Lecture of Module 1

Introduction to Digital Systems

Overview

- **Introduction**
- **Digital and Analog Signals**
- **Logic Levels and Digital Waveforms**
- **Positive and Negative Logics**
- **Combinational and Sequential logics**
- **Types of Logic Devices**

Introduction

Digital electronics is a field of electronics involving the study of digital signals and the engineering of devices that use or produce them.

This is in contrast to analog electronics and analog signals.

Digital electronic circuits are usually made from large assemblies of logic gates, often packaged in integrated circuits.

Complex devices may have simple electronic representations of Boolean logic functions.

Analog versus Digital

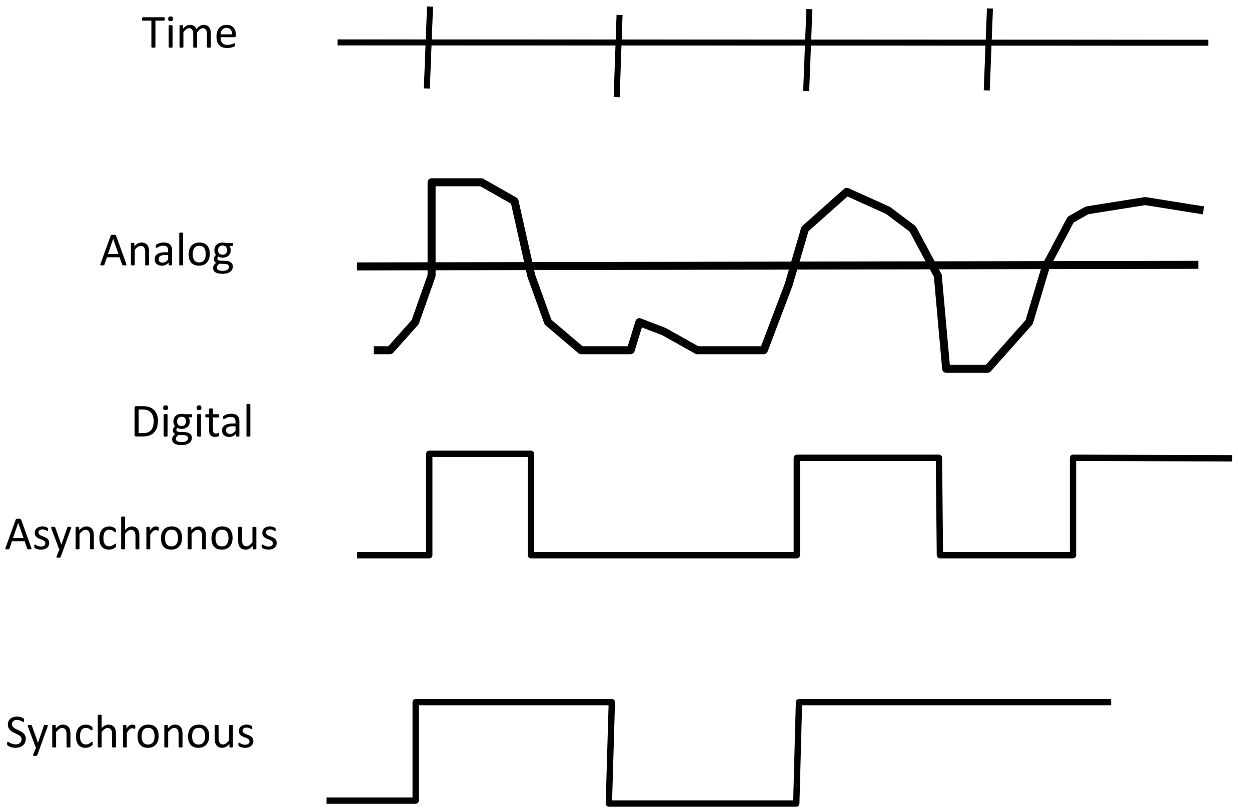
- Most observables are analog
- But the most convenient way to represent and transmit information electronically is digital
- Analog/digital and digital/analog conversion is essential

Analog Signals: The analog signals were used in many systems to produce signals to carry information. These signals are continuous in both values and time.
In short, analog signals – all signals that are natural or come naturally are analog signals.

Digital Signals: Unlike analog signals, digital signals are not continuous but signals are discrete in value and time. These signals are represented by binary numbers and consist of different voltage values.

Difference Between Analog And Digital Signal	
Analog Signals	**Digital Signals**
Continuous signals	Discrete signals
Represented by sine waves	Represented by square waves
Human voice, natural sound, analog electronic devices are few examples	Computers, optical drives, and other electronic devices
Continuous range of values	Discontinuous values
Records sound waves as they are	Converts into a binary waveform.
Only be used in analog devices.	Suited for digital electronics like computers, mobiles and more.

Signal Examples Over Time



Continuous in value & time

Discrete in value

Digital Signal

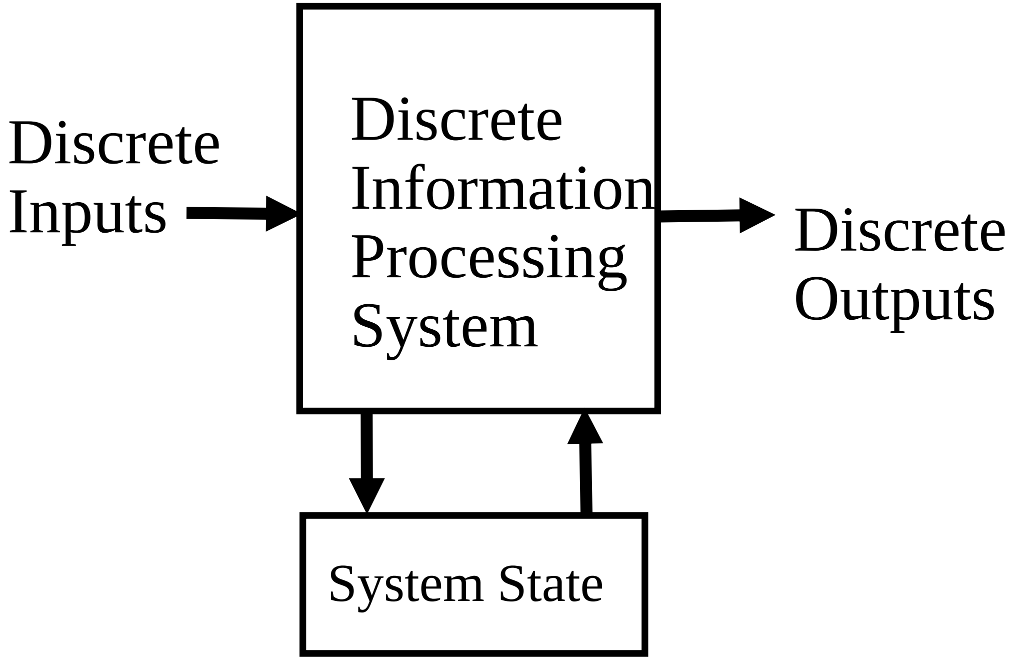
- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
- Two level, or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words On and Off.
- Binary values are represented by values or ranges of values of physical quantities

