Unit 4 Combinational Circuits

Structure:

4.1 Introduction

In the last unit, we studied about Boolean expressions and functions and simplification of Boolean expressions using Karnaugh map and Quine-McCluskey Methods. The digital system which drives the modern computer can be constructed using the basic gates i.e., AND, OR, and NOT gates. There are two types of digital circuit: Combinational circuits and Sequential circuits. Circuits whose outputs depend only on the current inputs are known as Combinational circuits; hence the outputs will be generated by combining the inputs according to the functionality. Circuits whose outputs depend on both past and current inputs are known as Sequential circuits. Outputs of sequential circuits can be determined by using the sequence of inputs over the time. Memory circuits are considered as sequential circuits and adders which is constructed using gates is considered as combinational circuits. In this unit we will study about combinational circuits, and different codes like Gray code, BCD code, and Excess-3 code.

Objectives:

By the end of Unit 4, the learners are able to:

- explain different types of combinational circuits.
- explain Gray code and its conversion
- explain BCD code
- explain Excess-3 code

4.2 Definition of Combinational Circuits

Combinational circuits are those whose outputs depend only on the current inputs. Here digital circuits generate a set of outputs from set of inputs using Boolean operations. Usually many Boolean functions are implemented using combinational circuit and each Boolean function corresponds to a particular output. In a combinational circuit, it is important to observe that each Boolean function implemented is represented by an output.

Combinational circuits are used to construct the computer's Central Processing Unit (CPU). For example, a set of Boolean functions can be used to implement an adder circuit. For example, if there are two one bit numbers X and Y. By using Boolean functions, SUM and CARRY values can be generated.

SUM (S) = XY'+X'Y= X (XOR) Y.
CARRY (C) = XY = X (AND) Y.

A half-adder can be implemented by using above Boolean functions. As the adder can add only two input bits but carry from the previous stage cannot be added, engineers call is as half adder.

The circuit which adds the carry bit from previous stage with two one bit inputs is known as Full adder. Full adder accepts three one bit inputs and generates SUM and CARRY outputs. The SUM and CARRY Boolean functions for a full adder is given below.

$S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$ = A (XOR) B (XOR) C
$C_{out} = AB + AC_{in} + BC_{in}$

Using 1 bit adders, an n bit adder can be constructed easily by combining many 1 bit adder circuits. The figure 4.1 shows building an N-Bit Adder Using Half and Full Adders.

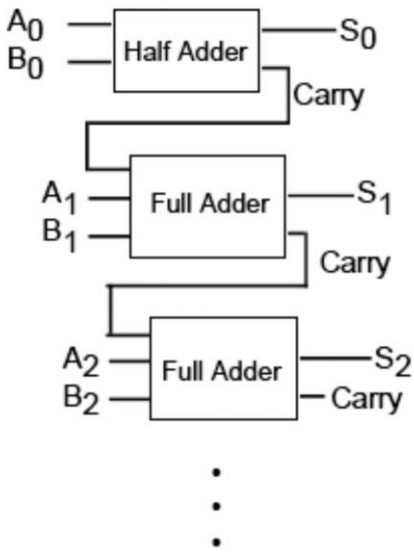


Figure 4.1: Building an N-Bit Adder Using Half and Full Adders

From above discussion we can clearly say that Boolean operations and arithmetic can be implemented using Boolean logic functions.

BCD to Seven segment decoder is the combination circuit which is used very commonly. Using this circuit, we can determine which Light Emitting Diode (LED) on seven segment should be displayed. As there are seven outputs, there will be seven Boolean logic functions (segment 0 through segment 6). Note that light-emitting diode (LED) is a two-lead semiconductor light source that resembles a basic pn-junction diode, except that an LED also emits light.

The four inputs to each of these seven Boolean functions are the four bits from a binary number in the range 0...9. Let D be the Higher Order bit of this number and A be the Lower Order bit of this number. Each logic function should produce a one (segment on) for a given input if that particular segment should be illuminated.

Counting as we have been taught since kindergarten is based on the decimal number system. *Decimal* means base 10 (the prefix *dec*). In any number system, given the base (often referred to as *radix*), the number of digits that can be used to count is fixed. For example in the base 10 number system, the digits that can be used to count are 0,1,2,3,4,5,6,7,8,9.

Generalizing that for any base b, the first b digits (starting with 0) represent the digits that are used to count. When a number ≥ b has to be represented, the *place values* are used.

4.3 Types of Combinational Circuits

4.3.1 Adders

There are two types of single bit adders. They are Half adder and Full adder.

A half adder circuits will generate sum (S) and carry (C) from the two of its inputs A and B. Sum S can be generated by XORing inputs A and B and Carry C is generated by ANDing A and B . The output of the half adder will be of 2 bits. The sum is indicated by least significant bit and carry is indicated by most significant bit.

A full adder circuit adds the carry in C along with the two inputs A and B. A large adder circuit can be constructed using multiple full adders. In full adder circuit, carry in is labeled as C_i or C_{in} and carry out is labeled as C_o or C_{out}, labeling this way will remove the ambiguity between the output carry and the input carry..

Half adder

A half adder is a digital circuit which accepts two inputs and performs addition on them and generates two binary digits known as sum(S) and carry(C).

As the carry is not included in the addition, this circuit cannot be used to perform addition of multibit numbers.

The table 4.1 shows the truth table for a half adder:

Table 4.1: Truth Table for Half adder

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The figure 4.2 shows the logic circuit diagram for Half adder.

