FPGA BASED MORSE CODE DECODER USING DE1 SOC BOARD

Digital Design using FPGA Term Project report
Submitted in Partial Fulfillment of the Requirements for the Degree of
MASTER OF TECHNOLOGY
in
SIGNAL PROCESSING AND MACHINE LEARNING
by

Dasari Revanth Roy (232SP005)
Jaydeep Rao(232SP009)
Yogesh Raju N R(232SP029)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575025
December 2022

CERTIFICATE

This is to certify that the project report entitled "FPGA based Morse Code decoder using DE1 SOC Board" submitted by Dasari Revanth Roy(232SP005),Jaydeep Rao (232SP009) and Yogesh Raju N R(232SP029) as a record of the Digital Design using FPGA course work, presented by them is accepted as the Project Report Submission in partial fulfillment of the requirements for the award of Master of Technology (Signal Processing and Machine Learning) in the Department of Electronics and Communication Engineering, National Institute of Technology Karnataka, Surathkal.

Course Coordinator
(Dr. Sumam David)

ABSTRACT

Morse code is a method of communication that uses sequences of dots and dashes to represent letters, numerals, and punctuation marks. It was developed by Samuel Morse and Alfred Vail in the 1830s for use with early telegraph systems. Morse code assigns a unique combination of short and long signals, typically represented as dots (.) and dashes (-), to each letter and character in the alphabet.

Each letter, numeral, or punctuation mark is represented by a unique sequence of dots and dashes.

Morse code, was initially used in telegraph systems, Morse code was crucial in early long-distance communication. It has also been employed in aviation, radio communication, and as an emergency signaling system due to its simplicity and versatility. Although largely replaced by more modern communication technologies, Morse code remains significant in certain niche areas and is still taught and used by amateur radio enthusiasts and some military services.

Learning and decoding Morse code involves understanding the sequences of dots and dashes that represent each character, enabling individuals to communicate messages using this system of encoding text.

This report aims to describe the implementation of Morse code decoder on the DE1 SoC FPGA board using external peripherals. The morse code decoder is realized with the help of DE-01 SOC FPGA board, the morse code is given as input to the FPGA board using the slide switches and the equivalent converted text is generated at the external peripheral. The external peripheral used is the monitor, controlled using the VGA port, to display the converted text.

Table of Contents

1. INTRODUCTION 4
2. METHODOLOGY 5
   2.1. COMPONENTS 6
   2.2. PERIPHERALS USED 7
   2.3. WORKING 10
   2.4. SOFTWARE MODULES 12
3. RESULT 16
4. CONCLUSION 21
REFERENCES

List Of Figures

Figure 1 – International Morse code representation 5
Figure 2 - DE1-SoC development board 6
Figure 3 – DE1 slide switches and push buttons 7
Figure 4 – 7-segment display 7
Figure 5 - Connection between FPGA and VGA 8
Figure 6 - VGA horizontal timing specification 9
Figure 7 – Peripheral Interface 10
Figure 8 – Experimental Setup 11
Figure 9 – Device Utilization Statistics 16
Figure 11 – 7-segment display showing alphabet 'N' 17
Figure 10 – VGA Interfaced Monitor Showing 'N' 17
Figure 12 - 7-segment display showing alphabet 'I' 18
Figure 13 - VGA Interfaced Monitor Showing 'I' 18
Figure 14 - 7-segment display showing alphabet 'T' 19
Figure 15- VGA Interfaced Monitor Showing 'T' 19
Figure 16-7-segment display showing alphabet 'K' 20
Figure 17 - VGA Interfaced Monitor Showing 'K' 20

1. INTRODUCTION

Morse code is a communication language used for transmitting large memory size data to long distances without incurring any loss in information. It is combination of dots and dashes, various combinations of dots and dashes are mapped to any alphabet, letter, number or symbol.

Since, morse code uses only dot's and dashes the memory size used to store an piece of information gets drastically reduced, this makes a space for application of morse code in transmitting large information to long distances without any data loss or any crucial information loss.

The data gets compressed and thus the memory requirement is also less, this saves costs for transmission and storage of data.

