Abstract

This project implements a dual-mode Morse Code Encoder-Decoder system using a CPLD board. The encoder sequentially displays Morse code for decimal digits (0-9) using a finite state machine (FSM), while the decoder translates Morse code inputs into corresponding alphabets on a 7-segment display. The system effectively demonstrates the use of D flip-flops, combinational logic, and 7-segment interfacing in digital systems, all within limited hardware resources.

Objective

To design and implement a CPLD-based system that:

1. Encodes decimal digits (0–9) into Morse code using a state machine.
2. Decodes user-input Morse code into the alphabet on a 7-segment display.

Apparatus:

1. CPLD Board
2. Bread Board
3. 7-segment display
4. Jumper Cables
5. Quartus || 64 bit software

Theory:

Morse code is a method of encoding textual information using a system of standardized sequences composed of short and long signals, commonly referred to as dots (·) and dashes (–). Each letter of the alphabet, numeral, and certain punctuation marks are represented by a unique combination of these two basic signal units.

Originally developed in the 1830s and 1840s by Samuel Morse and Alfred Vail for use in telegraphy, Morse code enabled long-distance communication over wires using electric pulses. A dot corresponds to a short pulse, while a dash represents a longer pulse. The timing between elements (dots and dashes), letters, and words is also standardized to ensure clarity in transmission.

Morse code played a crucial role in early radio communications, maritime signaling, and military operations, and it remains a valuable tool in aviation, amateur radio, and emergency signaling due to its simplicity and effectiveness in low-bandwidth or noisy environments.

In digital systems like ours, Morse code can be emulated using binary signals—assigning 0 to represent a dot and 1 to represent a dash—making it suitable for implementation on programmable logic devices such as CPLDs.

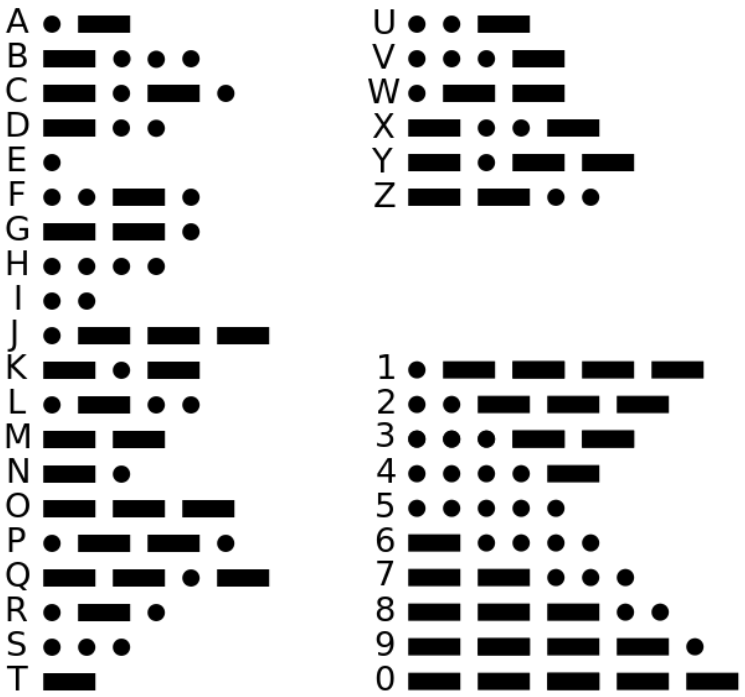


Fig. A typical morse code translator using dots and dash

Working Mechanism:

Input and output pins:

1. Number of inputs from CPLD board = 8
2. Number of outputs = 12
3. Switches marked with ‘2’ are used to give input from the CPLD board. Output for the digit sequence generator is obtained from the LEDs present on the CPLD board marked with ‘3’.
4. The output for the 7-segment display is obtained from the 7 pins marked with ‘1’.

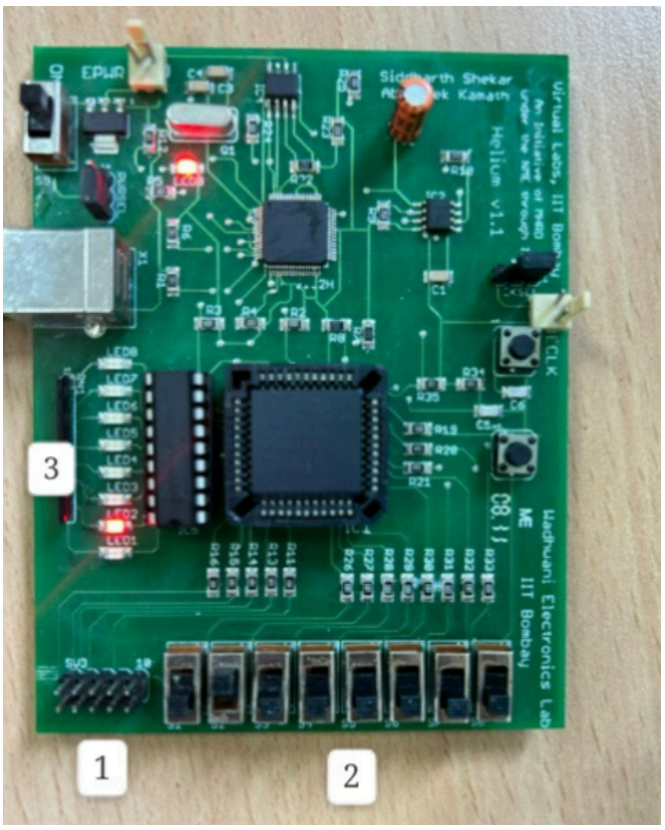


Fig. A typical CPLD board

Our project implements a Morse Code Encoder-Decoder system with two distinct modes of operation, controlled via switches SW1 and SW2 on the CPLD board. The mode selection is as follows:

- **SW1 SW2 = 01**: Morse Code Decoder Mode
- **SW1 SW2 = 10**: Morse Code Digit Sequence Generator Mode
- **SW1 SW2 = 00 or 11**: Reset mode (all the outputs are 0)

1. Morse Code Digit Sequence Generator Mode:

In this mode, the system sequentially generates and displays the Morse code for decimal digits 0 to 9. This is achieved using a Finite State Machine (FSM) designed using four D flip-flops (Q0–Q3), which act as state registers. The FSM operates as follows:

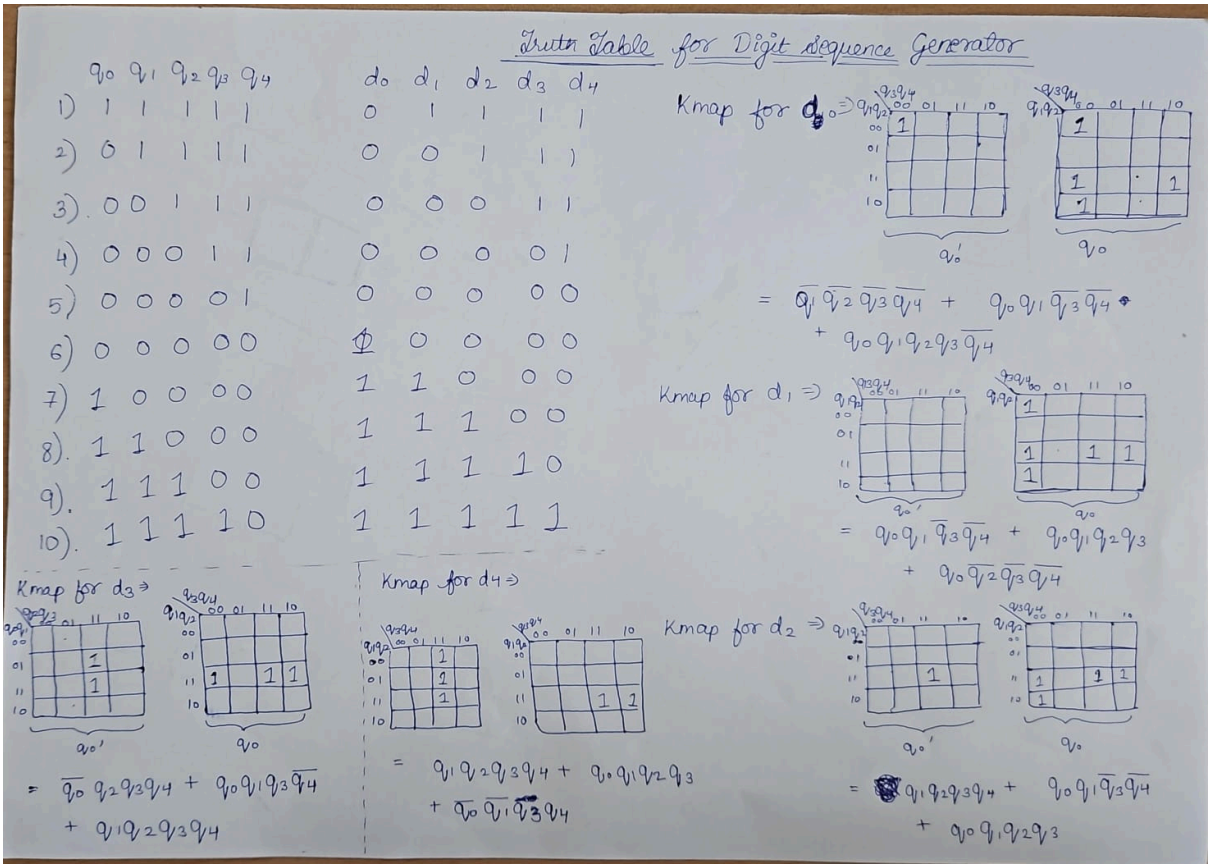
- **States:** Each state of the FSM represents one of the ten decimal digits (0–9).

- **State Transitions:** For each desired output state we first designed a truth table where the current state (d0, d1, d2, d3, d4) is achieved by a combination of the outputs (q0, q1, q2, q3, q4) of the previous state. We start with the morse code representation of 1 and count till 9 then 0 then again 1 and so on we are able to represent the morse code representation of the digits 0-9 sequentially. From the truth table we write each output state (d0 - d4) as a function of the previous states (q0 - q4) and simplify them using Kmaps.

- **Output Logic:** The corresponding **Morse code** is generated for each digit (state) and output **bit-by-bit** using built-in LEDs on the CPLD board. Here:

- A 0 represents a **dot (·)**.
- A 1 represents a **dash (–)**.