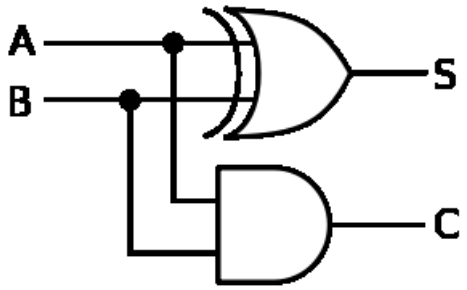


Figure 4.2: Circuit diagram for Half adder

Full adder: The figure 4.3 shows the the logic circuit diagram for Full adder

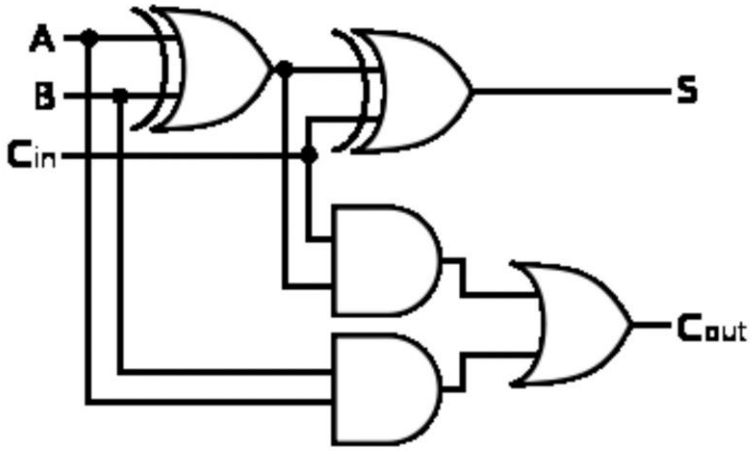


Figure 4.3: Circuit diagram for Half adder

Inputs: {A, B, Cin} Outputs: {Sum, Cout}

The figure 4.4 shows the schematic symbol for a 1-bit full adder.

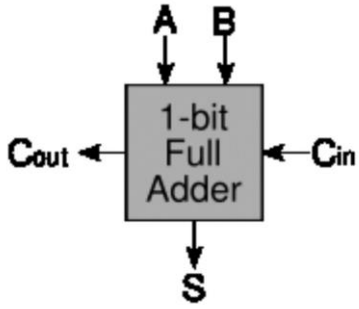


Figure 4.4: Schematic symbol for a 1-bit full adder

$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)) = (A \cdot B) + (C_{in} \cdot B) + (C_{in} \cdot A)$$

A full adder circuit adds the carry in C_{in} (or Ci) along with the two inputs A and B. The full adder performs addition on 3 input bits and generates carry and sum binary outputs. Multiple full adders can be used to perform the addition of multibit (binary) inputs. The table 4.2 shows the truth table for Full adder.

Table 4.2: Truth table for Full adder

Input			Output	
A	B	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Two half adder circuits can be used to implement a full adder circuit, the two inputs A and B are connected to one half adder and its output sum is connected as input to the second half adder. Carry in will be the other input to the second half adder and two carry outputs are ORed. The Boolean function for S can be expressed as XOR of three inputs A, B, Ci and a majority function of A, B, and Ci can be Carry out equation. Sum of three one bit numbers will be the output of the full adder circuit.

4.3.2 Subtractors

The approach used for designing an adder can be used to design a subtractor. Following is the summary of the process used for subtracting binary numbers. For multi-bit numbers during subtraction of each bit, following three bits will be involved: the subtrahend (Y_i), minuend (X_i), and a borrow in from the previous bit order position (Bi). The difference bit (D_i) and borrow bit Bi+1 are the outputs of the subtractor. Following is the equation for the difference of two bits.

$$D_i = X_i \oplus Y_i \oplus B_i$$

4.3.3 Comparators

A digital comparator is a hardware electronic device that compares two numbers in binary form and generates a one or a zero at its output depending on whether they are the same (i.e. equal) or not.

Comparators can be used in a central processing unit (CPU) or microcontroller in branching software. A comparator can be simulated by subtracting the two values (A & B) in question and checking if the result is zero. This works because if A = B then A - B = 0.

Many microcontrollers have analog comparators on some of their inputs that can be read or trigger an interrupt.

The operation of a single bit digital comparator can be expressed as truth table shown in the table 4.3.

Table 4.3: Truth table for Single Bit Comparator

Inputs		Outputs		
A	B	A < B	A = B	A > B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

The operation of a two bit digital comparator can be expressed as a truth table shown in the table 4.4.

Table 4.4: Truth table for Two Bit Comparator

Inputs				Outputs		
A1	A0	B1	B0	A < B	A = B	A > B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Examples of the comparator include the CMOS 4063 and 4585 and the TTL 7485 and 74682-'89.

4.3.4 Decoders

A decoder is a digital device which decodes the original information from the encoded inputs. The functionality of a decoder is exactly the reverse of an encoder. In order to decode the information from the signals, the method used to encode the information is reversed.

In digital systems, decoder is a logic circuit with multiple inputs and multiple outputs that produces a coded output from the coded inputs. In decoder the input and output codes are different i.e., if there is an n-bit input code then decoder produces 2^n output code. So a decoder is a combinational circuit that detects the binary information present on n input lines and decodes it and indicates this decoded output on any one of the 2^n output lines.

The outputs of a decoder without enable input are assumed as single “disabled” output code word. In digital circuits, decoders are used in many applications like 7 segment display, multiplexing of the data and decoding of the memory address.

When all the input of an AND gates are “high” the output will also be “high”, therefore the AND gate can be considered as the simple decoder. The output which goes “high” when the inputs are high is known as “active High output". If NAND gate is connected instead of AND gate, then when all the inputs are “high” the output will be "Low" (0). Such output is called as "active low output". Example: A 2-to-4 Line Decoder. This is shown in figure 4.5.