Morse code also finds applications in Military field for sending confidential information to other unions, states or territories. Since morse code is a unique language and can communicate anything via dots and dashes, it finds application in various other stations like. telecommunication,

This project focuses on decoding the morse information to text using Verilog language. This module will be very helpful to decode as well as understand how the Morse code works.

2. METHODOLOGY

The morse code is a communication language, which has the capability to compress large data to consume less memory, making it compatible to transmit huge data to various destinations.

In Morse code, each letter of the alphabet, as well as numbers and punctuation marks, is represented by a unique sequence of dots and dashes. The code is structured so that more frequently used letters have shorter codes, while less frequently used letters have longer ones

Fig.1 shows the international morse code representation.

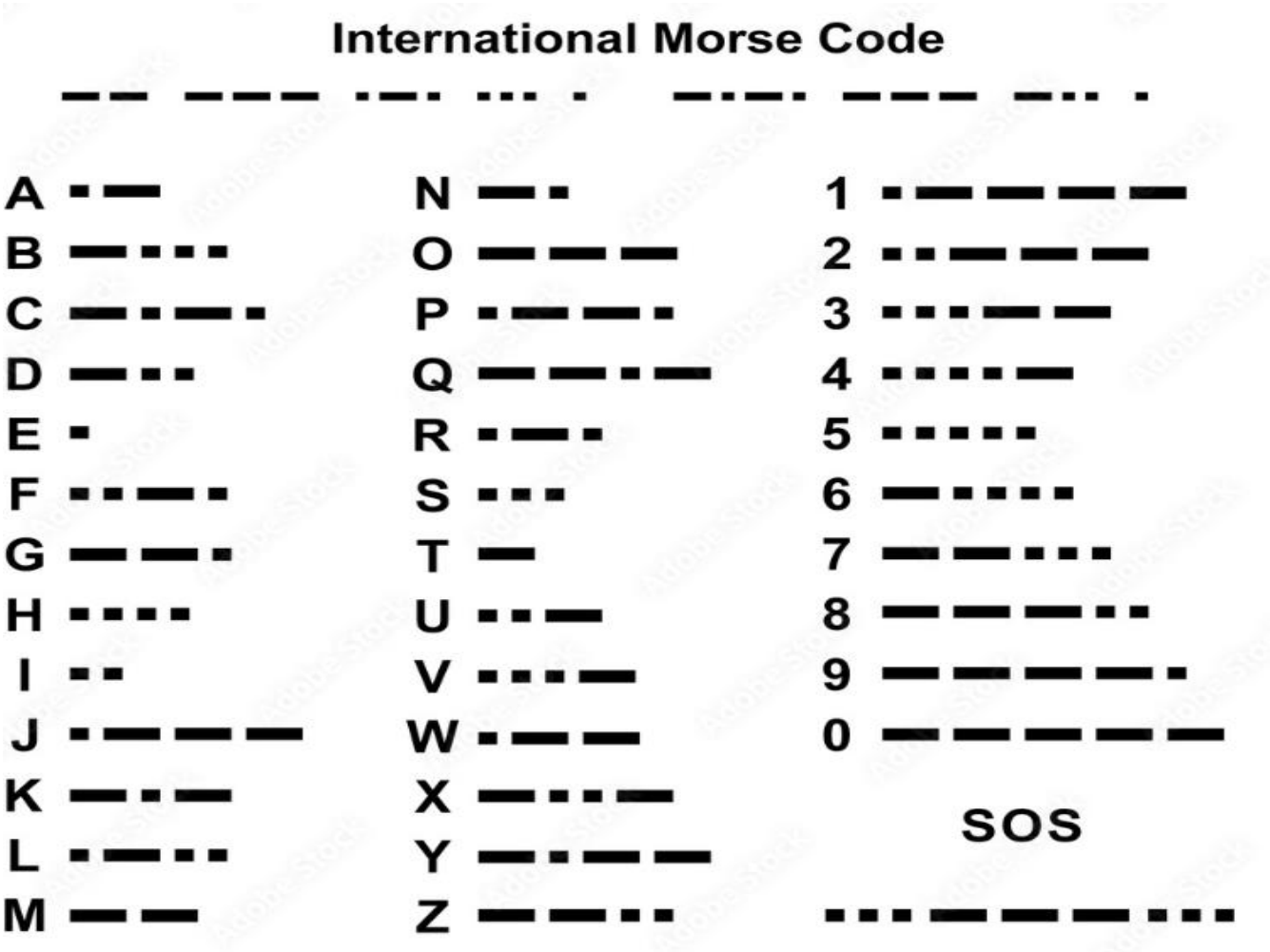

Figure 1 – International Morse code representation

The Morse code decoder is implemented on DE1 SOC Board with outputs being displayed on Monitor and input is taken from the FPGA board's own slide switches. The Figure 1 presents the communication between the FPGA Board and the external peripherals. The communication between FPGA board and monitor is established using VGA port present in FPGA board. In this project, various features of DE1 SOC board, like 7-segment display, slide switches, push buttons, internal 50MHz clock were used for convenience of implementation.

2.1. COMPONENT DESCRIPTION

i. DE1 SOC board

The DE1 boards feature an FPGA chip from Altera/Intel, providing programmable logic resources and other hardware components that can be configured as per user requirements

