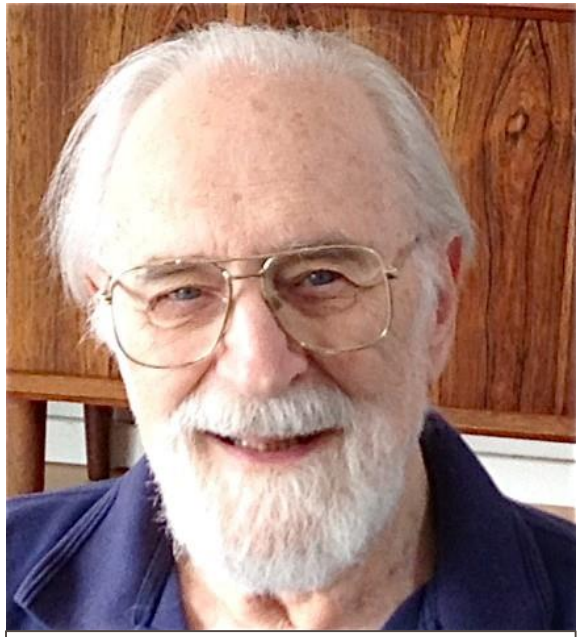


Combinational Circuit Design

Gate-level minimization

- Finding an optimal gate-level implementation of a Boolean function
- Important to understand the mathematics of gate-level minimization
- For simple circuits, solutions can be found manually
- For large number of variables (inputs) CAD logic synthesis tools are used: underlying mathematics is similar

Karnaugh Map Technique



Maurice Karnaugh

- Goal is to minimize a given logic function
- Algebraic techniques lack rules to predict each successive step
- Karnaugh Map or K-map technique is a simple and straightforward procedure for minimizing Boolean function

- K-map is a pictorial representation of the truth-table of a given function
  - ➢ Humans can easily find patterns in the K-map that provide opportunities of minimization