DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC-201 LOGIC & SEQUENTIAL CIRCUIT DESIGN

MORSE CODE ENCODER AND DECODER

PROJECT REPORT

SUBMITTED TO:
Dr. Soyiba Jawed

SUBMITTED BY:

Group Members:	Registration No.
Eshmal Rumman	413285
Shaheer Mukhtiar	432017
Abeera Shahid	417494

DE- 44 Dept C&SE

Submission Date:14/1/2024

1. Project Overview:

The "Morse Code Encoder and Decoder" project focuses on creating a versatile system in Verilog to encode and decode Morse code, linking historical significance with modern applications. Emphasizing its relevance in digital circuit design, the project underscores the importance of applying theoretical knowledge to real-world scenarios. The primary objectives include designing modules for encoding and decoding, allowing users to activate either function based on their needs. By incorporating 5-bit binary representation and configurability, the project aims to showcase the practical application of digital circuit design principles in developing adaptable communication systems.

2. Objectives:

1. **Encoder Module:**
    - Convert alphabets into Morse code.
    - Utilize a 5-bit binary representation of characters (1-26) as input.
    - Output Morse code with dots represented as 0 and dashes as 1.
2. **Decoder Module:**
    - Translate Morse code back into alphabets.
    - Accept Morse code (1s and 0s) as input.
    - Output 5-bit binary representation of characters (1-26).

3. Implementation Details:

Verilog Implementation:

- Utilized Verilog for coding the Morse Code Encoder and Decoder modules.
- Ensured the code is structured, readable, and adheres to best practices in hardware description languages.

Equations Derivation and K-map Implementation:

- Derived K-map equations for each bit of the output using the truth table approach studied in class.
- Translated these equations into Proteus simulation for practical implementation on hardware.

Hardware Implementation for 3-Bit Morse Code:

- Implemented the designed circuits on breadboard.
- Validated the functionality of the circuits through hardware testing and troubleshooting.

Configurability using Demux:

- Integrated a demux to enable either the Morse code encoder or decoder based on user input (1/0).
- Ensured seamless switching between encoder and decoder functionalities.

4. Design Specifications:

1- Morse Code Table:

A table mapping each alphabet to its Morse code representation is created. This table serves as the basis for both the encoder and decoder implementation.

Letter	Morse	Number	Binary
A	.-	00001	1
B	-...	00010	2
C	-.-.	00011	3
D	-..	00100	4
E	.	00101	5
F	..-.	00110	6
G	--.	00111	7
H	....	01000	8
I	..	01001	9
J	.---	01010	10
K	-.-	01011	11
L	.-..	01100	12
M	--	01101	13
N	-.	01110	14
O	---	01111	15
P	.--.	10000	16

Q	--.-	10001	17
R	.-.	10010	18
S	...	10011	19
T	-	10100	20
U	..-	10101	21
V	...-	10110	22
W	.--	10111	23
X	-..-	11000	24
Y	-.--	11001	25
Z	--..	11010	26

2- Verilog:

Encoder:

The Verilog code for encoder provided in half implementation report consists of two modules: alphaROM and MorseCodeEncoder. The alphaROM module acts as a ROM, converting Morse code bits for individual alphanumeric characters. The MorseCodeEncoder module, which instantiates alphaROM, functions as a Morse code encoder. It takes a 5-bit alphanumeric input and a 2-bit selector, producing four outputs representing Morse code bits for the selected alphanumeric character. The test bench, tb_MorseCodeEncoder, simulates the encoder's behavior under different input scenarios. A VCD file is generated for waveform analysis.