The DE1-SoC Development Kit presents a robust hardware design platform built around the Altera System-on-Chip (SoC) FPGA, which combines the latest dual-core Cortex-A9 embedded cores with industry-leading programmable logic for ultimate design flexibility.

Onboard memory components such as SDRAM, SRAM, and Flash memory are used for storing configuration data, software, and temporary data storage.
Various interfaces such as GPIO (General Purpose Input/Output), VGA (Video Graphics Array), audio in/out, USB, Ethernet are present in this SoC board which can be used for various application level projects.
The flexibility that a FPGA board provides in appreciated in the industry level projects, thus is widely used.

Physical components like switches, LEDs, and displays for input/output interaction and status indication during development and testing.
Built-in clock sources and timers to facilitate synchronous operations and timing-related experiments.

The Fig 2. Represents the De1 SoC development board with each and every featured marked on it.

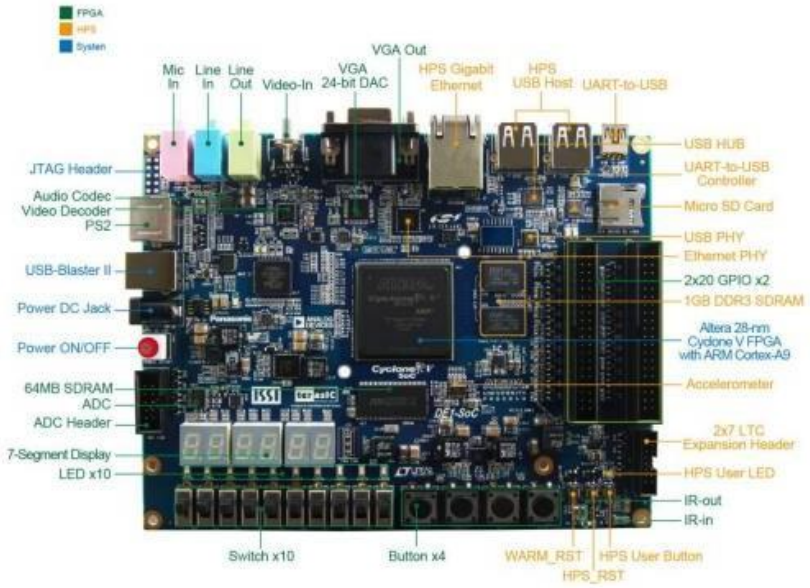

Figure 2 - DE1-SoC development board

2.2. PERIPHERALS USED

i. Switches

The DE1 SoC board features ten slide switches directly linked to the FPGA. These switches act as level-sensitive data inputs for the circuitry. Each switch offers a binary function: in the DOWN position, it generates a low logic level input to the FPGA, while in the UP position, it produces a high logic level. Notably, these switches lack debounce mechanisms, meaning they don't stabilize signals against potential bouncing effects. As a result, users can utilize these switches to provide direct input signals, establishing distinct logic levels within the FPGA-based design or experiment, offering a straightforward means of interfacing with the circuitry.