Binary Values: Other Physical Quantities

- What are other physical quantities represent 0 and 1?
  - CPU: Voltage Levels
  - Disk: Magnetic Field Direction
  - CD: Surface Pits/Light
  - Dynamic RAM: Electrical charge

Digital System

Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.



Digital representations of logical functions

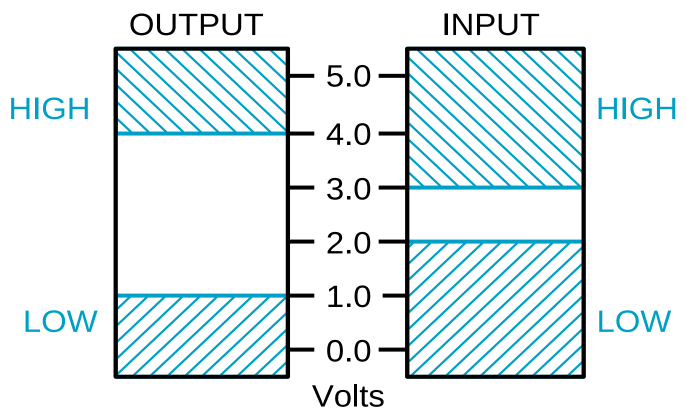
- Digital signals offer an effective way to execute logic. The formalism for performing logic with binary variables is called switching algebra or Boolean algebra.
- Digital electronics combines two important properties:
  - The ability to represent real functions by coding the information in digital form.
  - The ability to control a system by a process of manipulation and evaluation of digital variables using switching algebra.
- Digital signals can be transmitted, received, amplified, and retransmitted with no degradation.
- Binary numbers are a natural method of expressing logic variables.
- Complex logic functions are easily expressed as binary function.
- Digital information is easily and inexpensively stored

Logic Levels

In digital circuits, a **logic level** is one of a finite number of states that a digital signal can inhabit. Logic levels are usually represented by the voltage difference between the signal and ground, although other standards exist. The range of voltage levels that represent each state depends on the logic family being used.

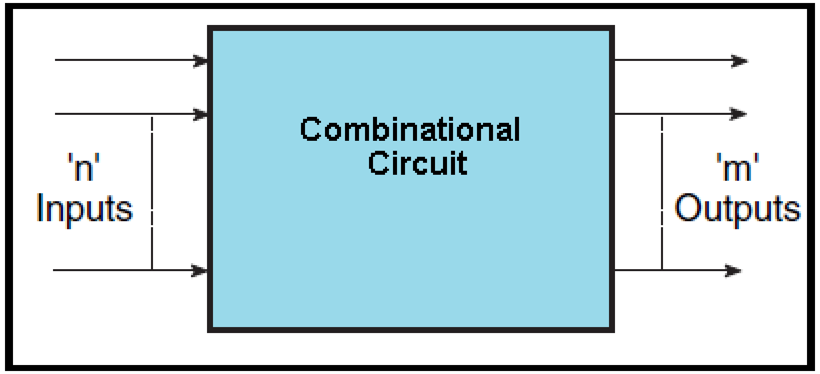
In binary logic the two levels are **logical high** and **logical low**, which generally correspond to binary numbers 1 and 0 respectively. Signals with one of these two levels can be used in Boolean algebra for digital circuit design or analysis.

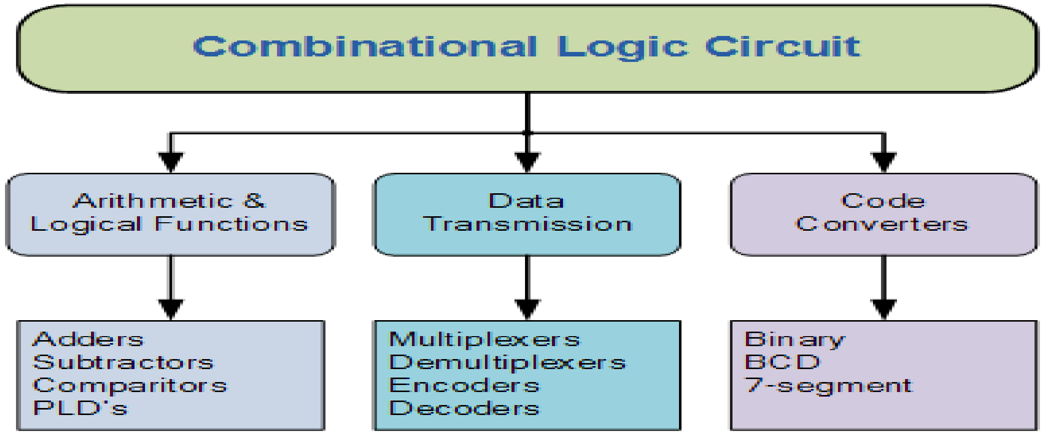
Logic level	Active-high signal	Active-low signal
Logical high	1	0
Logical low	0	1



Combinational Logic Circuit

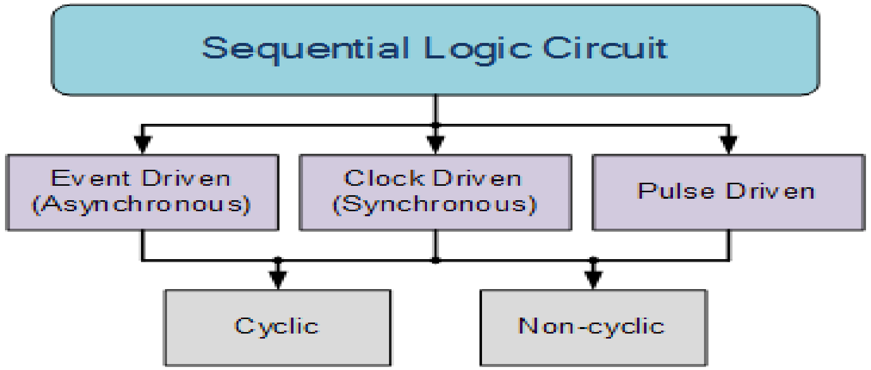
The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic “0” or logic “1”, at any given instant in time.



