DEPARTMENT OF COMPUTER &
SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC-201 LOGIC & SEQUENTIAL CIRCUIT DESIGN

MORSE CODE ENCODER AND DECODER

HALF IMPLEMENTATION 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

1. Project Overview:

The **"Morse Code Encoder and Decoder"** project aims to design a configurable Morse code system in Verilog, providing users with the option to enable either the Morse code encoder or decoder based on their requirements. Morse code is a method used to represent text characters using sequences of dots and dashes. The project will involve creating two main modules: an encoder that converts text messages into Morse code and a decoder that translates Morse code back into text.

2. System Architecture

2.1. Morse Code Encoder

The Morse code encoder will convert alphabets into Morse code, providing users with the option to enable this functionality.

2.2. Morse Code Decoder

The Morse code decoder will retrieve the original alphabet from a Morse code sequence, and users can choose to enable this decoding option.

3. Implementation Details

3.1. Morse Code Table

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

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

4. Decoder Implementation

HIGH-LEVEL

Code:

```
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 A = 2'b 01; //A
parameter B = 4'b 1000;         //B
parameter C = 4'b 1010;         //C
parameter D = 3'b 100;          //D
parameter E = 1'b 0 ;  //E
parameter F = 4'b 0010;         //F
parameter G = 3'b 110;          //G
parameter H = 4'b 0000;         //H
parameter I = 2'b 00;  //I
parameter J = 4'b 0111;         //J
parameter K = 3'b 101;          //K
parameter L = 4'b 0100;         //L
parameter M = 2'b 11;           //M
parameter N = 2'b 10; //N
parameter O = 3'b 111;          //O
parameter P = 4'b 0110;         //P
parameter Q = 4'b 1101;         //Q
```

```
parameter R = 3'b 010;          //R
parameter S = 3'b 000;          //S
parameter T = 1'b 1;   //T
parameter U = 3'b 001;          //U
parameter V = 4'b 0001;         //V
parameter W = 3'b 011;          //W
parameter X = 4'b 1001;         //X
parameter Y = 4'b 1011;         //Y
parameter Z = 4'b 1100;         //Z

initial begin
        mem[0] = 5'b 00001;  //A
        mem[1] = 5'b 00010;  //B
        mem[2] = 5'b 00011;  //C
        mem[3] = 5'b 00100;  //D
        mem[4] = 5'b 00101;  //E
        mem[5] = 5'b 00110;  //F
        mem[6] = 5'b 00111;  //G
        mem[7] = 5'b 01000;  //H
        mem[8] = 5'b 01001;  //I
        mem[9] = 5'b 01010;  //J
        mem[10] = 5'b 01011;//K
        mem[11] = 5'b 01100;//L
        mem[12] = 5'b 01101;//M
        mem[13] = 5'b 01110;//N
        mem[14] = 5'b 01111;//O
        mem[15] = 5'b 10000;//P
```

```
        mem[16] = 5'b 10001;//Q
        mem[17] = 5'b 10010;//R
        mem[18] = 5'b 10011;//S
        mem[19] = 5'b 10100;//T
        mem[20] = 5'b 10101;//U
        mem[21] = 5'b 10110;//V
        mem[22] = 5'b 10111;//W
        mem[23] = 5'b 11000;//X
        mem[24] = 5'b 11001;//Y
        mem[25] = 5'b 11010;//Z
end

always@(posedge clk) begin
if (in_morse_bit == 3'b 001)
   case (morse_in)
        E: out = mem[4];
        T: out = mem[19];
        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];
        default: out = 7'b0000000;
   endcase
```