Figure 3 – DE1-Slide Switches and Push Buttons

ii. 7-segment Displays

The DE1-SoC board consists of six 7-segment displays, each configured to output numbers of varying sizes. The displays are interconnected to the Cyclone V SoC FPGA through their seven segments, having a common anode arrangement. By applying a low logic level from the FPGA, a specific segment illuminates, while a high logic level turns it off. Each of the seven segments within a display is indexed from 0 to 6, and their respective positions and connections to the FPGA are detailed in Figure 4. This setup allows precise control over individual segments within the 7-segment displays, facilitating the representation of numeric values and characters in different sizes based on the activation or deactivation of specific segments through FPGA-controlled logic levels.

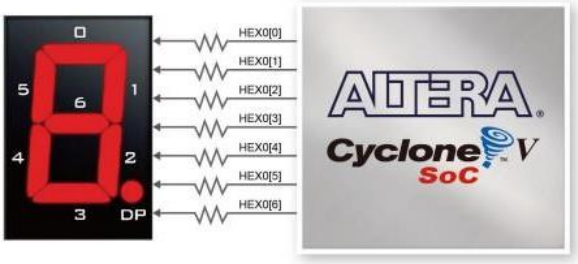

Figure 4 - 7-segment display

iii. VGA

The DE1-SoC board consists of a 15-pin D-SUB connector designated for VGA output. This connector is designed to transmit video signals. The synchronization signals essential for VGA display, comprising horizontal and vertical sync, are directly generated by the Cyclone V SoC FPGA. Additionally, the Analog Devices ADV7123, a triple 10-bit high-speed video DAC, is employed specifically for transforming digital signals from the FPGA into analog signals. Notably, the ADV7123 utilizes only the higher 8 bits for this purpose. The DAC facilitates the representation of three primary colors—red, green, and blue—in an analog format. This functionality allows the board to support resolutions up to SXGA (1280x1024) with signals transmitted at a rate of 100MHz. The interconnection between the FPGA and the VGA connector, outlining the transmission of signals for generating the VGA output, is depicted in detail in Figure 5. This configuration enables the FPGA to control the video output by translating digital signals into analog representations of the fundamental colors, contributing to the successful rendering of images on an external display through the VGA interface.

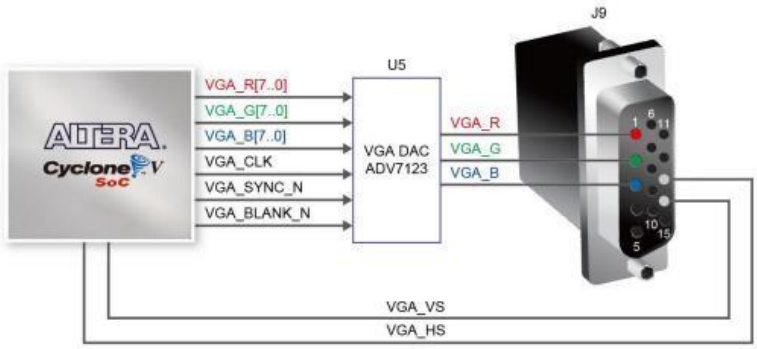

Figure 5 - Connection between FPGA and VGA

Figure 6 illustrates the basic timing requirements for each row (horizontal) displayed on a VGA monitor. n the context of monitor synchronization, the horizontal synchronization (hsync) input is triggered by an active-low pulse of specific duration, indicating the completion of one row of data and the initiation of the next. Following the hsync pulse, a designated duration known as the "back porch" (b) ensues. During this period, the data output to the monitor (RGB) must remain off, maintaining a signal at 0 volts. Subsequently, the "display interval" (c) commences, where the RGB data actively drives each pixel sequentially across the current row being displayed.