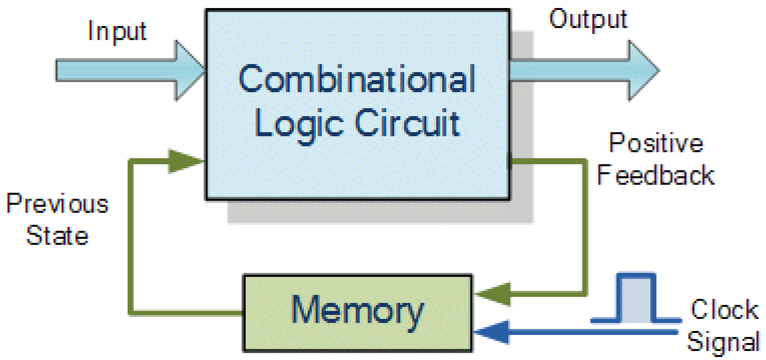


Sequential Logic Circuits

the output state of a "sequential logic circuit" is a function of the following three states, the "present input", the "past input" and/or the "past output". *Sequential Logic circuits* remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits "Memory".



Sequential logic circuits are generally termed as *two state* or Bistable devices which can have their output or outputs set in one of two basic states, a logic level "1" or a logic level "0" and will remain "latched" (hence the name latch) indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause the bistable to change its state once again.



Fixed function Logic devices

Fixed logic device such as a logic gate or a multiplexer or a flip-flop performs a given logic function that is known at the time of device manufacture

Complexity Classification for Fixed-Function ICs
SSI (Small-scale integration) – 10 gates–
MSI (Medium-scale integration) – 10—100 gates
LSI (Large-scale integration) – 100—10,000 gates
VLSI (Very large-scale integration) – 10,000—100,000 gates
ULSI (Ultra large-scale integration) -- >100,000 gates

Programmable Logic Devices

A programmable **logic device** can be configured by the user to perform a large variety of **logic functions**

A **programmable logic device** (**PLD**) is an electronic component used to build reconfigurable digital circuits

PLD has an undefined function at the time of manufacture

Before using PLD in a circuit it must be programmed (reconfigured) by using a specialized program

Purpose of PLD:

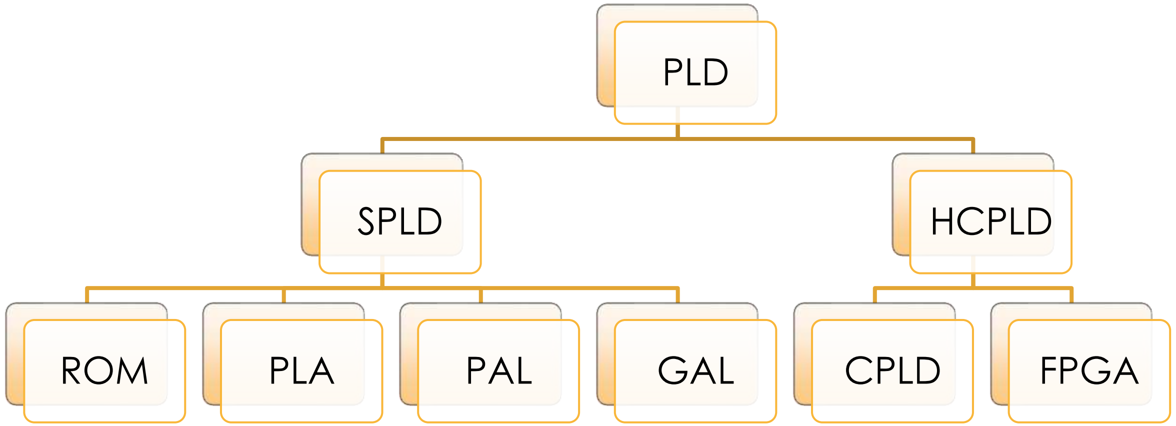
- Permits elaborate digital logic designs to be implemented by the user on a single device.
- Is capable of being erased and reprogrammed with a new design.

Advantages of PLDs

- Programmability
- Re-programmability
  - PLDs can be reprogrammed without being removed from the circuit board.
- Low cost of design
- Immediate hardware implementation
- less board space
- lower power requirements (i.e., smaller power supplies)
- Faster assembly processes
- higher reliability (fewer ICs and circuit connections => easier troubleshooting)
- availability of design software

Types of PLDs

- SPLDs (Simple Programmable Logic Devices)
  - ROM (Read-Only Memory)
  - PLA (Programmable Logic Array)
  - PAL (Programmable Array Logic)
  - GAL (Generic Array Logic)
- HCPLD (High Capacity Programmable Logic Device)
  - CPLD (Complex Programmable Logic Device)
  - FPGA (Field-Programmable Gate Array)