Here's a brief overview of Verilog code:

```
// alphaROM Module for Decoder

module alphaROM (out, in_morse_bit, morse_in, clk);

  reg [5:0] mem [0:25];

  input clk;

  input [2:0] in_morse_bit;

  input [3:0] morse_in;
```

```
  output reg [6:0] out;
  // Parameter and Memory Initialization
  // ... (same as in previous response)
  always @(posedge clk) begin
    if (in_morse_bit == 3'b001)
      case (morse_in)
        E: out = mem[4];
        T: out = mem[19];
        // ... (additional cases for other letters)
        default: out = 7'b0000000;
      endcase
    else if (in_morse_bit == 3'b010)
      case (morse_in)
        I: out = mem[8];
        A: out = mem[0];
        N: out = mem[13];
        M: out = mem[12];
        // ... (additional cases for other letters)
        default: out = 7'b0000000;
      endcase
    // ... (additional cases for in_morse_bit 3'b011, 3'b100, etc.)
  end
endmodule

// Decoder Module
module de (morse_in, num, clk, out);
  input clk;
```

```
  input [3:0] morse_in;
  input [2:0] num;
  output [4:0] out;
  alphaROM r (out, num, morse_in, clk);
endmodule

// Test Bench for Decoder
module tb_de;
  reg clk;
  reg [3:0] morse_in;
  reg [2:0] num;
  wire [4:0] out;

  // Instantiate the Decoder module
  de my_de (
    .out(out),
    .num(num),
    .morse_in(morse_in),
    .clk(clk)
  );
  // Clock generation
  initial begin
    clk = 1;
    forever #5 clk = ~clk;  // Toggle the clock every 5 time units
  end

  // Test scenario
```

```
 initial begin
  // Apply inputs and observe outputs for a few clock cycles
  num = 3'b100; morse_in = 4'b1010;   // C
  // ... (additional test cases)
  #10 $finish;
 end
 initial begin
  $dumpfile("dump.vcd"); $dumpvars;
 end
endmodule
```

Decoder:

Similarly, the Verilog code for decoder provided in half implementation report includes two main modules: alphaROM and de (Demultiplexer), designed to create a configurable Morse code to alphanumeric character converter. The alphaROM module acts as a ROM, mapping Morse code bits to alphanumeric characters based on a predefined set of parameters. The de module serves as a demultiplexer, taking Morse code and a selector to output the corresponding alphanumeric character. The code is organized, with a test bench (tb_de) simulating various input scenarios. The generated VCD file allows for waveform analysis to verify the correct functionality of the system.

Here's a brief overview of Verilog code:

```
// Initialization of Parameters and ROM Data
parameter A = 5'b00001; // A
parameter B = 5'b00010; // B
parameter C = 5'b00011; // C
// ... (parameters for other letters)


initial begin
 rom_data1[0] = 1'b0; // E (dot)
 rom_data1[1] = 1'b1; // T (dash)
```

```
  // ... (initialization for other Morse code bits)
end
// alphaROM Module Always Block
always @(posedge clk) begin
  if (in_morse_bit == 2'b00)
    case (alpha_in)
      E: out1 = rom_data1[0];
      T: out1 = rom_data1[1];
      // ... (additional cases for other letters)
    endcase
  else if (in_morse_bit == 2'b01)
    case (alpha_in)
      A: out2 = rom_data2[0];
      I: out2 = rom_data2[1];
      // ... (additional cases for other letters)
    endcase
  // ... (additional cases for in_morse_bit 2'b10 and 2'b11)
end
// MorseCodeEncoder Test Bench
initial begin
  // Test case 1
  alpha_in = 5'b00101; // E
  num = 2'b00;
  #10;
  alpha_in = 5'b01110; // N
  num = 2'b01;
  #10;
```

```
  alpha_in = 5'b00111; // G
  num = 2'b10;
  #10;
  alpha_in = 5'b10001; // Q
  num = 2'b11;
  #10;
  $stop;
end
```

3- K-map Equations:

Encoder:

For 1-bit morse code:

Y=e’

For 2-bit morse code:

Y1=c

Y2=b’+cd’

For 3-bit morse code:

Y1=a’

Y2=c’e’+cd

Y3=b+ac

For 4-bit morse code:

Y1=e+ab+a’b’c’

Y2=b’d’+cd’+bd

Y3=a’e+be+b’d’e’+a’b’c+a’bd

Y4=d’e+ac+a’bd+abd’

Decoder:

For 1-bit morse code:

a=x

b=0

c=1

d=0

e=x’