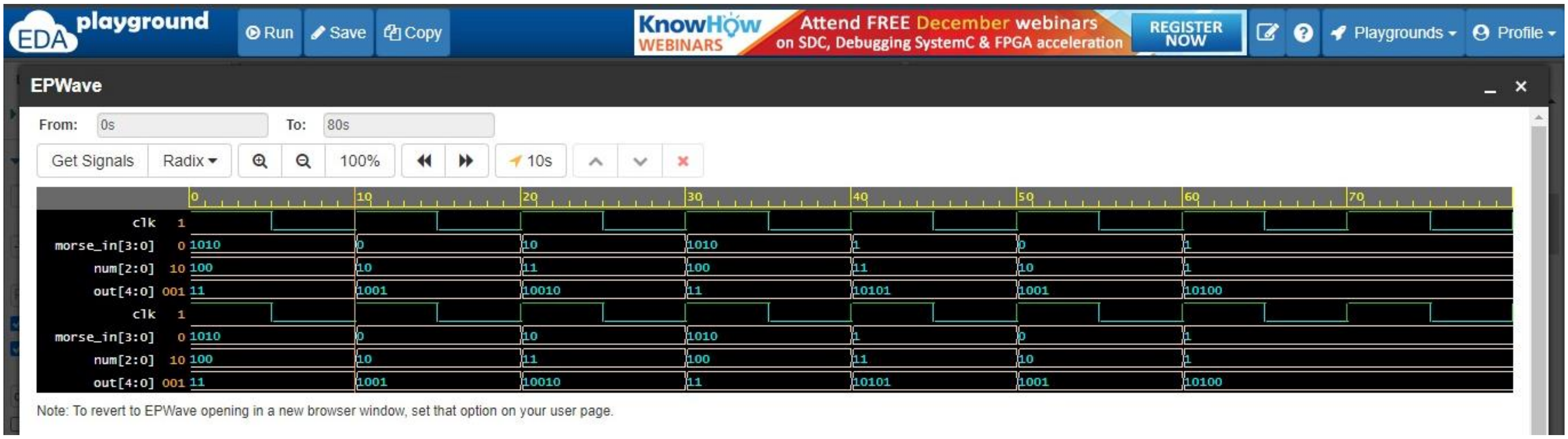
```
if (in_morse_bit == 3'b 011)
   case (morse_in)
         S: out = mem[18];
         U: out = mem[20];
         R: out = mem[17];
         W: out = mem[22];
         D: out = mem[3];
         K: out = mem[10];
         G: out = mem[6];
         O: out = mem[14];
         default: out = 7'b0000000;
   endcase
if (in_morse_bit == 3'b 100)
   case (morse_in)
         H: out = mem[7];
         V: out = mem[21];
         F: out = mem[5];
         L: out = mem[11];
         P: out = mem[15];
         J: out = mem[9];
         B: out = mem[1];
         X: out = mem[23];
         C: out = mem[2];
         Y: out = mem[24];
         Z: out = mem[25];
         Q: out = mem[16];
         default: out = 7'b0000000;
```

```
   endcase
end
endmodule

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:

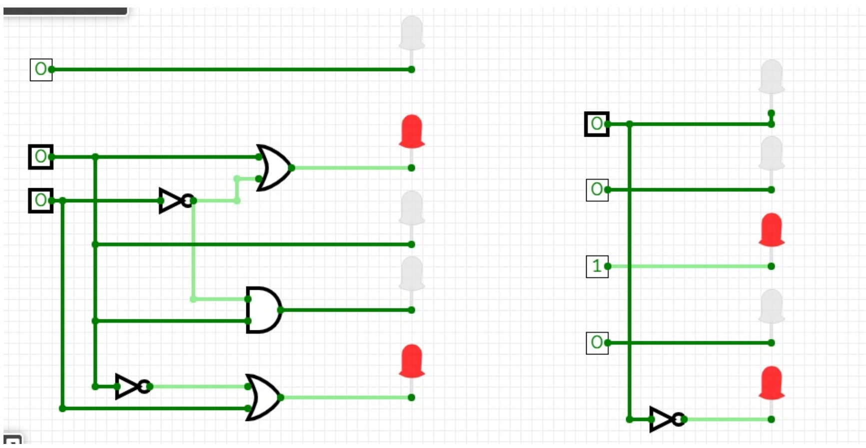
```
module tb_de;

  reg clk;
  reg [3:0] morse_in;
  reg [2:0] num;
  wire [4:0] out;

  // Instantiate the JK flip-flop module
  de my_de (
    .out(out),
    .num(num),
    .morse_in(morse_in),
    .clk(clk)
  );
```

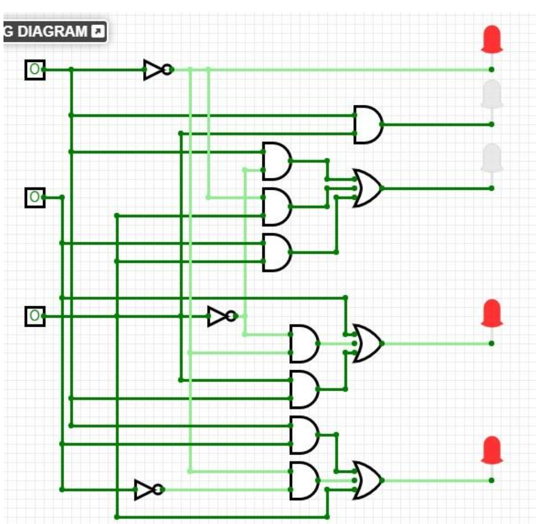
```
  // Clock generation
  initial begin
    clk = 1;
    forever #5 clk = ~clk;  // Toggle the clock every 5 time units
  end