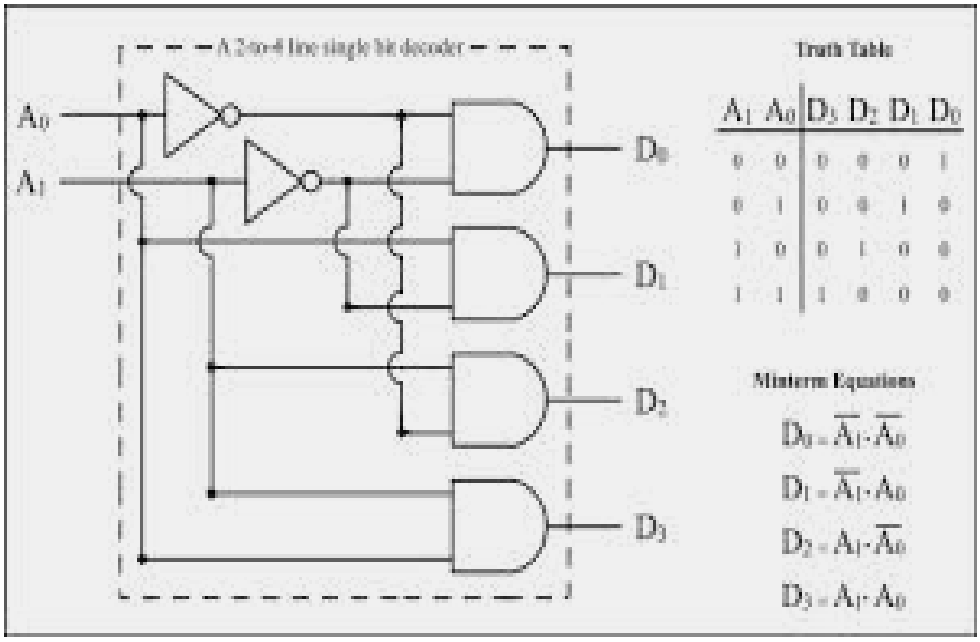


Figure 4.5: 2-to-4 Line Decoder

A slightly more complex decoder would be the n-to-2^n type binary decoders. With n-to-2^n binary decoders, maximum of ‘2^n’ outputs are generated from ‘n’ coded inputs which carry information. For ‘n’ bit coded input, if there are any unused combinations then number of outputs of the decoder can be less than 2^n. From above discussion we can say that a decoder can produce maximum of 2^n outputs. In digital systems various types of decoders like 3-to-8 decoder, 2-to-4 decoder or 4-to-16 decoder are used.

Two 2-to-4 decoders along with enable signal can be used to construct a 3-to-8 decoders.

In the same way, a 4-to-16 decoder can be constructed by combining two 3-to-8 decoders. In the above design process, the 4^{th} input which is given as enable input to both 3-to-8 decoders acts like selector between two 3-to-8 decoders.. The outputs D(0) through D(7) are produced by the first decoder and D(8) through D(15) are produced by the second decoder, the 4^{th} input in 4-to-16 decoder enables either the first decoder or second decoder.

This kind of decoders with enable signals is also known as a decoder-demultiplexer. Thus, we have a 4-to-16 decoder produced by adding a 4th input shared among both decoders, producing 16 outputs.

4.3.5 Encoders

An encoder is a device used to change a signal (such as a bit stream) or data into a code. The code may serve any of a number of purposes such as compressing information for transmission or storage, encrypting or adding redundancies to the input code, or translating from one code to another. This is usually done by means of a programmed algorithm, especially if any part is digital, while most analog encoding is done with analog circuitry.

Single bit 4 to 2 Encoder

An encoder has 2^n input lines and n output lines. The output lines generate a binary code corresponding to the input value. The figure 4.6 shows the logic symbol of typical encoder.

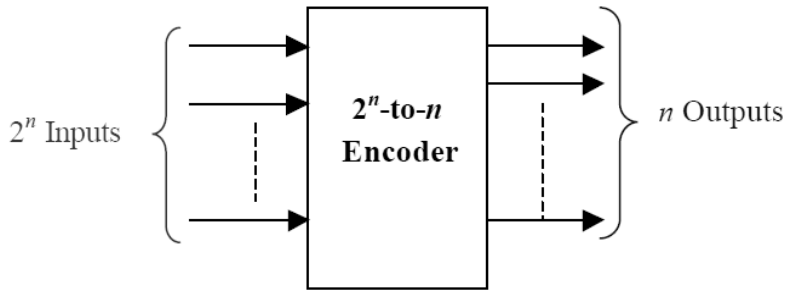


Figure 4.6: Logic symbol of typical encoder.

For example a single bit 4 to 2 encoder takes in 4 bits and outputs 2 bits. The Gate level circuit diagram of a single bit 4-to-2 line encoder is shown in figure 4.7 and its truth table shown in the table 4.5.

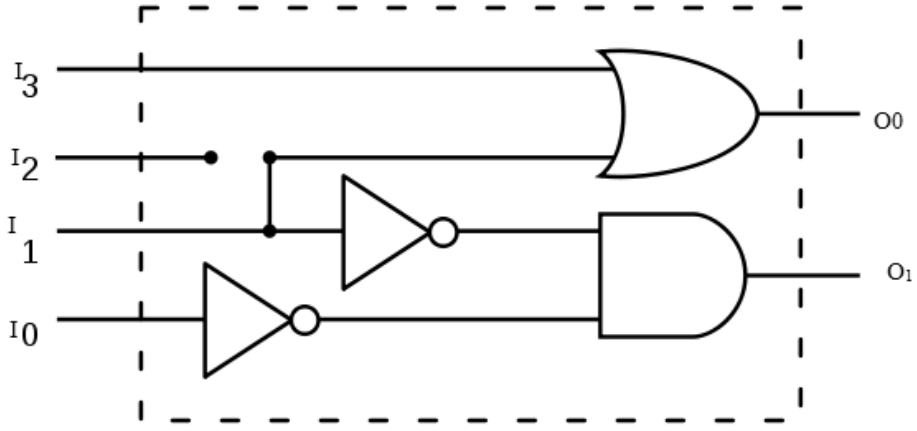


Figure 4.7: Gate level circuit diagram of a single bit 4-to-2 line encoder

Table 4.5: Truth table of 4-to-2 line Encoder

Inputs				Outputs	
I3	I2	I1	I0	**O1**	**O0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

The encoder has the limitation that only one input can be active at any given time. If two inputs are simultaneously active, the output produces an undefined combination. To prevent this we make use of the priority encoder.