For 2-bit morse code:

a=0

b=x+y’

c=x

d=xy’

e=x’+y

For 3-bit morse code:

a=x’

b=xz

c=xz’+x’z+yz

d=y+x’z’+xz

e=z+x’y’+xy

For 4-bit morse code:

a=wx+x’z+xyz

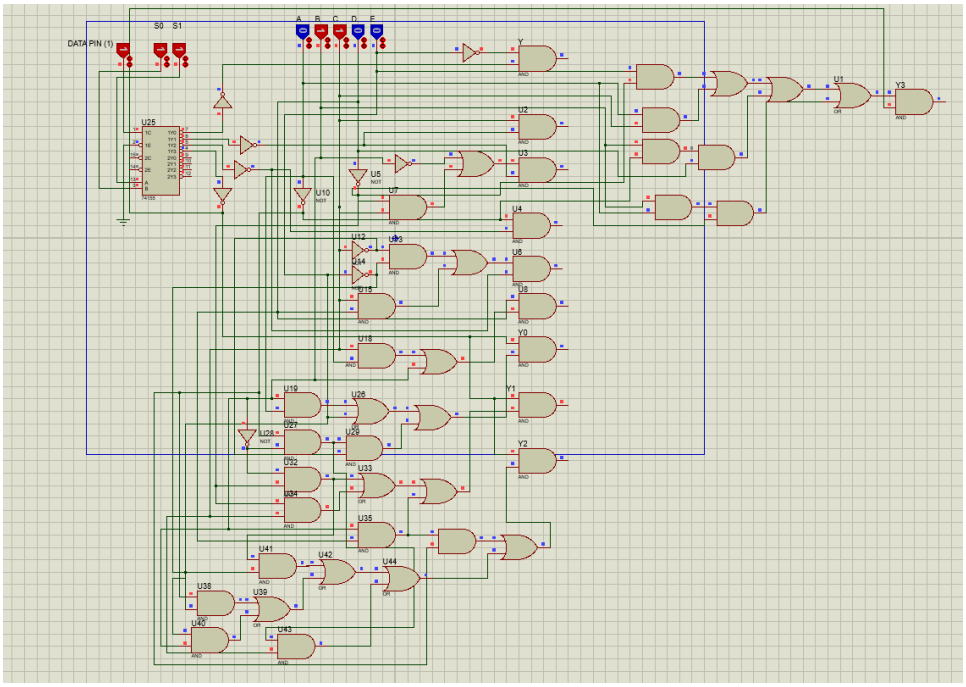
b=yz+wxz’+w’y’z’+wx’z

c=w’xy’+w’x’z+w’x’y

d=wz’+w’z+x’yz’

e=wy+xy’z

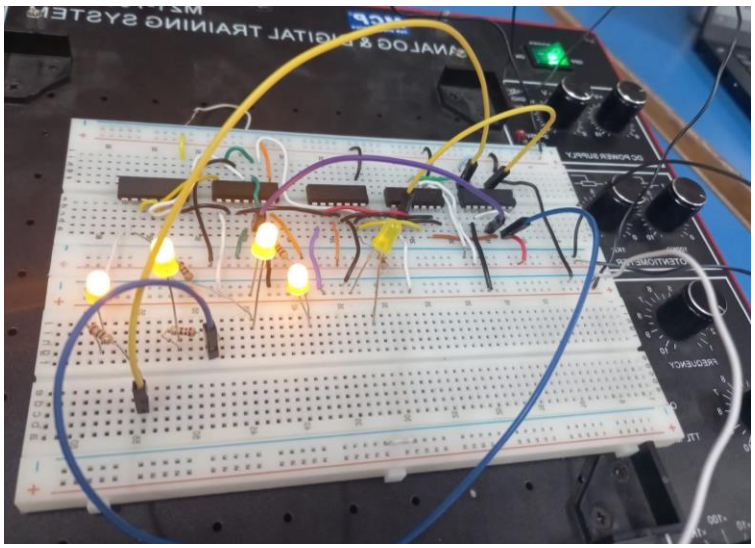
4- Circuit Simulation:

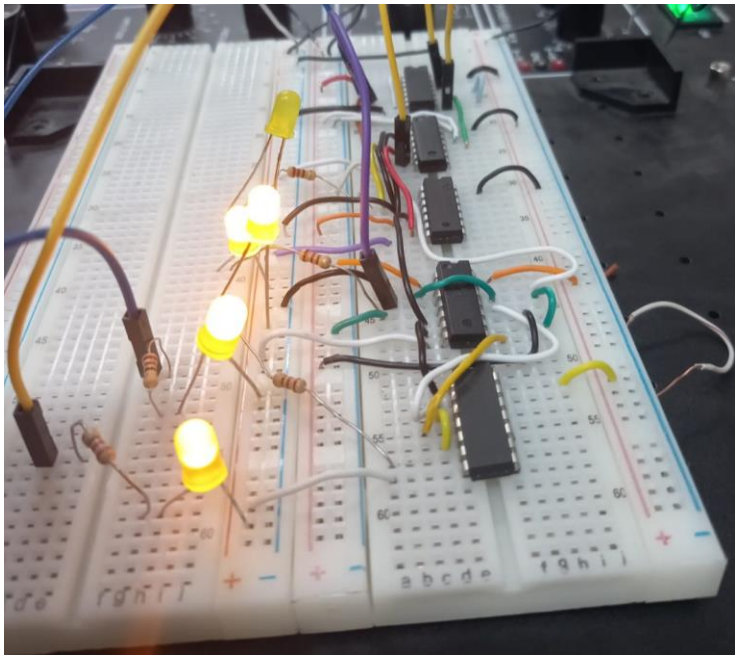


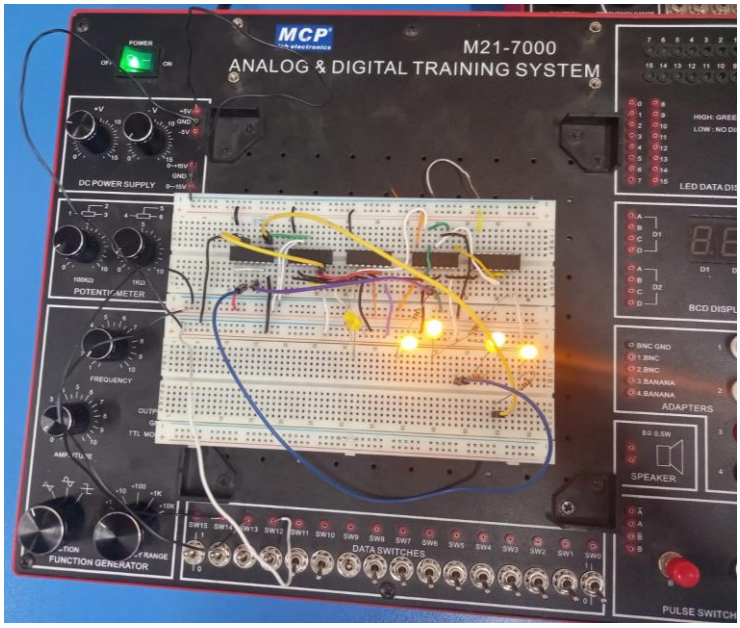
5. Hardware Implementation:

Components used:

- AND Gate IC 74LS08
- OR Gate IC 74LS32
- NOT Gate IC 74LS04
- 1x4 Demux IC 74LS139
- LEDs







6. Results:

The project's outcomes confirm the effective implementation of Morse Code Encoder and Decoder circuits. Testing shows the encoder's precise conversion of 5-bit binary representations to Morse code and the decoder's accurate translation of Morse code back to binary form.

7. Course Learning Outcomes Alignment:

The Morse Code Encoder and Decoder project aligns with the following Course Learning Outcomes:

CLO 2: Demonstrate knowledge and application of digital logic design concepts, including basic and universal gates, number systems, and the design of combinational and sequential circuits.

CLO 3: Analyse small- and medium-scale combinational and sequential digital circuits using Boolean algebra and Karnaugh map methods.

CLO 4: Design small-scale combinational and synchronous sequential digital circuits through experimentation, validating concepts learned in class.

8. Troubleshooting:

Problem: Initial Complexity with 7-Bit ASCII Representation

Representing alphabet characters by their ASCII values in 7-bit binary initially posed complexity, especially when dealing with 7-variable Karnaugh maps during logical equation derivation for the Morse code encoder and decoder.

Solution: Streamlined 5-Bit Binary Representation

To address this challenge, we transitioned to representing alphabets with numerical values and their corresponding 5-bit binary codes. This shift simplified the logic design process, ensuring a more efficient and manageable implementation.

9. Conclusion:

This project provided hands-on experience in applying digital circuit design concepts to create a Morse Code Encoder and Decoder using Verilog. It reinforced understanding of Boolean algebra, Karnaugh maps, and the design of combinational and sequential circuits. The practical implementation involved systematic equation derivation, gate utilization, and thorough circuit validation through simulations. The incorporation of user-selectable functionality demonstrated the versatility of digital circuit designs. Overall, the project deepened our knowledge and skills in designing, implementing, and testing digital circuits, bridging theoretical concepts with practical applications.

10. References:

- Morse Code Basics