  // Test scenario
  initial begin
    // Apply inputs and observe outputs for a few clock cycles
        num = 3'b100; morse_in = 4'b1010;   //C
    #10 num = 3'b010; morse_in = 2'b00;        //I
    #10 num = 3'b011; morse_in = 3'b010;      //R
    #10 num = 3'b100; morse_in = 4'b1010;    //C
    #10 num = 3'b011; morse_in = 3'b001;      //U
    #10 num = 3'b010; morse_in = 2'b00;        //I
    #10 num = 3'b001; morse_in = 1'b1;          //T
    #10
    #10 $finish;
  end
  initial
    begin
      $dumpfile("dump.vcd"); $dumpvars;
    end
endmodule
```



Output:

GATE-LEVEL





5. Encoder Implementation

HIGH-LEVEL:

Code:

```
module alphaROM (out1, out2, out3, out4, in_morse_bit, alpha_in, clk);
  input clk;
  input [1:0] in_morse_bit;
  input [4:0] alpha_in;
  output reg out1;tput reg [2:0] out3;
  output reg [3:0] out4;

  reg rom_data1[1:0];
  reg [1:0] rom_data2 [3:0];
  reg [2:0] rom_data3 [7:0];
  reg [3:0] rom_data4 [11:0];

  parameter A = 5'b00001; // A
  parameter B = 5'b00010; // B
  parameter C = 5'b000
  output reg [1:0] out2;
  ou 11; // C
  parameter D = 5'b00100; // D
  parameter E = 5'b00101; // E
  parameter F = 5'b00110; // F
  parameter G = 5'b00111; // G
  parameter H = 5'b01000; // H
```

```
parameter I = 5'b01001; // I

parameter J = 5'b01010; // J

parameter K = 5'b01011; // K

parameter L = 5'b01100; // L

parameter M = 5'b01101; // M

parameter N = 5'b01110; // N

parameter O = 5'b01111; // O

parameter P = 5'b10000; // P

parameter Q = 5'b10001; // Q

parameter R = 5'b10010; // R

parameter S = 5'b10011; // S

parameter T = 5'b10100; // T

parameter U = 5'b10101; // U

parameter V = 5'b10110; // V

parameter W = 5'b10111; // W

parameter X = 5'b11000; // X

parameter Y = 5'b11001; // Y

parameter Z = 5'b11010; // Z


initial begin

  rom_data1[0] = 1'b0; // E (dot)

  rom_data1[1] = 1'b1; // T (dash)

end


initial begin

  rom_data2[0] = 2'b01; // A (dot dash)

  rom_data2[1] = 2'b00; // I (dot dot)
```

```
  rom_data2[2] = 2'b11; // M (dash dash)
  rom_data2[3] = 2'b10; // N (dash dot)
end

initial begin
  rom_data3[0] = 3'b100; // D (dash dot dot)
  rom_data3[1] = 3'b110; // G (dash dash dot)
  rom_data3[2] = 3'b101; // K (dash dot dash)
  rom_data3[3] = 3'b111; // O (dash dash dash)
  rom_data3[4] = 3'b010; // R (dot dash dot)
  rom_data3[5] = 3'b000; // S (dot dot dot)
  rom_data3[6] = 3'b011; // W (dot dash dash)
  rom_data3[7] = 3'b001; // U (dot dot dash)
end

initial begin
  rom_data4[0] = 4'b1000; // B (dash dot dot dot)
  rom_data4[1] = 4'b1010; // C (dash dot dash dot)
  rom_data4[2] = 4'b0010; // F (dot dot dash dot)
  rom_data4[3] = 4'b0000; // H (dot dot dot dot)
  rom_data4[4] = 4'b0111; // J (dot dash dash dash)
  rom_data4[5] = 4'b0100; // L (dot dash dot dot)
  rom_data4[6] = 4'b0110; // P (dot dash dash dot)
  rom_data4[7] = 4'b1101; // Q (dash dash dot dash)
  rom_data4[8] = 4'b0001; // V (dot dot dot dash)
  rom_data4[9] = 4'b1001; // X (dash dot dot dash)
  rom_data4[10] = 4'b1011; // Y (dash dot dash dash)
```