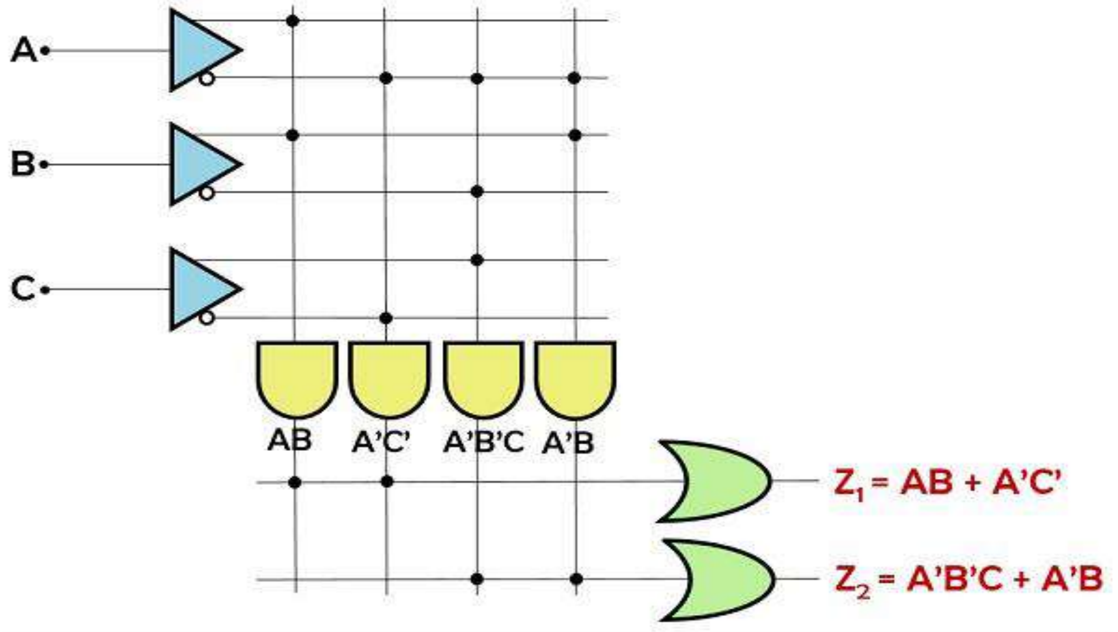


PLD Configuration

- Combination of a logic device and memory
- Memory stores the pattern the PLD was programmed with
  - EPROM
    - Non-volatile and reprogrammable
  - EEPROM
    - Non-volatile and reprogrammable
  - Static RAM (SRAM)
    - Volatile memory
  - Flash memory
    - Non-volatile memory
  - Antifuse
    - Non-volatile and no re-programmability

PLA

PLA: A programmable logic array (PLA) has a programmable AND gate array, which links to a programmable OR gate array

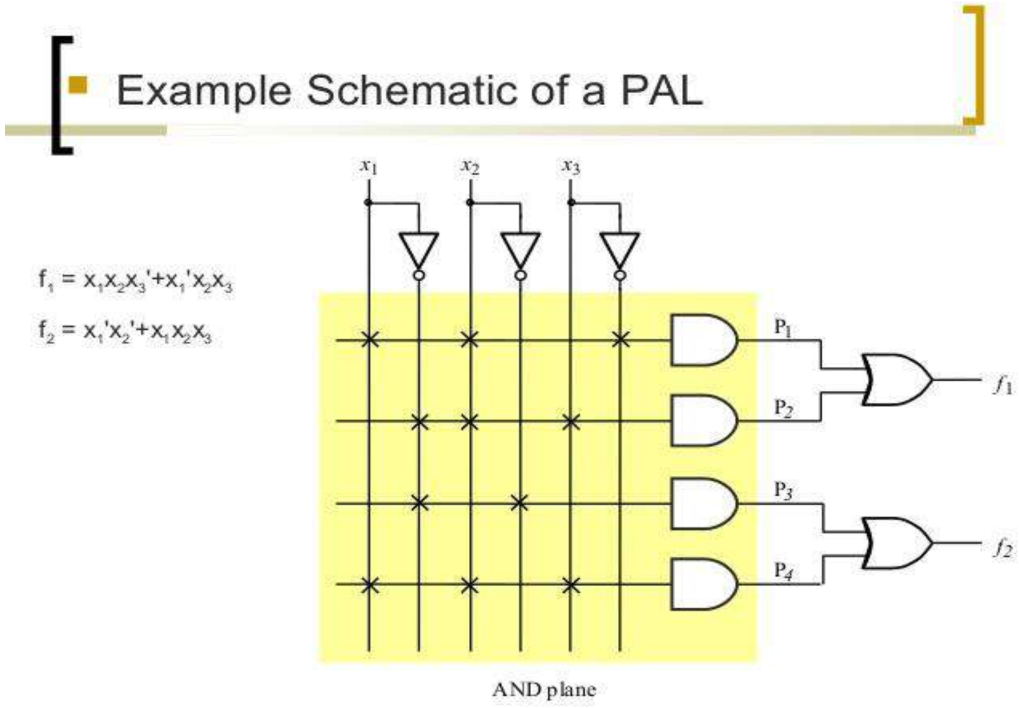


Implementation of Programmable Logic Array

Electronics Coach

PAL

PAL: PAL devices have arrays of transistor cells arranged in a "fixed-OR, programmable-AND" plane



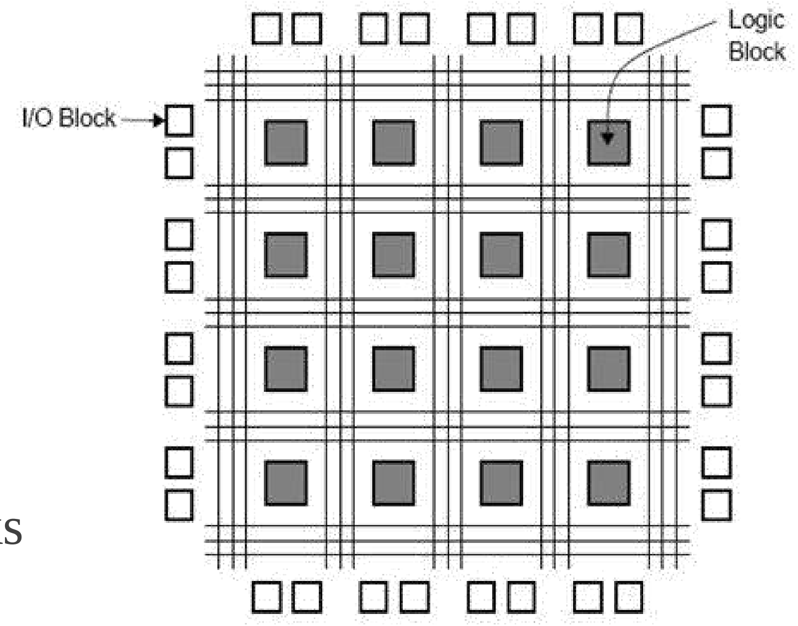
GAL

GAL: An improvement on the PAL was the Generic Array Logic device

- This device has the same logical properties as the PAL but can be erased and reprogrammed
- The GAL is very useful in the prototyping stage of a design, when any bugs in the logic can be corrected by reprogramming
- GALs are programmed and reprogrammed using a PAL programmer

HCPLD

- CPLD (Complex Programmable Logic Device)
  - Lies between PALs and FPGAs in degree of complexity.
  - Inexpensive
- FPGA (Field-Programmable Gate Array)
  - Truly parallel design and operation
  - Fast turnaround design
  - Array of logic cells surrounded by programmable I/O blocks