- **Timing Control:** The Morse code for each digit is shown sequentially, with each bit displayed per clock cycle, providing a stepwise visual representation of the digit's Morse code before transitioning to the next digit.



Truth table for digit sequence generator

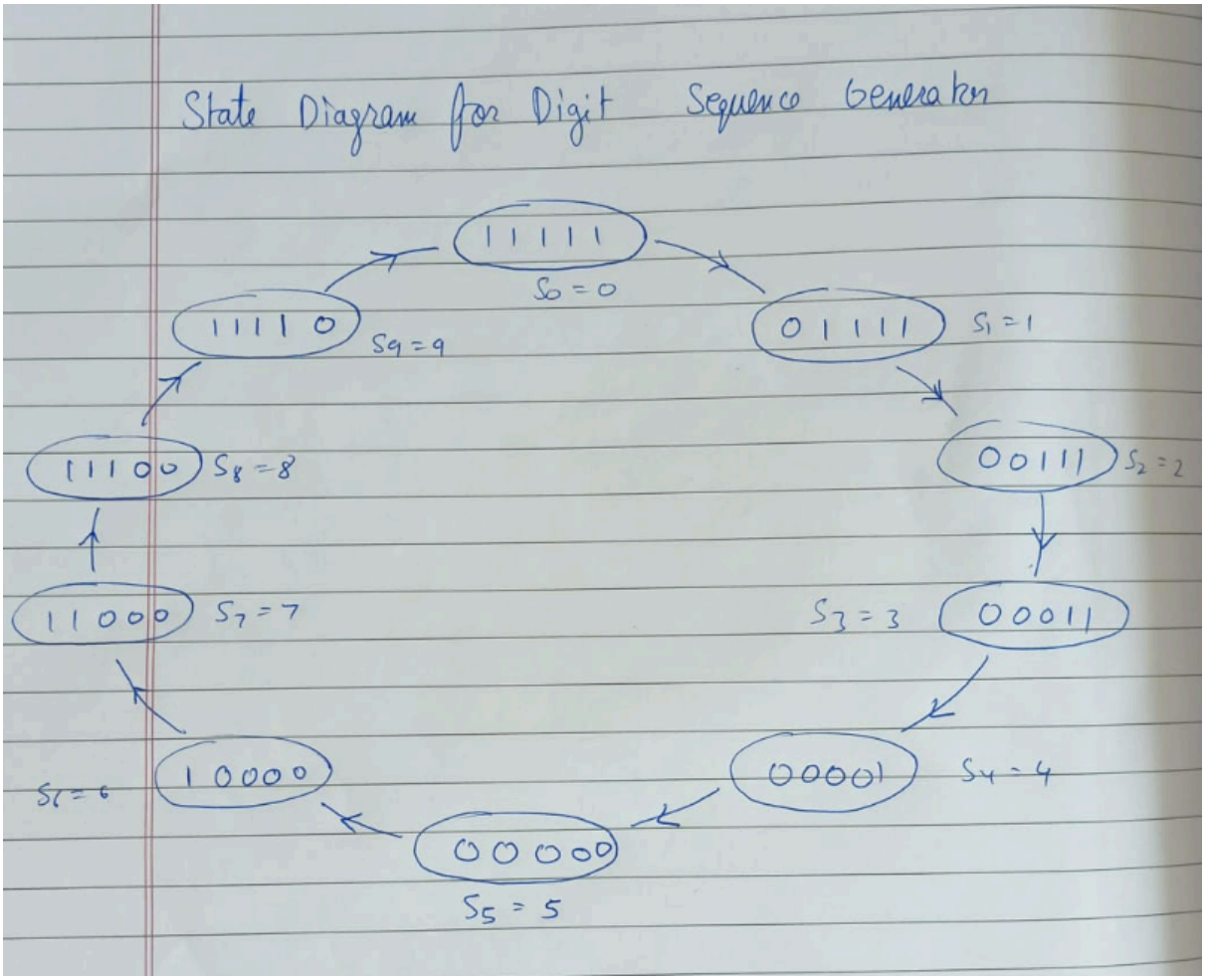


Fig. State diagram of Digit sequence generator

2. Morse Code Decoder Mode

In this mode, users can decode Morse code inputs into the corresponding alphabets. Due to switch limitations on the CPLD board, the following compact input scheme is used:

- **Length Input**: The user specifies the length of the Morse code (ranging from 1 to 4) using the last two switches. Since a 2-bit binary input can represent values from 0 to 3, the user inputs **(length - 1)**.

Morse Code Input: The following four switches (SW5–SW8) are used to input the Morse code itself, with the encoding 0 for '**.**' and 1 for dash '**_**'

The system processes the input and displays the corresponding alphabet on the 7-segment display. This input design was chosen to balance functionality with hardware constraints on the number of available switches.

7-segment display: We have used a common anode 7-segment display to implement the decoder. A common anode 7-segment display is a type of display where all the anodes of the LEDs are connected to a common terminal, while the cathodes are left free. To illuminate

a segment, the corresponding cathode must be connected to a logic '0' (ground), while the common anode is connected to a logic '1' (high voltage).