```
  rom_data4[11] = 4'b1100; // Z (dash dash dot dot)
end

always @(posedge clk) begin
  if (in_morse_bit == 2'b00)
    case (alpha_in)
      E: out1 = rom_data1[0];
      T: out1 = rom_data1[1];
      //default: out = 1'b0;
    endcase
  else if (in_morse_bit == 2'b01)
    case (alpha_in)
      A: out2 = rom_data2[0];
      I: out2 = rom_data2[1];
      M: out2 = rom_data2[2];
      N: out2 = rom_data2[3];
      //default: out = 2'b00;
    endcase
  else if (in_morse_bit == 2'b10)
    case (alpha_in)
      D: out3 = rom_data3[0];
      G: out3 = rom_data3[1];
      K: out3 = rom_data3[2];
      O: out3 = rom_data3[3];
      R: out3 = rom_data3[4];
      S: out3 = rom_data3[5];
      W: out3 = rom_data3[6];
```

```
      U: out3 = rom_data3[7];
      //default: out = 3'b000;
    endcase
  else if (in_morse_bit == 2'b11)
    case (alpha_in)
      B: out4 = rom_data4[0];
      C: out4 = rom_data4[1];
      F: out4 = rom_data4[2];
      H: out4 = rom_data4[3];
      J: out4 = rom_data4[4];
      L: out4 = rom_data4[5];
      P: out4 = rom_data4[6];
      Q: out4 = rom_data4[7];
      V: out4 = rom_data4[8];
      X: out4 = rom_data4[9];
      Y: out4 = rom_data4[10];
      Z: out4 = rom_data4[11];
      //default: out = 4'b0000;
    endcase
 end
endmodule

module MorseCodeEncoder (alpha_in, num, clk, out1, out2, out3, out4);
 input clk;
 input [4:0] alpha_in;
 input [1:0] num;
 output out1;
```

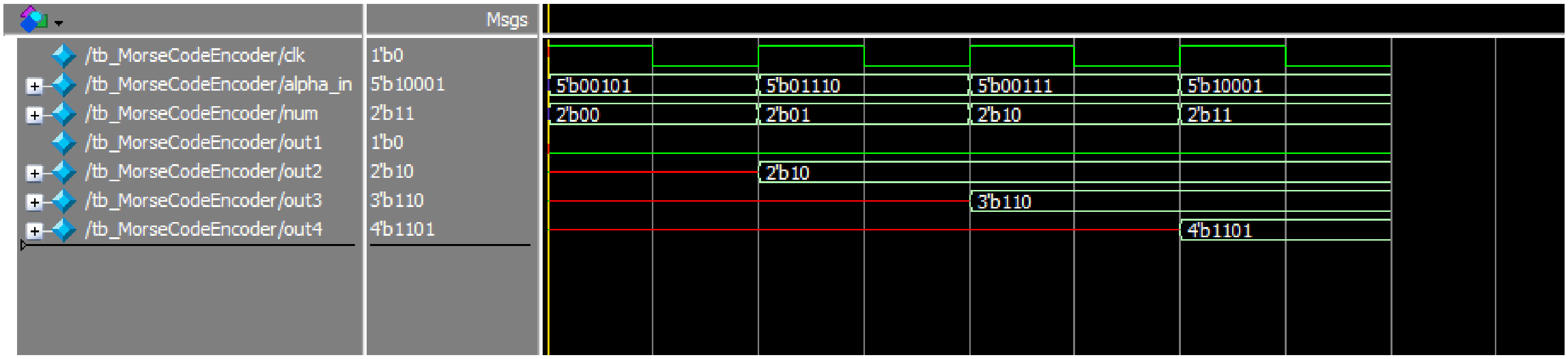
```
  output [1:0] out2;
  output [2:0] out3;
  output [3:0] out4;
  alphaROM r (out1, out2, out3, out4, num, alpha_in, clk);
endmodule
```

Test Bench:

```
module tb_MorseCodeEncoder;
  reg clk;
  reg [4:0] alpha_in;
  reg [1:0] num;
  wire out1;
  wire [1:0] out2;
  wire [2:0] out3;
  wire [3:0] out4;