FPGA

Number Systems

Overview

- **Introduction**
- **Number Systems** [binary, octal and hexadecimal]
- **Number System conversions**

Introduction

Number System

Code using symbols that refer to a number of items

Decimal Number System

Uses ten symbols (base 10 system)

Binary System

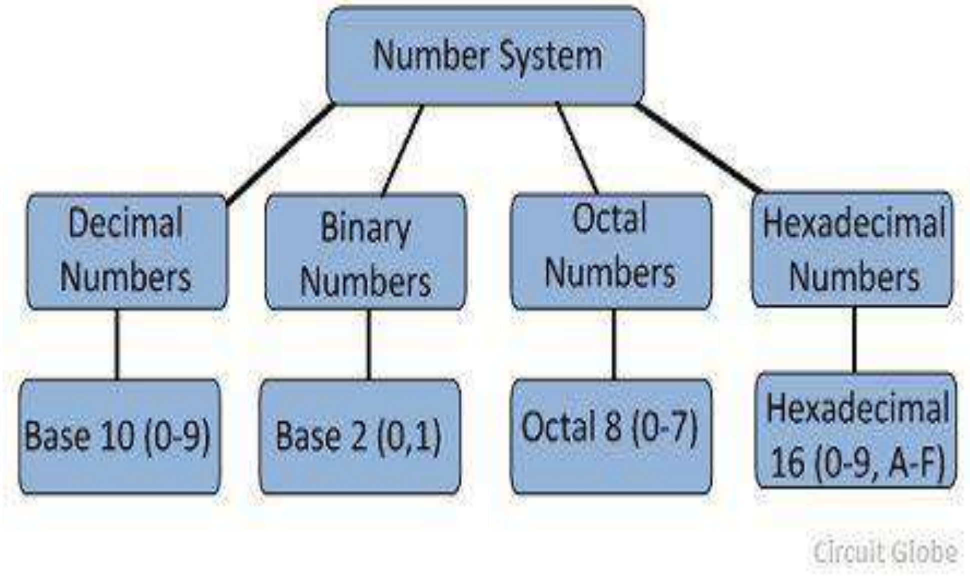
Uses two symbols (base 2 system)

Octal Number System

Uses eight symbols (base 8 system)

Hexadecimal Number System

Uses sixteen symbols (base 16 system)



Binary Number

- **Numeric value of symbols in different positions.**
- ***Example* - Place value in binary system:**

Place Value	8s	4s	2s	1s
Binary	Yes	Yes	No	No
Number	1	1	0	0

RESULT: Binary 1100 = decimal 8 + 4 + 0 + 0 = decimal 12

BINARY TO DECIMAL CONVERSION

Convert Binary Number 110011 to a Decimal Number:

Binary	1	1	0	0	1	1
	↓	↓	↓	↓	↓	↓

Decimal $32 + 16 + 0 + 0 + 2 + 1 = 51$

TEST

Convert the following binary numbers into decimal numbers:

Binary 1001 =

Binary 1111 =

Binary 0010 =

TEST

Convert the following binary numbers into decimal numbers:

Binary 1001 = 9

Binary 1111 = 15

Binary 0010 = 2

DECIMAL TO BINARY CONVERSION

Divide by 2 Process

Decimal # 13 ÷ 2 = 6 remainder 1

6 ÷ 2 = 3 remainder 0

3 ÷ 2 = 1 remainder 1

1 ÷ 2 = 0 remainder 1

1 1 0 1

TEST

Convert the following decimal numbers into binary:

Decimal 11 =

Decimal 4 =

Decimal 17 =

TEST

Convert the following decimal numbers into binary:

Decimal 11 = 1011

Decimal 4 = 0100

Decimal 17 = 10001

Binary-to-Decimal Conversion

- . 1 0 1 (base-2)

$1 \times 2^{-1} = 1 \times 0.5 = 0.5$

$0 \times 2^{-2} = 0 \times 0.25 = 0$

$1 \times 2^{-3} = 1 \times 0.125 = 0.125$

$0.5 + 0 + 0.125 = 0.625$

$0.101_2 = 0.625_{10}$

Converting Decimal Fraction to Binary

- Convert $N = 0.6875$ to Radix 2
- Solution: Multiply N by 2 repeatedly & collect integer bits

Multiplication	New Fraction	Bit
$0.6875 \times 2 = 1.375$	0.375	1
$0.375 \times 2 = 0.75$	0.75	0
$0.75 \times 2 = 1.5$	0.5	1
$0.5 \times 2 = 1.0$	0.0	1

First fraction bit

Last fraction bit

- Stop when new fraction = 0.0, or when enough fraction bits are obtained
- Therefore, $N = 0.6875 = (0.1011)_2$
- Check $(0.1011)_2 = 2^{-1} + 2^{-3} + 2^{-4} = 0.6875$

HEXADECIMAL NUMBER SYSTEM

Uses 16 symbols -Base 16 System, 0-9, A, B, C, D, E, F

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10

HEXADECIMAL AND BINARY CONVERSIONS

- Hexadecimal to Binary Conversion

Hexadecimal	C	3
	↓	↓
Binary	1100	0011

- Binary to Hexadecimal Conversion

Binary	1110	1010
	↓	↓
Hexadecimal	E	A

DECIMAL TO HEXADECIMAL CONVERSION

Divide by 16 Process

Decimal #   47 ÷ 16 = 2 remainder 15

2 ÷ 16 = 0 remainder 2

2 F

Decimal Fraction To Hex

- To convert Decimal fraction into Hex, multiply fractional part with 16 till you get fractional part 0.
- Example : convert 0.03125_{10} to Hex

	Integer Part	
$0.03125 * 16 = 0.5$	0	Write From Up to Down
$0.5 * 16 = 8.0$	8	

➔ $0.03125_{10} = 0.08_{16}$

HEXADECIMAL TO DECIMAL CONVERSION

Convert hexadecimal number 2DB to a decimal number

Place Value	256s	16s	1s
Hexadecimal	2	D	B
	(256 x 2)	(16 x 13)	(1 x 11)

Decimal 512 + 208 + 11 = $\boxed{731}$

Hexadecimal System

The weight associated with each symbol in the given hexadecimal number can be determined by raising 16 to a power equivalent to the position of the digit in the number.