Priority encoder

A priority encoder is a digital circuit in which when two or more inputs are given at the same time, the input having the highest priority will take precedence. An example of a single bit 4 to 2 priority encoder is shown in figure 4.8. Its truth table is shown in the table 4.6.

Table 4.6: Truth table of 4 to 2 priority encoder

I3	I2	I1	I0	O1	O0
0	0	0	d	0	0
0	0	1	d	0	1
0	1	d	d	1	0
1	d	d	d	1	1

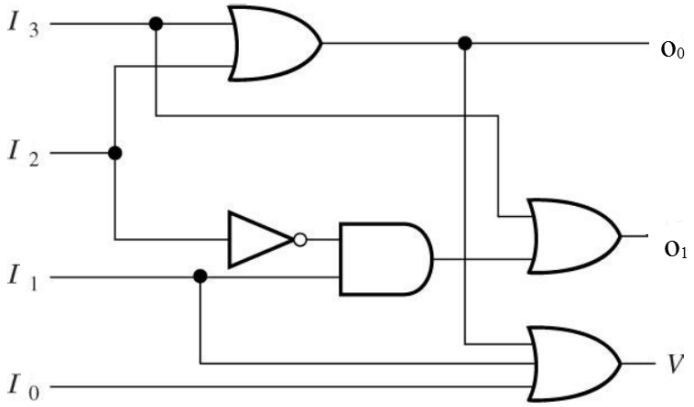


Figure 4.8: 4 to 2 priority encoder

Connecting Priority Encoders

Priority encoders can be easily connected in arrays to make larger encoders, such as a 16 to 4 encoder made from six 4 to 2 priority encoders (four encoders having the signal source connected to their inputs, and two encoders that take the output of the first four as input).

4.3.6 Multiplexers

In electronics, multiplexing of many signals is performed by the device known as multiplexer or mux. The combination of multiplexer and demultiplexer is termed as muldex. Many analog or digital signals can be given as input signals for the multiplexer; those input signals will be multiplexed into a single output line. An expensive device or any other resource can be shared with many signals using multiplexer, for example

instead of using A/D converter or communication line for every input signal, a single device can be used along with the multiplexer.

A multiplexer is a switch that has multiple inputs and single output. The schematic symbol for a multiplexer is an isosceles trapezoid. Inputs are connected to the multiplexer on the longer parallel side and output is connected on short parallel side. In the schematic shown in figure 4.9, on the left is the symbol for 2-to-1 multiplexer and on the right is the equivalent switch of the multiplexer. The single output line can be connected to the desired output through the sel wire.

In telecommunications, many communication channels can be carried using multiplexer which combines many input signals (which carries information) to a single output signal. A demultiplexer is a device which separates the input signal to multiple output signals.

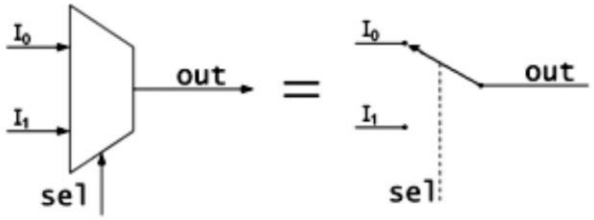


Figure 4.9: Schematic of a 2-to-1 Multiplexer

Multiplexer can be equated to a controlled switch.

Digital multiplexers

In digital circuit design, the values given for the selection line will be either zero or one. In the case of a 2-to-1 multiplexer, input I_0 is connected to the output signal when logic 0 is applied to selection line and input I_1 is connected to output when logic 1 is applied to selection line. In larger multiplexers, using n selection pins, 2^n input signals can be multiplexed.

For example, 4 selection pins are required to multiplex 9 to 16 input signals and 5 selection pins are required to multiplex 17 to 32 input signals. The selected input pin can be determined by the binary values on the selection pins.

For a 2-to-1 multiplexer shown in figure 4.10, the Boolean equation in terms of inputs and output can be as follows:

$$Z = (A \cdot \overline{S}) + (B \cdot S)$$

Where A and B are two input signals, S_0 is the selection line, and Z is the output signal.

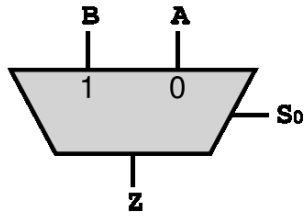


Figure 4.10: A 2-to-1 mux

This can be expressed in truth table as shown in the table 4.7.

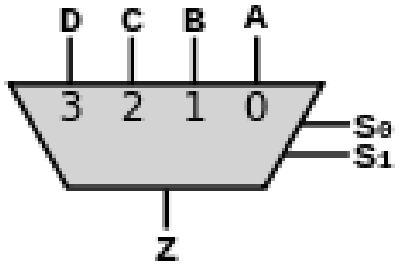
Table 4.7: Truth table for 2-to-1 Mux

S	A	B	Z
0	0	0	0
	0	1	0
	1	0	1
	1	1	1
1	0	0	0
	0	1	1
	1	0	0
	1	1	1

From the truth table 4.7, it can be observed that output Z will be equal to A when selection line S is 0 and Z is equal to B when S is 1. In order to realize the 2-to-1 multiplexer using gates, it requires 2 AND gates, a NOT gate and an OR gate.

In present day digital systems, large multiplexers are used commonly and, as stated above, for n input signals [$\log_2$ (n)] selection pins are required. Multiplexers that are used frequently apart from 2-to-1 multiplexer are 16-to-1, 8-to-1 and 4-to-1 multiplexers. Since digital logic uses binary values, for a given number of selection lines powers of 2 (2, 4, 8, 16) is used

to control the number of inputs signals. The figure 4.11 shows the symbol of 4 to1 Mux and figure 4.12 shows the symbol of 8 to1 Mux



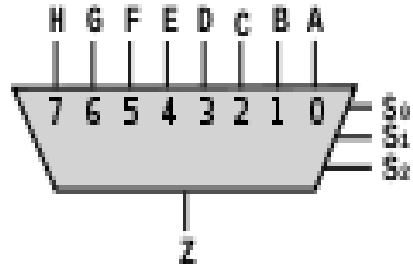


Figure 4.11: 4 to1 Mux **Figure 4.12: 8 to1 Mux**

The truth table of 4 to 1 Mux is shown in the table 4.8.