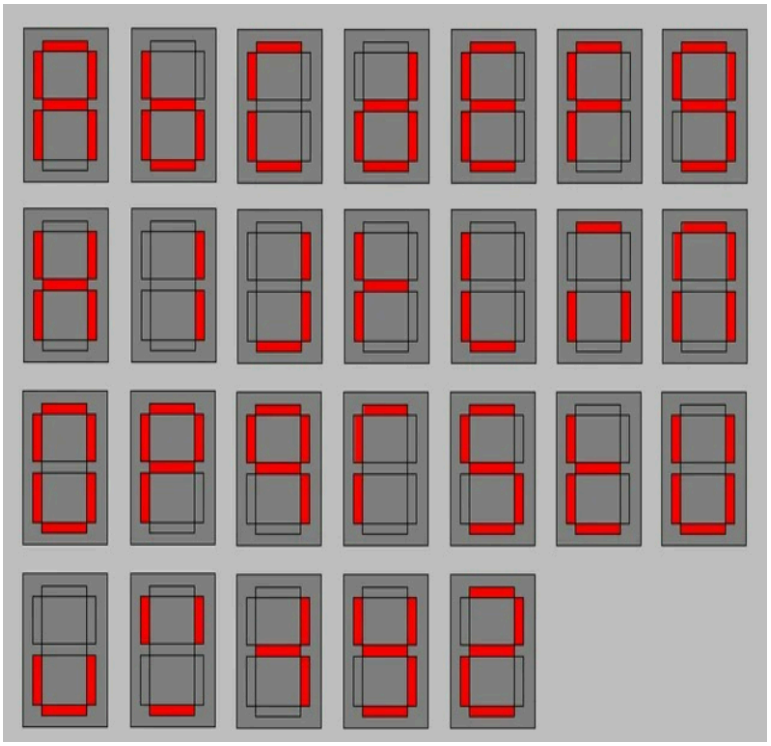
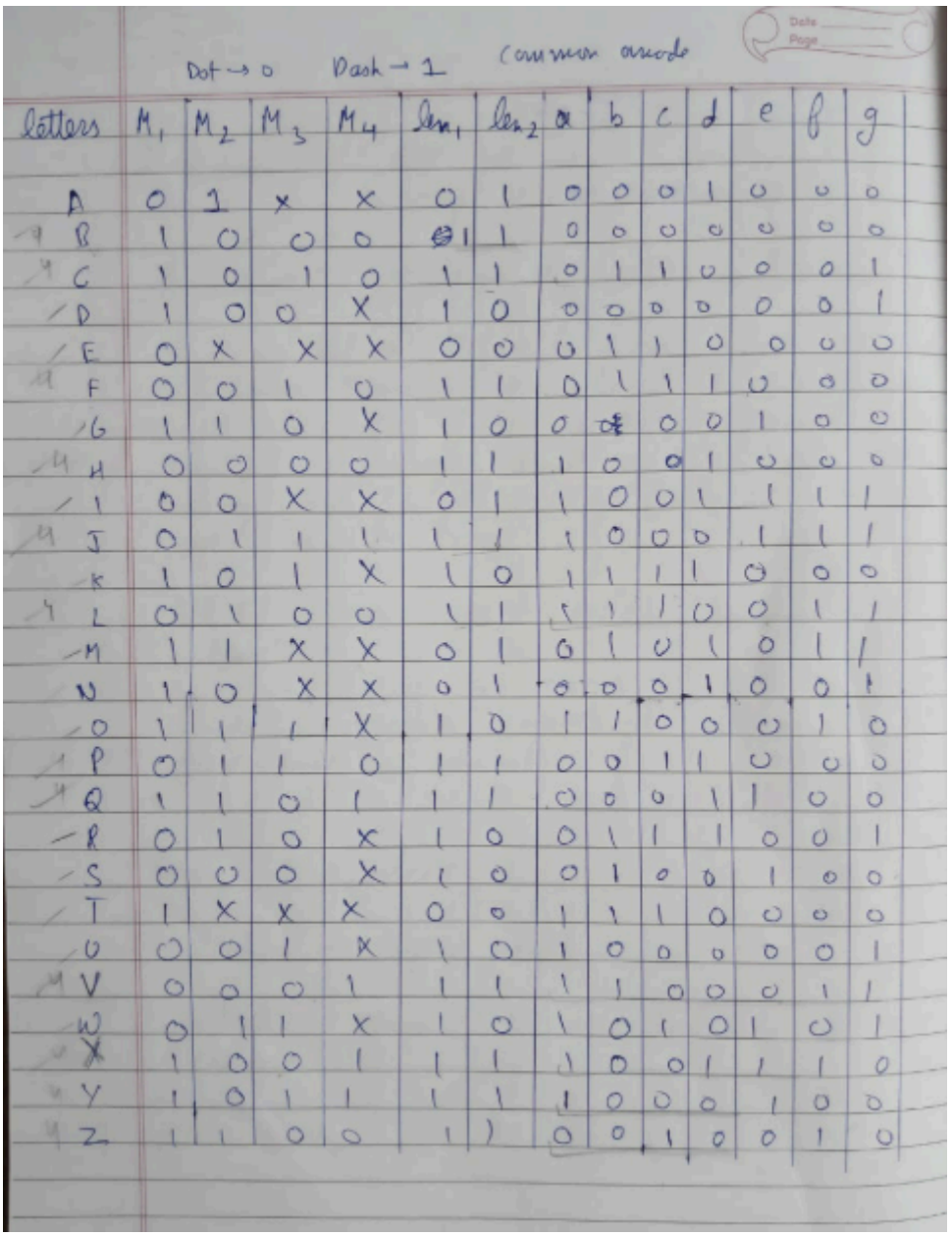


Fig. This image is used as a reference to show all the alphabets on the 7-segment display.