Example 4A90.2BC

Digit	4	A	9	0	.	2	B	C
Weight	16^3	16^2	16^1	16^0	Hexadecimal Point	16^{-1}	16^{-2}	16^{-3}

Example

The following shows that the number (2AE)16 in hexadecimal is equivalent to 686 in decimal.

		16^2		16^1		16^0	Place values
		2		A		E	Number
N	=	2×16^2	+	10×16^1	+	14×16^0	Values

The equivalent decimal number is N = 512 + 160 + 14 = 686.

TEST

Convert Hexadecimal number A6 to Binary

A6 = 1010 0110 (Binary)

Convert Hexadecimal number 16 to Decimal

16 = 22 (Decimal)

Convert Decimal 63 to Hexadecimal

63 = 3F (Hexadecimal)

- Translate every hexadecimal digit into its 4-bit binary equivalent
- Examples:

$(3A5)_{16} = (0011\ 1010\ 0101)_2$

$(12.3D)_{16} = (0001\ 0010\ .\ 0011\ 1101)_2$

$(1.8)_{16} = (0001\ .\ 1000)_2$

OCTAL NUMBERS

Uses 8 symbols -Base 8 System
0, 1, 2, 3, 4, 5, 6, 7

Decimal	Binary	Octal
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7
8	001 000	10
9	001 001	11

OCTAL AND BINARY CONVERSIONS

- Octal to Binary Conversion

Octal	5	6
	↓	↓
Binary	101	110

- Binary to Octal Conversion

Binary	100	101
	↓	↓
Octal	4	5

DECIMAL TO OCTAL CONVERSION

Divide by 8 Process

Decimal # 129 ÷ 8 = 16 remainder 1

16 ÷ 8 = 2 remainder 0

2 ÷ 8 = 0 remainder 2

$\boxed{2\ 0\ 1}$

Fraction Decimal to Octal Conversion - Example

- Example: convert 0.356_{10} to octal.

$0.356 * 8 = 2.848$ → integer part = 2
$0.848 * 8 = 6.784$ → integer part = 6
$0.784 * 8 = 6.272$ → integer part = 6
$0.272 * 8 = 2.176$ → integer part = 2
$0.176 * 8 = 1.408$ → integer part = 1
$0.408 * 8 = 3.264$ → integer part = 3, etc.

Answer = $0.266213\ldots_8$

OCTAL TO DECIMAL CONVERSION

Convert octal number 201 to a decimal number

Place Value	64s	8s	1s
Octal	2	0	1
	(64 x 2)	(8 x 0)	(1 x 1)
Decimal	128 +	0 +	1 = **129**

- **Octal fraction to decimal**
- **Example**
- Convert (23.25)8 to decimal
- 8^1 8^0 . 8^{-1} 8^{-2}

  2 3 2 5

$= (2 \times 8^1) + (3 \times 8^0) + (2 \times 8^{-1}) + (5 \times 8^{-2})$

= 16+3+0.25+0.07812

= (19.32812)10

Binary, Octal, and Hexadecimal

- Binary, Octal, and Hexadecimal are related:

  Radix 16 = 2^4 and Radix 8 = 2^3

- Hexadecimal digit = 4 bits and Octal digit = 3 bits

- Starting from least-significant bit, group each 4 bits into a hex digit or each 3 bits into an octal digit

- Example: Convert 32-bit number into octal and hex

3	5	3	0	5	5	2	3	6	2	4	Octal

1	1	1	0	1	0	1	1	0	0	0	1	0	1	1	0	1	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	32-bit binary

E	B	1	6	A	7	9	4	Hexadecimal

Convert 0.10111_2 to base 8: **0.101_110 = 0.56_8**

Convert 0.1110101 to base 16: **0.1110_1010 = $0.EA_{16}$**

Arithmetic Operations

Overview

- **Arithmetic Operations**
- **Decimal Arithmetic**
- **Binary Arithmetic**
- **Signed Binary Numbers**

Arithmetic Operations

Addition

- Follow same rules as in decimal addition, with the difference that when sum is 2 indicates a carry (not a 10)
- Learn new carry rules
  - 0+0 = sum 0 carry 0
  - 0+1 = 1+0 = sum 1carry 0
  - 1+1 = sum 0 carry1
  - 1+1+1 = sum 1carry1

Carry	1	1	1	1	1	0
Augend	0	0	1	0	0	1
Addend	0	1	1	1	1	1
Result	1	0	1	0	0	0

```
  1 1 1        <——— Carry Values
  0 1 0 1
+ 1 0 1 1
---------
1 0 0 0 0
```

Subtraction

- Learn new borrow rules
  - 0-0 = 1-1 = 0 borrow 0
  - 1-0 = 1 borrow 0
  - 0-1 = 1 borrow 1

Borrow	1	1	0	0	
Minuend	1	1	0	1	1
Subtrahend	0	1	1	0	1
Result	0	1	1	1	0

The rules of the decimal base applies to binary as well. To be able to calculate 0-1, we have to "borrow one" from the next left digit.

```
   1 2
 0 2 0 2
 1 0 1 0
-0 1 1 1
--------
 0 0 1 1
```

Decimal Subtraction

- 9’s Complement Method
- 10’s Complement Method

9’s Complement Method

Example: 72532 – 3250

9’s complement of 3250 is

9 9 9 9 9 – 0 3 2 5 0 = 9 6 7 4 9

If Carry, result is positive.
Add carry to the partial result

```
   7 2 5 3 2
 + 9 6 7 4 9
 -----------
 1 6 9 2 8 1
          +1
 -----------
   6 9 2 8 2
```

Example: 3250 – 72532

9’s complement of 72532 is

9 9 9 9 9 – 7 2 5 3 2 = 2 7 4 6 7

If no Carry, result is negative.
Magnitude is 9’s complement of the result

```
    0 3 2 5 0
  + 2 7 4 6 7
  -----------
    3 0 7 1 7
 = – 6 9 2 8 2
```

Decimal Subtraction

- 9's Complement Method
- 10's Complement Method

10's Complement Method

Example: 72532 – 3250

10's complement of 3250 is

1 0 0 0 0 0 – 0 3 2 5 0 = 9 6 7 5 0

```
   7 2 5 3 2
 + 9 6 7 5 0
 -----------
 1 6 9 2 8 2
```