Table 4.8: Truth table of 4 to 1 Mux

S1	S0	Z
0	0	A
0	1	B
1	0	C
1	1	D

The Boolean equation for a 4-to-1 multiplexer is:

$$F = (A \cdot \overline{S_0} \cdot \overline{S_1}) + (B \cdot S_0 \cdot \overline{S_1}) + (C \cdot \overline{S_0} \cdot S_1) + (D \cdot S_0 \cdot S_1)$$

The figure 4.13 shows the logic circuit diagram of 4 to1 Mux.

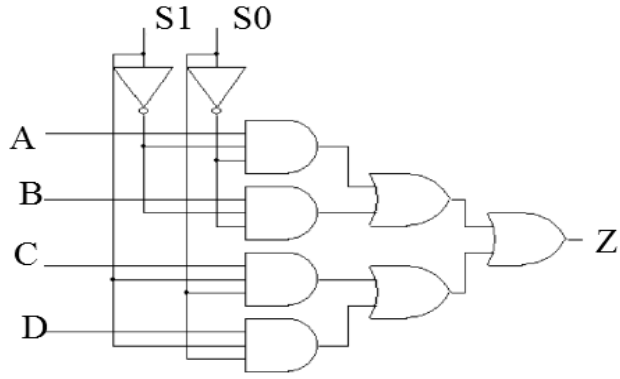


Figure 4.13: Logic circuit of 4 to1 Mux

4.3.7 De-Multiplexers

In electronics, a demultiplexer is a device which separates the input signal to multiple output signals. At the receiving end demultiplexer can be used as a complementary to the multiplexer which is at transmitting end. A demultiplexer can be considered as switch with single input and multiple-outputs. The figure 4.14 shows schematic of a 1 to 2 demultiplexer. A demultiplexer can also be equated to a controlled switch.

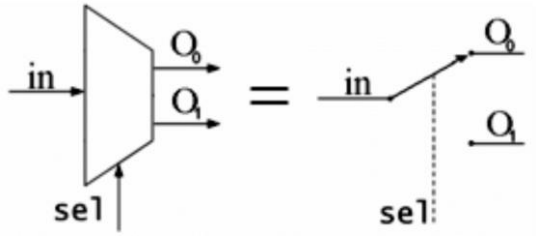


Figure 4.14: Schematic of a 1 to 2 Demultiplexer

4.4 Gray Code and its Properties

A Gray code is a function $G(i)$ of the integers i, that for each integer $N \geq 0$ is one-to-one for $0 \leq i \leq 2^N - 1$, and that has the following remarkable property: There will be only one bit difference between binary representation of G(i) and binary representation of G(i+1). Gray code is most commonly used technique in constructing digital systems. An example of a Gray code is, the sequence 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, and 1000. The bitwise exclusive or (XOR) of i with i /2(integer part) is performed to generate the Gray code. Think about how the carries work when you add one to a number in binary, and you will be able to see why this works. You will also see that $G(i)$ and $G(i+1)$ differ in the bit position of the rightmost zero bit of i (prefixing a leading zero if necessary).

The Gray codes are named after Frank Gray who patented the idea of using Gray codes in shaft encoders. A shaft encoder is a wheel which uses a fixed conducting brush to read the concentric coded stripes in order to generate a binary code for position of the wheel. Constructing a shaft encoder having a stripe conducting on one half of the wheel, insulating on other half; the next stripe is conducting in quadrants 1 and 3; the next stripe is conducting in octants 1, 3, 5, and 7; and so on. Then the brushes read a binary code from concentric coded stripes which describes the angle of the wheel.

With the above method, when the wheel is turning there is no guarantee that all the brushes break or make contact with stripes at the same time. As different brushes break or make contact, while going from position 7 (0111) to 8 (1000), one might pass spuriously and transiently through 6 (0110), 14 (1110), and 10 (1010).

One can avoid the transient states between 7 and 8 by using Gray code on the encoding stripes. An algorithm or a circuit which translates back the Gray code to integer is required. The Gray code is generated by a cascade of XOR gates; XOR of all most significant input bits gives an individual output bit. In a circuit, N-1 steps are required to perform the inversion of N bit Gray code. In a register with binary word operations, instead of doing N operations consecutively on $\ln_2 N$ operations are sufficient. The number of operations can be reduced by using associativity of XOR and hierarchically grouping the operations. The above step involves the right shits of input bits by 1, 2, 4, 8, and so on bits till the word length are zero.

Gray codes are not arithmetic codes. 4 bit gray code numbers for binary numbers are shown in table 4.15.

Table 4.9: 4 bit gray code numbers for Binary numbers

Decimal	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

Binary to Gray Conversion:

Binary to gray conversion employs following rules

- The most significant bit (MSB) in the Gray code is the same as the corresponding digit in the binary number
- Starting from left to right, perform Ex-OR operation on adjacent pair of binary bits to get the next Gray code bit.

Example: Compute the Gray code of the Binary number $110010_{(2)}$

Binary Number	1		1		0		0		1		0
	↓										
Gray Code	1										
Binary Number	1	⊕	1		0		0		1		0
			↓								
Gray Code	1		0								
Binary Number	1		1	⊕	0		0		1		0
					↓						
Gray Code	1		0		1						
Binary Number	1		1		0	⊕	0		1		0
							↓				
Gray Code	1		0		1		0				
Binary Number	1		1		0		0	⊕	1		0
									↓		
Gray Code	1		0		1		0		1		
Binary Number	1		1		0		0		1	⊕	0
											↓
Gray Code	1		0		1		0		1		1

Thus Binary number 110010 has a Gray code 101011

A combinational logic circuit to implement binary to gray code conversion is shown in figure 4.15 which employs Ex-OR logic gates.