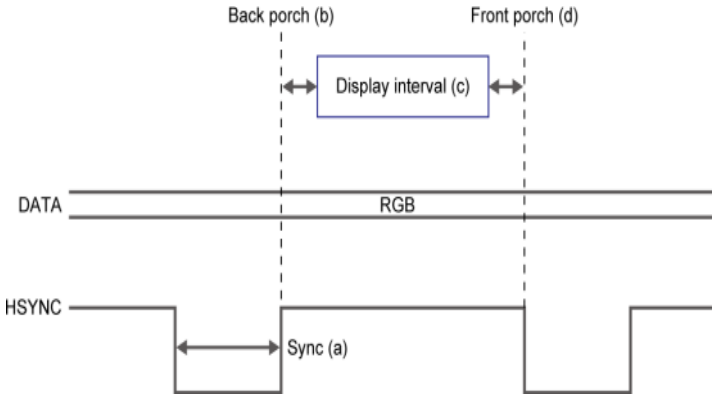

Figure 6 - VGA horizontal timing specification

2.3 WORKING

The input in the form of morse code is given to FPGA board using the slide switches by setting it as high for generating a 'DASH' and low for 'DOT'.

The slide switch is used as reset in this project, a negative edge reset is used, which means when the reset is low, the FPGA boards resets all required output variable values as 0.

The push button is mapped to Clock pulse. As the morse input given to the FPGA board varies in size for every alphabet, the timing issues arises if the inbuilt 50MHz clock is used. Thus, the use of push button as the clock for giving morse input is preffered.

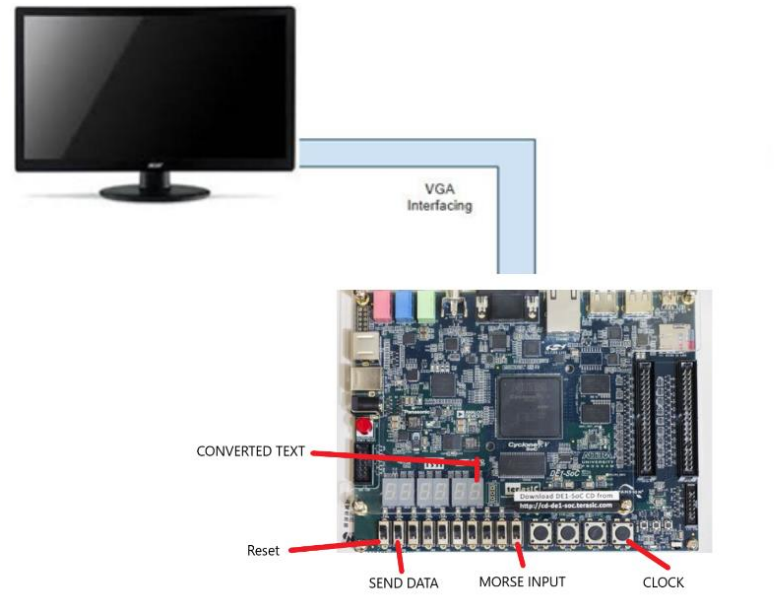

Figure 7 – Peripheral Interface

INPUT:

- When a sequence of morse input is given, the given input gets left shifted by the number of bits of input given. i.e., if 3 times a dot or dash is clicked then, the input get left shifted by 3 values.
- For every input of dot or a dash, the clock pulse given, so that the next input can be given.
- The "data_decoder" module converts the entered input sequence of morse code to text and the output text is stored in a register.
- When "Send data" switch is made high the output present in the register gets displayed on the 7-segment display.

OUTPUT:

- The output is the alphabets ranging from A to Z.
- The major output peripherals are VGA monitor display and seven segment monitor display.
- The VGA module is instantiated with the morse output so that it generates a RGB combination for that particular alphabet and shows the output on screen.
- After the display of the alphabet, the “send data” switch is made low and then, the next morse code input given, then again make the “send data” switch high for displaying the next alphabet.
- By repeating this procedure, we can obtain all the 26 alphabets on the monitor screen.