Result is 6 9 2 8 2

If Carry, result is positive.
Discard the carry

Example: 3250 – 72532

10's complement of 72532 is

1 0 0 0 0 0 – 7 2 5 3 2 = 2 7 4 6 8

```
   0 3 2 5 0
 + 2 7 4 6 8
 -----------
   3 0 7 1 8
 = – 6 9 2 8 2
```

If no Carry, result is negative.
Magnitude is 10's complement of the result

Binary Subtraction

- 1's Complement Method
- 2's Complement Method

1's Complement Method

Example: 1010100 – 1000100

1's complement of 1000100 is 0111011

If Carry, result is positive.
Add carry to the partial result

$$
\begin{array}{r}
1\,0\,1\,0\,1\,0\,0 \\
+\;0\,1\,1\,1\,0\,1\,1 \\
\hline
\mathbf{1}\;\;0\,0\,0\,1\,1\,1\,1 \\
+\mathbf{1} \\
\hline
0\,0\,1\,0\,0\,0\,0
\end{array}
$$

Example: 1000100 – 1010100

1's complement of 1010100 is 0101011

If no Carry, result is negative.
Magnitude is 1's complement of the result

$$
\begin{array}{r}
1\,0\,0\,0\,1\,0\,0 \\
+\;0\,1\,0\,1\,0\,1\,1 \\
\hline
1\,1\,0\,1\,1\,1\,1 \\
=\;-\,0\,0\,1\,0\,0\,0\,0
\end{array}
$$

Binary Subtraction

- 1’s Complement Method
- 2’s Complement Method

2’s Complement Method

```
  1 0 1 0 1 0 0
+ 0 1 1 1 1 0 0
---------------
1 0 0 1 0 0 0 0
```

Example: 1010100 – 1000100

2’s complement of 1000100 is 0111100

If Carry, result is positive.
Discard the carry

0 0 1 0 0 0 0

Example: 1000100 – 1010100

2’s complement of 1010100 is 0101100

```
  1 0 0 0 1 0 0
+ 0 1 0 1 1 0 0
---------------
  1 1 1 0 0 0 0
= – 0 0 1 0 0 0 0
```

If no Carry, result is negative.
Magnitude is 2’s complement of the result

Signed Binary Numbers

- When a signed binary number is positive
  - The MSB is ‘0’ which is the sign bit and rest bits represents the magnitude

- When a signed binary number is negative
  - The MSB is ‘1’ which is the sign bit and rest of the bits may be represented by three different ways
    - Signed magnitude representation
    - Signed 1’s complement representation
    - Signed 2’s complement representation

Signed Binary Numbers

	- 9	**+ 9**
Signed magnitude representation	1 1001	0 1001
Signed 1’s complement representation	1 0110	0 1001
Signed 2’s complement representation	1 0111	0 1001

	- 0	**+ 0**
Signed magnitude representation	1 0000	0 0000
Signed 1’s complement representation	1 1111	0 0000
Signed 2’s complement representation	-None-	0 0000

Range of Binary Number

Binary Number of n bits

- General binary number: ($2^n - 1$)
- Signed magnitude binary number: – ($2^{n-1} - 1$) to + ($2^{n-1} - 1$)
- Signed 1's complement binary number: – ($2^{n-1} - 1$) to + ($2^{n-1} - 1$)
- Signed 2's complement binary number: – (2^{n-1}) to + ($2^{n-1} - 1$)

Signed Binary Number Arithmetic

- Add or Subtract two signed binary number including its sign bit either signed 1's complement method or signed 2's complement method
- The 1's complement and 2's complement rules of general binary number is applicable to this

- It is important to decide how many bits we will use to represent the number
- Example: Representing +5 and -5 on 8 bits:
  - +5: 00000101
  - -5: 10000101
- *So the very first step we have to decide on the number of bits to represent number*

Digital Codes

Overview

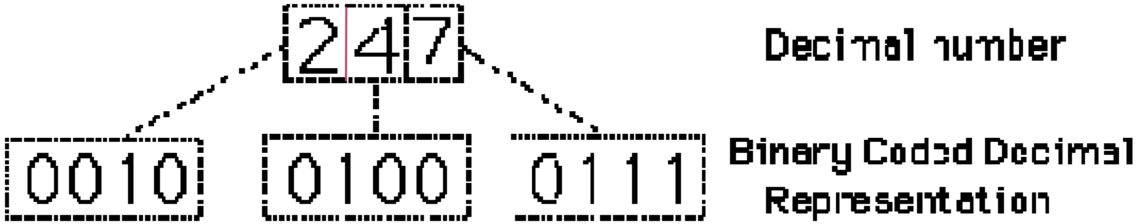
- **Introduction**
- **Binary Coded Decimal Code**
- **EBCDIC Code**
- **Excess-3 Code**
- **Gray Code**
- **ASCII Code**

Introduction

- Calculations or computations are not useful until their results can be displayed in a manner that is meaningful to people.
- We also need to store the results of calculations, and provide a means for data input.
- Thus, human-understandable characters must be converted to computer-understandable bit patterns using some sort of character encoding scheme.
- As computers have evolved, character codes have evolved.
- Larger computer memories and storage devices permit richer character codes.
- The earliest computer coding systems used six bits.
- Binary-coded decimal (BCD) was one of these early codes. It was used by IBM mainframes in the 1950s and 1960s.

- In 1964, BCD was extended to an 8-bit code, Extended Binary-Coded Decimal Interchange Code (EBCDIC).
- EBCDIC was one of the first widely-used computer codes that supported upper *and* lowercase alphabetic characters, in addition to special characters, such as punctuation and control characters.
- EBCDIC and BCD are still in use by IBM mainframes today.
- Other computer manufacturers chose the 7-bit ASCII (American Standard Code for Information Interchange) as a replacement for 6-bit BCD codes.
- While BCD and EBCDIC were based upon punched card codes, ASCII was based upon telecommunications (Telex) codes.
- Until recently, ASCII was the dominant character code outside the IBM mainframe world.

Binary Coded Decimal (BCD)



Convert 0100 0010 1000 0010 BCD to decimal

0100	0010	1000	0110	BCD
4	**2**	**8**	**6**	Decimal

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

- Consider 5 + 5
- 5 0 1 0 1
- +5 0 1 0 1
- giving 1 0 1 0 which is binary 10 but not a BCD digit!
- What to do?
- Try adding 6??