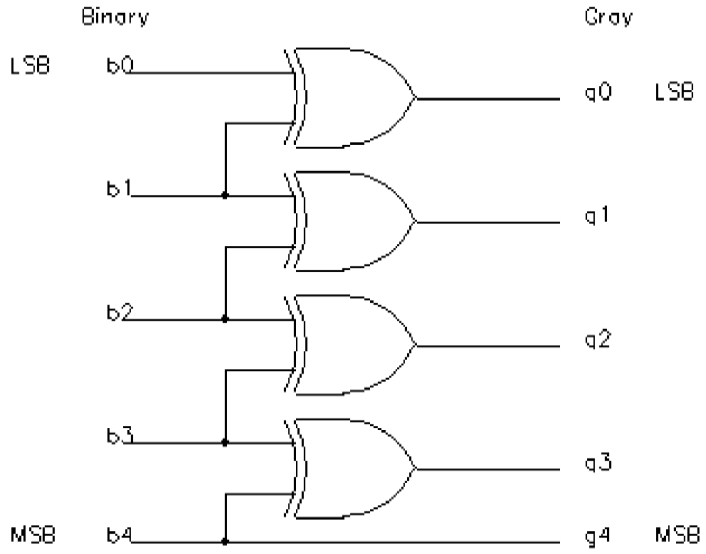


Figure 4.15: Logic circuitry for 5 bit Binary to Gray Conversion

Gray code to Binary Conversion

To convert from Gray code to binary, a similar method is used with the following rules

- The most significant bit (MSB) in the binary code is the same as the corresponding digit in the Gray code
- Perform Ex-OR operation with each binary bit generated to the next Gray code bit to get new binary bit.

Example: Convert the Gray code 101011 to Binary

Gray Code	1	0	1	0	1	1
	↓					
Binary Number	1					
Gray Code	1	0	1	0	1	1
	⊕	↓				
Binary Number	1	1				
Gray Code	1	0	1	0	1	1
		⊕	↓			
Binary Number	1	1	0			
Gray Code	1	0	1	0	1	1
			⊕	↓		
Binary Number	1	1	0	0		
Gray Code	1	0	1	0	1	1
				⊕	↓	
Binary Number	1	1	0	0	1	
Gray Code	1	0	1	0	1	1
					⊕	↓
Binary Number	1	1	0	0	1	0

Thus Gray code 101011 has a Binary number 110010

A combinational logic circuit to implement binary to gray code conversion is shown in figure 4.16 which employs Ex-OR logic gates.

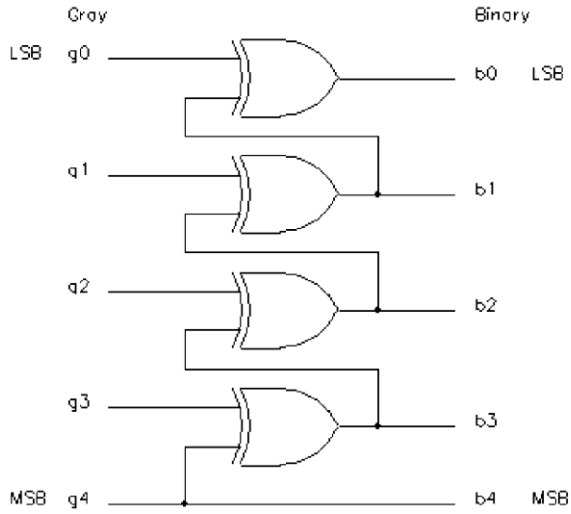


Figure 4.16: Logic circuitry for 5 bit Gray to Binary Conversion

4.5 BCD Code and its properties

In digital system, decimal numbers are encoded to binary sequence using binary coded decimal (BCD) technique. The advantage of the BCD encoding is that the conversion of binary sequence to decimal digits for printing or display is easier and also the decimal calculations will be faster. The drawbacks of the BCD encoding are the complexity of the digital circuits which are used to implement the mathematical operations increases and because of a relative inefficient encoding than a pure binary representation BCD requires more space.

For financial, industrial and commercial computing, decimal fixed point and floating point are used frequently, but BCD is not used widely as it was once. Instead of using BCD encodings, base 10 exponents are used to represent modern floating point. In BCD, four bits are used to represent a digit. In general these fours bits represent the values/digits/characters 0-9. For sign or other indications, other bit combinations are used.

To BCD-encode a decimal number using the common encoding, each decimal digit is stored in a four-bit nibble.

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

Thus, the BCD encoding for the number 256 would be:
0010 0101 0110

As the data in most of the computers are stored in eight bit bytes, the four bit BCD digits can be stored in those bytes using two ways:

In a byte, BCD digit is stored in one nibble, and other nibble of a byte is set to all zeros or all ones (as in the EBCDIC code), or to 0011 (as in the ASCII code) two digits are stored in each byte.

By mapping each nibble to a different character, BCD encoded numbers can be easily displayed. Generally, integer multiplication or division operation are involved in conversion of binary coded number to decimal, because of above reason the display of binary coded number is harder. In electronic systems which have only digital logic but no microprocessor, BCD is used to display the numeric values. In BCD each digit has been treated as a single sub circuit, because of that the manipulation of numerical values for display can be simplified to great extent. This matches much more closely the physical reality of display hardware. For example, a designer can develop a metering circuit by choosing a series of identical 7-segment displays. A complex circuitry would be required to interface a display device if the numeric data is manipulated and stored in the form of pure binary. Therefore, using BCD a simple digital system can be developed than converting to pure binary, and also calculations are relatively simple. Even the digital system which has embedded microcontroller or small processor will hold the same argument. On limited processors the process of converting numeric's to or from binary representation can be expensive, less expensive systems can be developed by representing numeric's in BCD format and often it results in smaller code. For these applications, some small processors feature BCD arithmetic modes, which assist when writing routines that manipulate BCD quantities.