Dot → 0 Dash → 1 (common anode

letters	M_1	M_2	M_3	M_4	len_1	len_2	a	b	c	d	e	f	g
A	0	1	x	x	0	1	0	0	0	1	0	0	0
B	1	0	0	0	1	1	0	0	0	0	0	0	0
C	1	0	1	0	1	1	0	1	1	0	0	0	1
D	1	0	0	X	1	0	0	0	0	0	0	0	1
E	0	X	X	X	0	0	0	1	1	0	0	0	0
F	0	0	1	0	1	1	0	1	1	1	0	0	0
G	1	1	0	X	1	0	0	0	0	0	1	0	0
H	0	0	0	0	1	1	1	0	0	1	0	0	0
I	0	0	X	X	0	1	1	0	0	1	1	1	1
J	0	1	1	1	1	1	1	0	0	0	1	1	1
K	1	0	1	X	1	0	1	1	1	1	0	0	0
L	0	1	0	0	1	1	1	1	1	0	0	1	1
M	1	1	X	X	0	1	0	1	0	1	0	1	1
N	1	0	X	X	0	1	0	0	0	1	0	0	1
O	1	1	1	X	1	0	1	1	0	0	0	1	0
P	0	1	1	0	1	1	0	0	1	1	0	0	0
Q	1	1	0	1	1	1	0	0	0	1	1	0	0
R	0	1	0	X	1	0	0	1	1	1	0	0	1
S	0	0	0	X	1	0	0	1	0	0	1	0	0
T	1	X	X	X	0	0	1	1	1	0	0	0	0
U	0	0	1	X	1	0	1	0	0	0	0	0	1
V	0	0	0	1	1	1	1	1	0	0	0	1	1
W	0	1	1	X	1	0	1	0	1	0	1	0	1
X	1	0	0	1	1	1	1	0	0	1	1	1	0
Y	1	0	1	1	1	1	1	0	0	0	1	0	0
Z	1	1	0	0	1	1	0	0	1	0	0	1	0

Fig. Truth table for morse code decoder with 6 inputs and 7 outputs

Circuit:

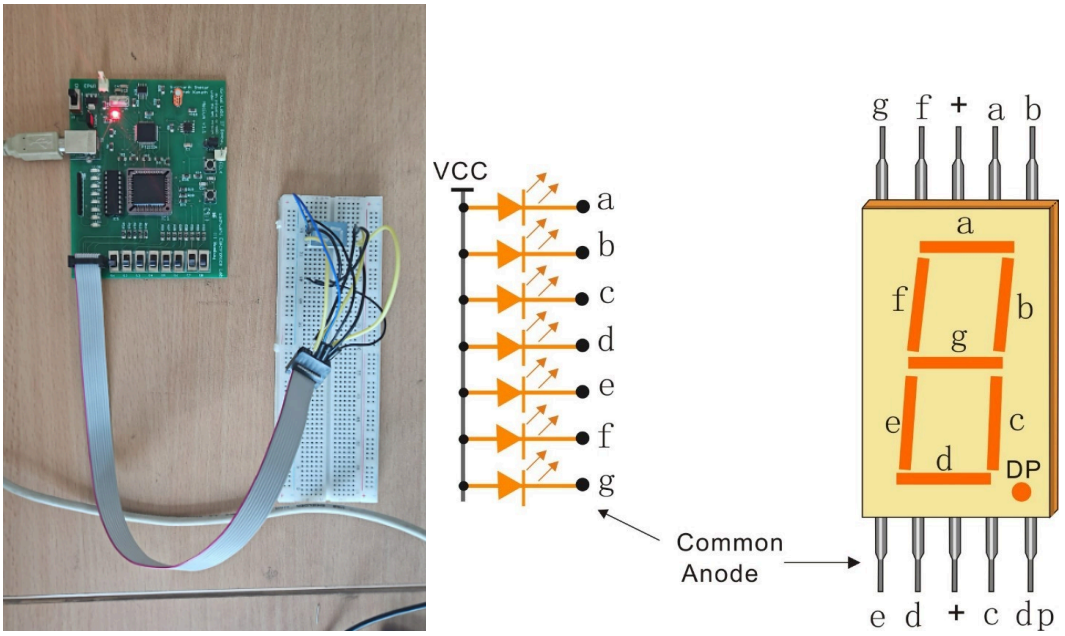


Fig. Circuit and 7-segment pin diagram

Verilog Code:

```
module prj2 (
    input clk,
    input s1,
    input s2,
    input m1,
    input m2,
    input m3,
    input m4,
    input len1,
    input len2,
    output q0,
    output q1,
    output q2,
    output q3,
    output q4,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g
);

    wire [4:0] counter_q;
    wire [6:0] morse_seg;

    counter u_counter(
```

```
        .clk(clk),
        .q0(counter_q[0]),
        .q1(counter_q[1]),
        .q2(counter_q[2]),
        .q3(counter_q[3]),
        .q4(counter_q[4])
    );


    morse_combinational u_morse(
        .m1(m1),
        .m2(m2),
        .m3(m3),
        .m4(m4),
        .len1(len1),
        .len2(len2),
        .seg(morse_seg)
    );


    assign {q0, q1, q2, q3, q4} = (s1 && ~s2) ? counter_q : 5'b00000;
    assign {a, b, c, d, e, f, g} = (~s1 && s2) ? morse_seg :
7'b1111111;


endmodule

module counter(
    input clk,
    output q0,
    output q1,
    output q2,
    output q3,
    output q4
);
    wire [4:0] d;

    d_ff d1(.d(d[0]), .clk(clk), .q(q0));
    d_ff d2(.d(d[1]), .clk(clk), .q(q1));
    d_ff d3(.d(d[2]), .clk(clk), .q(q2));
    d_ff d4(.d(d[3]), .clk(clk), .q(q3));
    d_ff d5(.d(d[4]), .clk(clk), .q(q4));

    assign d[0] = (~q1 & ~q2 & ~q3 & ~q4) | (q0 & q1 & ~q3 & ~q4) | (q0
& q1 & q2 & q3 & ~q4);
    assign d[1] = (q0 & q1 & ~q3 & ~q4) | (q0 & q1 & q2 & q3) | (q0 &
~q2 & ~q3 & ~q4);
    assign d[2] = (q1 & q2 & q3 & q4) | (q0 & q1 & ~q3 ~q4) | (q0 & q1
& q2 & q3);
    assign d[3] = (~q0 & q2 & q3 & q4) | (q0 & q1 & q3 & ~q4) | (q1 &
q2 & q3 & q4);
    assign d[4] = (q1 & q2 & q3 & q4) | (q0 & q1 & q2 & q3) | (~q0 &
~q1 & q3 & q4);
```

```
endmodule

module morse_combinational(
    input m1,
    input m2,
    input m3,
    input m4,
    input len1,
    input len2,
    output reg [6:0] seg
);
    always @(*) begin
        if (~len1 & ~len2) begin
            if (~m1 & ~m2 & ~m3 & ~m4) begin // E
                seg = 7'b0110000;
            end else if (m1&(~m2)&(~m3)&(~m4)) begin // T
                seg = 7'b1110000;
            end else begin
                     seg = 7'b1111111;
                end
        end else if (~len1 & len2) begin
            if (~m1 & ~m2 & ~m3 & ~m4) begin // i
                seg = 7'b1001111;
            end else if (m1 & ~m2 & ~m3 & ~m4) begin // n
                seg = 7'b0001001;
            end else if (m1 & m2 & ~m3 & ~m4) begin // M
                seg = 7'b0101011;
            end else if (~m1 & m2 & ~m3 & ~m4) begin // A
                seg = 7'b0001000;
            end else begin
                seg = 7'b1111111; // Invalid
            end
        end else if (len1 & ~len2) begin
            if (m1 & ~m2 & ~m3 & ~m4) begin // D
                seg = 7'b0000001;
            end else if (m1 & m2 & ~m3 & ~m4) begin // G
                seg = 7'b0000100;
            end else if (m1 & ~m2 & m3 & ~m4) begin // K
                seg = 7'b1111000;
            end else if (m1 & m2 & m3 & ~m4) begin // O
                seg = 7'b1100010;
            end else if (~m1 & m2 & ~m3 & ~m4) begin // R
                seg = 7'b0111001;
            end else if (~m1 & ~m2 & ~m3 & ~m4) begin // S
                seg = 7'b0100100;
            end else if (~m1 & ~m2 & m3 & ~m4) begin // U
                seg = 7'b1000001;
            end else if (~m1 & m2 & m3 & ~m4) begin // W
                seg = 7'b1010101;
            end else begin
                seg = 7'b1111111; // Invalid
            end
```

```
        end else begin
            if (m1 & ~m2 & ~m3 & ~m4) begin // D
                seg = 7'b0000000;
            end else if (m1 & ~m2 & m3 & ~m4) begin // C
                seg = 7'b0110001;
            end else if (~m1 & ~m2 & m3 & ~m4) begin // F
                seg = 7'b0111000;
            end else if (~m1 & ~m2 & ~m3 & ~m4) begin // h
                seg = 7'b1001000;
            end else if (~m1 & m2 & m3 & m4) begin // j
                seg = 7'b1000111;
            end else if (~m1 & m2 & ~m3 & ~m4) begin // l
                seg = 7'b1110011;
            end else if (~m1 & m2 & m3 & ~m4) begin // p
                seg = 7'b0011000;
            end else if (m1 & m2 & ~m3 & m4) begin // q
                seg = 7'b0001100;
            end else if (~m1 & ~m2 & ~m3 & m4) begin // v
                seg = 7'b1100011;
            end else if (m1 & ~m2 & ~m3 & m4) begin // x
                seg = 7'b1001110;
            end else if (m1 & ~m2 & m3 & m4) begin // y
                seg = 7'b1000100;
            end else if (m1 & m2 & ~m3 & ~m4) begin // z
                seg = 7'b0010010;
            end else begin
                seg = 7'b1111111; // Invalid
            end
        end
    end
endmodule

module d_ff(
    input d,
    input clk,
    output reg q
);
    always @(posedge clk) begin
        q <= d;
    end
Endmodule
```