- Had 1010 and want to add 6 or 0110
  - so 1 0 1 0
  - plus 6 0 1 1 0
  - Giving 1 0 0 0 0

- Add 7 + 6
  - have 7 0 1 1 1
  - plus 6 <u>0 1 1 0</u>
  - Giving 1 1 0 1 and again out of range
  - Adding 6 <u>0 1 1 0</u>
  - Giving 1 0 0 1 1 so a 1 carries out to the next BCD digit
  - FINAL BCD answer 0001 0011 or 13_{10}

6	0110	BCD for 6	42	0100 0010	BCD for 42
+3	0011	BCD for 3	+27	0010 0111	BCD for 27
9	1001	BCD for 9	69	0110 1001	BCD for 69

- Add the BCD for 417 to 195
- Would expect to get 612
  - BCD setup - start with Least Significant Digit
  - 0 1 0 0 0 0 0 1 0 1 1 1
  - <u>0 0 0 1</u> <u>1 0 0 1</u> <u>0 1 0 1</u>
  - 1 1 0 0
  - Adding 6 <u>0 1 1 0</u>
  - Gives 1 0 0 1 0

- Had a carry to the 2nd BCD digit position
  - 1
  - 0 1 0 0 0 0 0 1 done
  - <u>0 0 0 1</u> <u>1 0 0 1</u> 0 0 1 0
  - 1 0 1 1
  - Again must add 6 <u>0 1 1 0</u>
  - Giving 1 0 0 0 1
  - And another carry

- Had a carry to the 3rd BCD digit position
  - 1
  - 0 1 0 0 done done
  - <u>0 0 0 1</u> 0 0 0 1 0 0 1 0
  - 0 1 1 0
  - And answer is 0110 0001 0010 or the BCD for the base 10 number 612

EBCDIC Code

- The EBCDIC code is an 8-bit alphanumeric code that was developed by IBM to represent alphabets, decimal digits and special characters, including control characters.
- The EBCDIC codes are generally the decimal and the hexadecimal representation of different characters.
- This code is rarely used by non IBM-compatible computer systems.

The Excess-3- Code

DECIMAL	BCD	EXCESS-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

(a) 13 **(b) 430**

	1	3		(b)	4	3	0	
+	3	3			3	3	3	
	4	6			7	6	3	
	0100	0110	Excess-3		0111	0110	0011	Excess-3

- Excess-3 code is self complementary code? Justify.

Gray Code

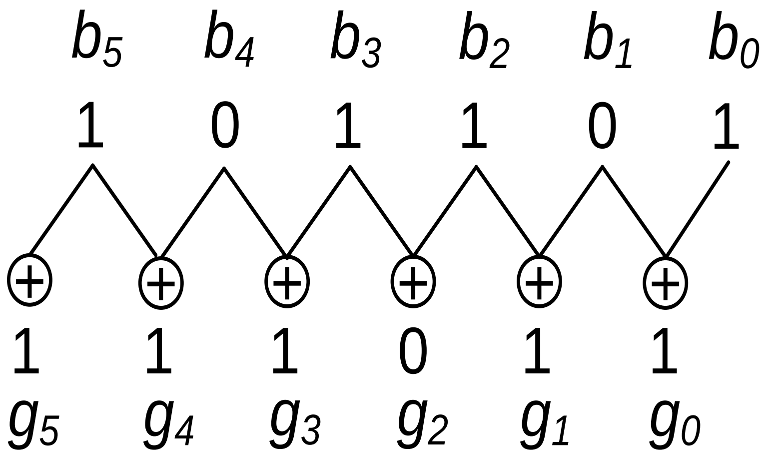
- Gray code is another important code that is also used to convert the decimal number into 8-bit binary sequence. However, this conversion is carried in a manner that the contiguous digits of the decimal number differ from each other by one bit only
- In pure binary coding or 8421 BCD then counting from 7 (0111) to 8 (1000) requires 4 bits to be changed simultaneously
- Gray coding avoids this since only one bit changes between subsequent numbers

Binary to Gray

Example:

Binary:

Gray:



$$g_5 = b_5$$
$$g_4 = b_5 \oplus b_4$$
$$g_3 = b_4 \oplus b_3$$
$$g_2 = b_3 \oplus b_2$$
$$g_1 = b_2 \oplus b_1$$
$$g_0 = b_1 \oplus b_0$$

Gray to Binary

Decimal number	*Gray* g_3	g_2	g_1	g_0	*Binary* b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

$b_5 = g_5$
$b_4 = g_5 \oplus g_4$
$b_3 = g_5 \oplus g_4 \oplus g_3$
$b_2 = g_5 \oplus g_4 \oplus g_3 \oplus g_2$
$b_1 = g_5 \oplus g_4 \oplus g_3 \oplus g_2 \oplus g_1$
$b_0 = g_5 \oplus g_4 \oplus g_3 \oplus g_2 \oplus g_1 \oplus g_0$

Reflection of Gray Codes

00	0	00	0	000
01	0	01	0	001
11	0	11	0	011
10	0	10	0	010
	1	10	0	110
	1	11	0	111
	1	01	0	101
	1	00	0	100
			1	100
			1	101
			1	111
			1	110
			1	010
			1	011
			1	001
			1	000

So, called reflected code

Alphanumeric Codes

- How do you handle alphanumeric data?
- Easy answer!
- Formulate a binary code to represent characters! ☺
- For the 26 letter of the alphabet would need 5 bit for representation.
- But what about the upper case and lower case, and the digits, and special characters

ASCII

- ASCII stands for American Standard Code for Information Interchange
- The code uses 7 bits to encode 128 unique characters
- Formally, work to create this code began in 1960. 1st standard in 1963. Last updated in 1986
- Represents the numbers
  - All start 011 xxxx and the xxxx is the BCD for the digit
- Represent the characters of the alphabet
  - Start with either 100, 101, 110, or 111
  - A few special characters are in this area
- Start with 010 – space and !”#$%&’()*+.-,/
- Start with 000 or 001 – control char like ESC

Table 1.7
American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	–	o	DEL

Control characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

ASCII Properties

ASCII has some interesting properties:

- Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}
- Upper case A - Z span 41_{16} to $5A_{16}$
- Lower case a - z span 61_{16} to $7A_{16}$
  - Lower to upper case translation (and vice versa) occurs by flipping bit 6.
- Delete (DEL) is all bits set, a carryover from when punched paper tape was used to store messages.
- Punching all holes in a row erased a mistake!