Packed BCD

In packed BCD encoding two digits are placed in a single byte. Packed BCD encoding is also known as simply packed decimal. Decimal integers are stored in all upper bytes plus upper four bits of the lowest byte of a multi byte word. The lower four bits of the lowest byte are used as the sign flag. As an example, in a word of 32 bits which contains 8 nibbles or 4 bytes

lowest nibble is used stores sign flag of the decimal value and integers are stored in upper 7 nibbles.

Standard sign values are 1100 (Hex C i.e Ch) for positive (+) and 1101 (Dh) for negative (–). Other allowed signs are 1010 (Ah) and 1110 (Eh) for positive and 1011 (Bh) for negative. Some implementations also provide unsigned BCD values with a sign nibble of 1111 (Fh). In packed BCD, the number 127 is represented by "0001 0010 0111 1100" (127Ch) and -127 is represented by "0001 0010 0111 1101 (127Dh).

Sign	BCD		
Digit	**8 4 2 1**	**Sign**	**Notes**
A	1 0 1 0	+	
B	1 0 1 1	−	
C	1 1 0 0	+	Preferred
D	1 1 0 1	−	Preferred
E	1 1 1 0	+	
F	1 1 1 1	+	Unsigned

As two digits are stored in a byte, the number of nibbles is always even irrespective of size of the word. Therefore there can be (2n)-1 decimal digits in a n bytes word. The number of decimal digit is always odd. A decimal number with d digits requires ½(d+1) bytes of storage space.

For example, a four-byte (32bit) word can hold seven decimal digits plus a sign, and can represent values ranging from ±9,999,999. Thus the number

-1,234,567 is 7 digits wide and is encoded as:

0001 0010 0011 0100 0101 0110 0111 1101

1 2 3 4 5 6 7 -

(Note that, like character strings, irrespective to endianness of the digital system – lowest address space in memory is used to store the first byte of the packed decimal).

In contrast, a four-byte binary two's complement integer can represent values from −2,147,483,648 to +2,147,483,647.

While packed BCD does not make optimal use of storage (about 1/6 of the memory used is wasted), conversion to ASCII, EBCDIC, or the various encodings of Unicode is still trivial, as no arithmetic operations are required.

The extra storage requirements are usually offset by the need for the accuracy that fixed-point decimal arithmetic provides. More dense packings of BCD exist which avoid the storage penalty and also need no arithmetic operations for common conversions.

Fixed-point packed decimal

Programming languages such as COBOL and PL/I support fixed point decimal numbers, and a decimal point has to be provided in front of one of the digits. For example, the fixed point value +1234.567 can be represented by 12 34 56 7C (a packed decimal value) if the decimal point is placed between 4^{th} and 5^{th} digits.

12 34 56 7C

12 34.56 7+

Higher-density encodings

Totally 12 bits are required for a three decimal digits if each digit is represented using four bits. However, since 2^{10} (1,024) is greater than 10^3 (1,000), only 10 bits are sufficient if the encoding of all three decimal digits done together. Two such encodings are Chen-Ho encoding and Densely Packed Decimal. Densely Packed Decimal encodes two digits in the optimal 7 bits and one digit in 4 bits.

Zoned decimal

An IBM mainframe system uses Zoned decimal numeric representations. In this encoding, each digit is stored in one byte with lower four bits encoding the digit in BCD form. The upper bits are called the zone bits; they are set to a fixed value so that the character value of the digit is stored in the byte. EBCDIC systems use a zone value of 1111 (hex F); this yields bytes in the range F0 to F9 (hex), which are the EBCDIC codes for the characters "0" through "9". Similarly, ASCII systems use a zone value of 0011 (hex 3), giving character codes 30 to 39 (hex). For signed zoned decimal values, the sign digit of a numeric is stored in the least significant zone nibble of the byte. Even for packed decimal, sign bits are stored in the same set of values. Thus a zoned decimal value encoded as hex bytes F1 F2 D3 represents the signed decimal value −123.

i.e. F1 F2 D3 is the zoned representation of -123.

The table 4.10 shows the zoned decimal conversion table

Table 4.10: EBCDIC zoned decimal conversion table

Digit	EBCDIC Display	EBCDIC Hex
0+	{(*)	X'C0'
1+	A	X'C1'
2+	B	X'C2'
3+	C	X'C3'
4+	D	X'C4'
5+	E	X'C5'
6+	F	X'C6'
7+	G	X'C7'
8+	H	X'C8'
9+	I	X'C9'
0−	} (*)	X'D0'
1−	J	X'D1'
2−	K	X'D2'
3−	L	X'D3'
4−	M	X'D4'
5−	N	X'D5'
6−	O	X'D6'
7−	P	X'D7'
8−	Q	X'D8'
9−	R	X'D9'

(*) Note: These characters vary depending on the local character code page.

Fixed-point zoned decimal

Languages like COBOL and PL/I use fixed point decimal values. The decimal point will be assigned at some location between the decimal values of a number. For example, given a seven-byte signed zoned decimal value with an implied decimal point to the right of the fifth digit, the hex bytes F3 F2 F1 F6 F8 F4 C0 represent the value +32,178.40:

F3 F2 F1 F6 F8 F4 C0

3 2 1 6 8. 4 +0

IBM and BCD

For 6-bit alphanumeric codes that represents special characters, numbers and upper case letters, IBM uses the terms BCD and binary coded decimal. In most early IBM computers use some variations of BCD alphanumerics. These computers include IBM 1400 series, IBM 1620 and non-Decimal Architecture members of IBM 700/7000 series.

In BCD B, A, 8, 4, 2, 1 is labeled as bit positions. B and A are zero's while encoding for digits. The letter A was encoded (B, A, 1).

In IBM 1620 computer, a digit pairs are used to encode the BCD alphanumerics. In those digit pairs, the "digit" will be the odd digit and the "zone" will be the even digit. The conversion between external standard 6 bit BCD codes and the internal digit pairs is done by the input/output translation hardware.