Simulation:

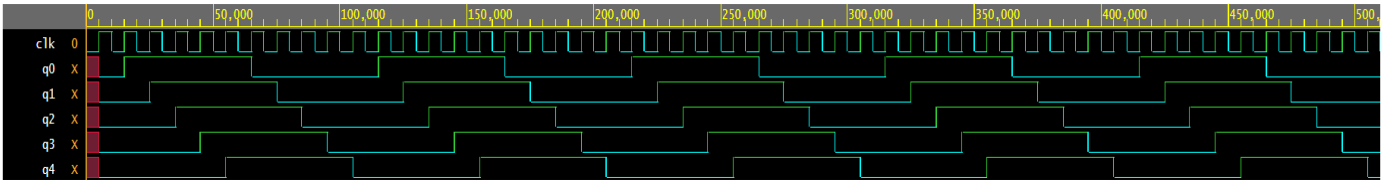


Fig. Simulation of waveforms of digital sequence encoder.

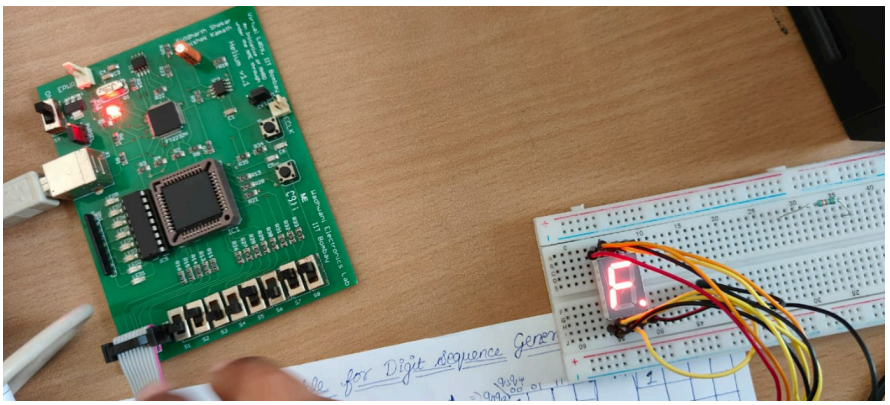
Testbench code for Digital sequence encoder:

```
`timescale 1ns / 1ps
module top_module_tb;
    // Inputs
    reg clk;
    wire q0, q1, q2, q3, q4;

    // Instantiate the Unit Under Test (UUT)
    top_module uut (
        .clk(clk),
        .q0(q0),
        .q1(q1),
        .q2(q2),
        .q3(q3),
        .q4(q4)
    );
    // Clock generation
    always #5 clk = ~clk;  // 10ns period (100MHz)

    // Initialize variables and run simulation
    initial begin

        clk = 0;
        // Wait for initialization
        #10;
        #500;
        $finish;
    end

    // Monitor the outputs
    initial begin
        $monitor("Time=%0t | clk=%b | State: q0=%b q1=%b q2=%b q3=%b q4=%b |
Binary=%b%b%b%b%b",
                 $time, clk, q0, q1, q2, q3, q4, q0, q1, q2, q3, q4);
    end

    initial begin
        $dumpfile("top_module_tb.vcd");
        $dumpvars(0, top_module_tb);
    end
endmodule
```

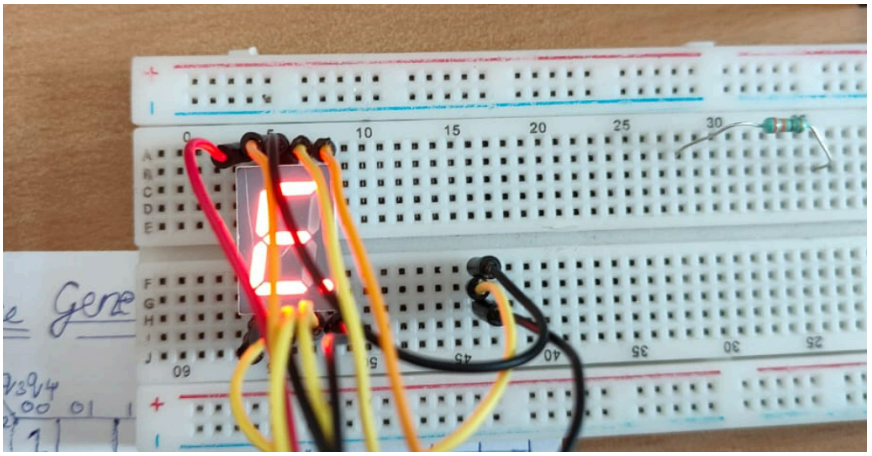
Output:

Link to demonstration video for Morse code sequence:: 1234demo.mp4

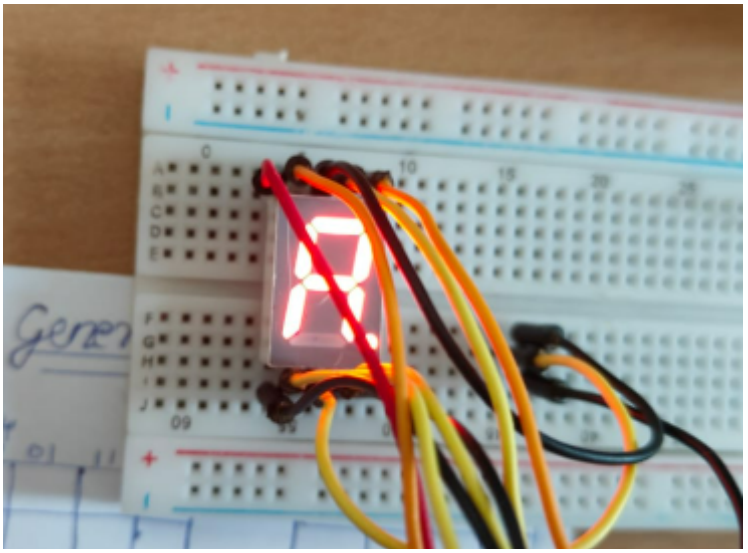
Link to demonstration video for Morse code to alphabet decoder: Table_20_Project.mp4



Letter F (when input was Morse code for F - 001011)

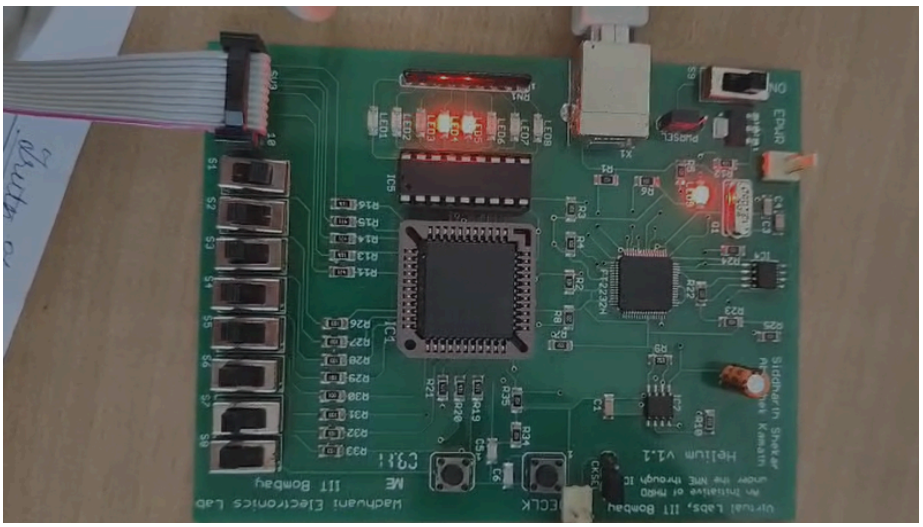


Letter E (when input was Morse code for E - 000000)



Letter A (when input was Morse code for A - 010001)

Fig. (1-3) Working of the Morse code decoder using a 7-segment display



Glimpse of the working of the Morse Code digit sequence generator

Inference:

Through the implementation of this project, we have effectively demonstrated the use of digital logic design principles to emulate a Morse code communication system on a CPLD board. By utilising finite state machines, D flip-flops, and combinational logic derived from Karnaugh maps, we could sequentially generate and visually represent Morse codes for decimal digits (0–9) precisely and reliably. The decoder module, which interprets Morse code input for alphabets using a compact switch-based interface, illustrates the feasibility of translating symbolic binary input into meaningful human-readable outputs using programmable logic.

Furthermore, the project showcases the versatility of the CPLD platform in supporting dual-mode operations with minimal hardware resources. The system reinforces a fundamental understanding of state transitions, binary encoding, and display interfacing by implementing both encoder and decoder functionalities.

Conclusion:

The successful completion of this Morse Code Encoder-Decoder project demonstrates our ability to apply theoretical digital logic design concepts in a practical setting and work within the constraints of real-world hardware. By splitting the system into two functional modes, we explored the sequential nature of digit-based encoding using FSMS and the conditional decoding of Morse signals into alphanumeric characters.

Despite limited input resources, the decoder mode enables efficient interpretation of Morse symbols by compactly mapping binary patterns to characters. In contrast, the digit generator mode showcases the power of state-driven control logic in generating structured sequential outputs. These modes highlight how minimal hardware can be orchestrated through well-designed digital logic to achieve versatile and educationally valuable functionality.

This project emphasises how legacy communication systems like Morse code can still be instructive in modern engineering contexts, especially in the design of compact, low-bandwidth digital communication tools. Furthermore, the project highlights how CPLDS

and hardware description languages like Verilog can be powerful tools for prototyping and educational applications.

Sources of Error:

1. Faulty components, such as the integrated circuit (IC) or LED display, and loose connections could impair the circuit's functionality.
2. Malfunctioning CPLD device.

Precautions:

1. Simulate the code first on an online software like EDA Playground before moving to its implementation on the CPLD board.
2. Run the test file in CPLD and check if all LEDs are functioning properly before running the code.