  MorseCodeEncoder dut (
    .alpha_in(alpha_in),
    .num(num),
    .clk(clk),
    .out1(out1),
    .out2(out2),
    .out3(out3),
    .out4(out4)
  );

  // Clock generation
  initial begin
```

```
    clk = 1;
    forever #5 clk = ~clk;  // Toggle the clock every 5 time units
  end

  // Test case 1
  initial begin
    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
initial
    begin
      $dumpfile("dump.vcd");$dumpvars;
    end
```

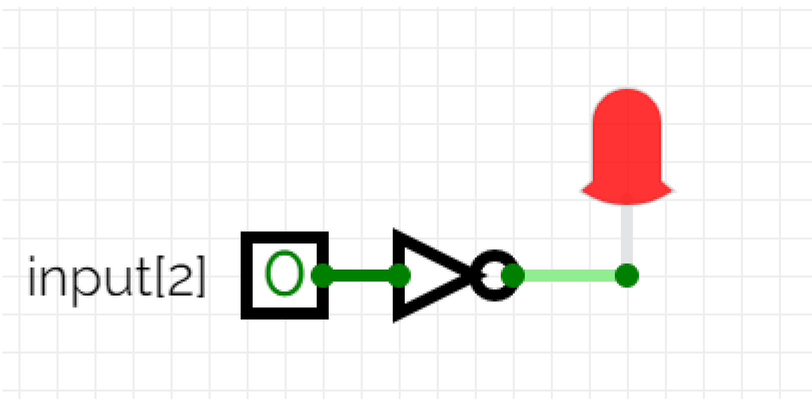
endmodule



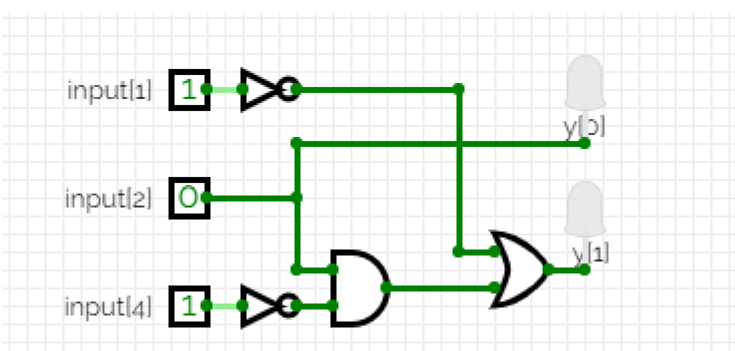
Output:

GATE-LEVEL

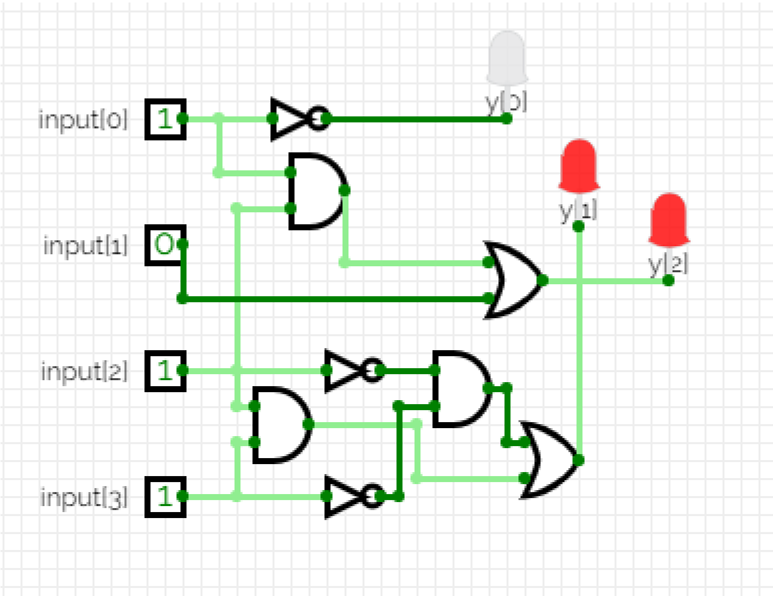
For 1-bit morse code:



For 2-bit morse code:



For 3-bit morse code:



CONCLUSION:

In summary, we've advanced well with the Morse Code project. We've crafted the primary code using Verilog and independently tested each part. Additionally, we've conducted circuit simulations for individual components. Moving forward, our next steps involve developing Verilog code for gate-level implementation. We'll then integrate circuit components to perform a comprehensive simulation of the entire circuit, bringing us closer to project completion.