In the Decimal Architecture 7074, 7070, and 7072, digit pairs of 10-digit word are used to encode the alphamerics. These digit pairs instead of BCD digits, use two-out-of-five code in the digits. "Digit" will be in right digit of digit pair and zone will be in the left of digit pair. The conversion between external standard 6 bit BCD codes and the internal digit pairs is done by the input/output translation hardware.

In IBM System/360, the addition of more characters like lowercase letters are allowed by expanding 6 bit BCD alphanumerics to 8 bit EBCDIC. Also IBM implemented a variable length Packed BCD numeric data type.

The current IBM databases and processors still use BCD data. BCD in the IBM processors and databases are usually Packed BCD, zoned BCD as in ASCII or EBCDIC or "pure" BCD encoding. The software, processing units and hardware registers uses the above BCD encodings.

Addition with BCD

The addition in BCD can be performed in three steps: 1) Add the numbers in binary. 2) Check if the number is greater than 9. 3) Then convert the sum of two numbers to BCD number by adding correction value. The correction value will be equal to 6 if the sum output in step 1 is greater than 9, else the correction value will be zero. For example:

8 + 6 = 14 = [1000] + [0110] = [0000 1110] in binary.

However, in BCD, per nibble value greater than 9 cannot exist. To correct the value of first two digits, sum will be added with 6 (0110):
[0000 1110] + [0000 0110] = [0001 0100]

[0001] and [0100] are two nibbles, which correspond to "1" and "4" respectively. This gives correct result “14” in BCD. The above technique can be extended to adding multiple digits groups from right to left, the second digit is propagated as a carry, and always 5 bit result is compared to 9.

Subtraction with BCD
Subtraction is done by adding the nines' complement plus 1, or by adding the ten's complement of the subtrahend.

4.6 Excess-3 Code and its Properties

Excess-3 binary coded decimal (XS-3), also called biased representation or Excess-N, is a numeral system used on some older computers that uses a pre-specified number N as a biasing value. It is a way to represent values with a balanced number of positive and negative numbers. In XS-3, numbers are represented as decimal digits, and each digit is represented by four bits as the BCD value plus 3 (the "excess" amount):

The smallest binary number represents the smallest value. (i.e. 0 - Excess Value).The greatest binary number represents the largest value. (i.e. 2^{N+1} - Excess Value - 1). The table 4.11 shows the XS3 code for decimal numbers.

Table 4.11: XS3 code for decimal numbers.

Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary
-3	0000	1	0100	5	1000	9	1100
-2	0001	2	0101	6	1001	10	1101
-1	0010	3	0110	7	1010	11	1110
0	0011	4	0111	8	1011	12	1111

To encode a number such as 127, then, one simply encodes each of the decimal digits as above, giving (0100, 0101, 1010).

The primary advantage of XS-3 coding over BCD coding is that a decimal number can be nines' complemented (for subtraction) as easily as a binary number can be ones' complemented; just invert all bits.

Adding Excess-3 works on a different algorithm than BCD coding or regular binary numbers. When you add two XS-3 numbers together, the result is not an XS-3 number. For instance, when you add 1 and 0 in XS-3 the answer seems to be 4 instead of 1. In order to correct this problem, when you are finished adding each digit, you have to subtract 3 (binary 11) if the digit is less than decimal 10 and add three if the number is greater than or equal to decimal 10.

Assessment Questions

1. Circuits whose outputs depend only on the current inputs are called _____.
2. The circuit which adds the carry bit from previous stage with two one bit inputs is known as _____.
3. A _____ circuits will generate sum (S) and carry (C) from the two of its inputs A and B.
4. Comparators can be used in a central processing unit (CPU) or microcontroller in branching software (True or False?).
5. A _____ is a digital device which decodes the original information from the encoded inputs.
6. A _____ is a digital circuit in which when two or more inputs are given at the same time, the input having the highest priority will take precedence.
7. The _____are named after Frank Gray who patented the idea of using Gray codes in shaft encoders.
8. The Gray code is generated by a cascade of OR gates (True or False?).
9. In binary to gray code conversion, the most significant bit (MSB) in the gray code is the same as the corresponding digit in the binary number (True or False?).
10. An IBM mainframe system uses _____ decimal numeric representations.
11. Excess-3 binary coded decimal (XS-3), also called _____.
12. When you add two XS-3 numbers together, the result is not an XS-3 number (True or False?).

4.7 Summary

Let us recapitulate the important concepts discussed in this unit:

- A digital circuit which generates a set of outputs from set of inputs using Boolean operations is known as combinational circuits.
- A half adder is a digital circuit which accepts two inputs and performs addition on them and generates two binary digits known as sum(S) and carry(C).
- A digital comparator is a hardware electronic device that compares two numbers in binary form and generates a one or a zero at its output depending on whether they are the same (i.e. equal) or not.
- An encoder is a device used to change a signal (such as a bit stream) or data into a code.
- A decoder is a digital device which decodes the original information from the encoded inputs.
- Multiplexing of many signals is performed by the device known as multiplexer or mux.
- In packed BCD encoding two digits are placed in a single byte.
- Excess-3 binary coded decimal (XS-3), also called biased representation

4.8 Terminal Questions

1. What is a full adder? Explain the operation with a neat logic diagram.
2. What is comparator? Design the combinational circuit that will compare two bits A and B produce 3 outputs A>B, A=B and A<B.
3. What is the function of decoder? Explain.
4. What is an encoder? Explain priority encoder.
5. What is Multiplexer? Draw the logic diagram of 4 to 1 line multiplexer.
6. Explain Binary to Gray Conversion with an example.

4.9 Answers

Self Assessment Questions

1. Combinational
2. Full adder
3. Half adder
4. True

5. Decoder
6. Priority encoder
7. Gray codes
8. True
9. True
10. Zoned
11. Biased representation
12. True

Terminal Questions

1. Refer to section 4.3.
2. Refer to section 4.3.3
3. Refer to sub-section 4.3.4
4. Refer to sub-section 4.3.5
5. Refer to sub-section 4.3.6
6. Refer to section 4.4