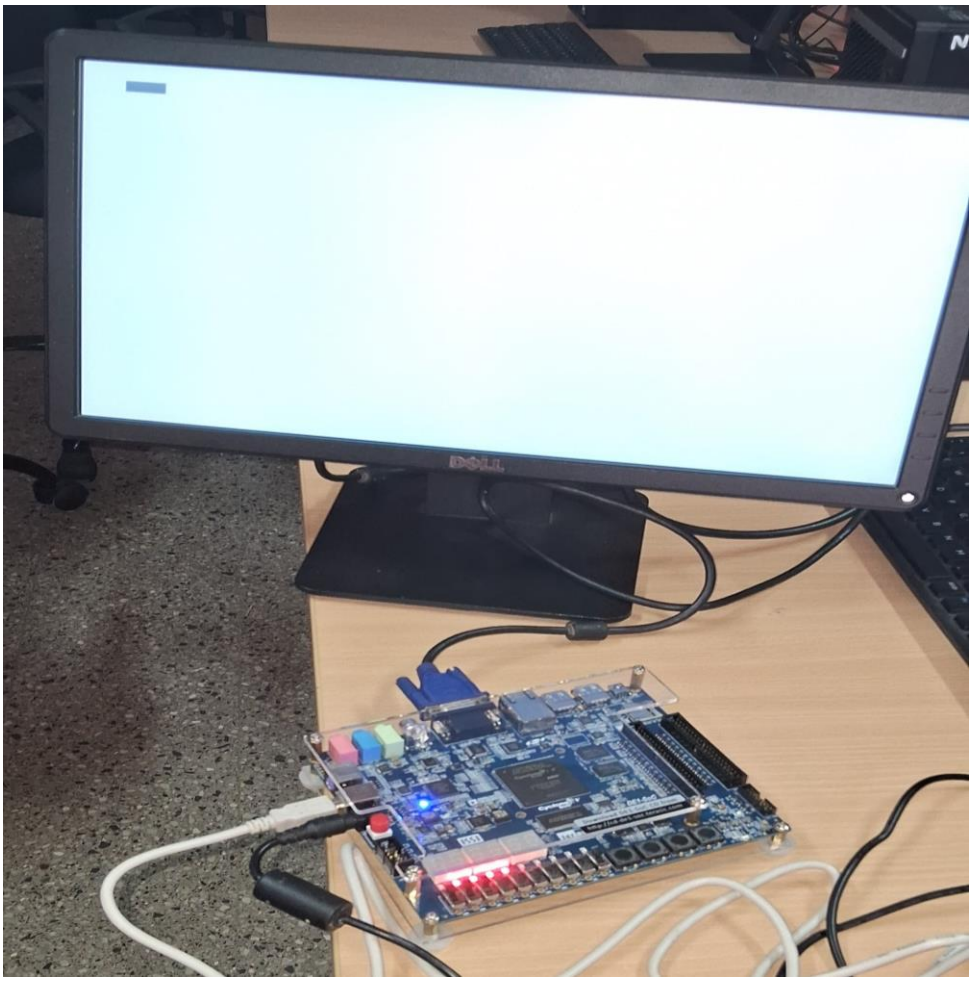

Figure 8 – Experimental Setup

2.4 SOFTWARE MODULES

i. **top_mod_morse**: (Top module)

This module implements the logic upon which the Morse Code Decoder works and combines rest of the modules to get inputs. It also synchronizes the function of VGA.

This module instantiates all the other modules present in the code and

ii. **data_in** module

- this module takes in the morse input by left shifting the input given by user.
- Also, counts the number bits the user is entering, which is helpful for determining which alphabet to map for the sequence of morse entered

iii. **data_decoder** module

- This module decodes the morse input to text.
- Stores the text into a output register.

iv. **display_project** module

- This module is the VGA interface module
- The pixel width, length, color and location on the monitor screen is determined.
- The pixel values for all the 26 Alphabets are coded
- chooses which output to show on the VGA screen as per user morse input.

data_in Module code:

```
module data_in(
input rst,
input clk,
input data_in,
input count_reset_sig,
output reg [4:0] out,
output reg [3:0] count_out,
output reg handshake_sig );
reg [36:0] count;

  always@(posedge clk)
  begin
    if(!rst)
```

```
            begin
               out = 5'b00000;
               count_out = 4'b1111;
            end
            begin
               if(count_reset_sig)
                  begin
                     count_out = 4'hf;
                     handshake_sig = 1'b1;
                  end
               else
                  begin
                     if(handshake_sig)
                     begin
                              out = 5'b00000;
                              handshake_sig = 1'b0;
                     end
                     out = {out[3:0], data_in};
                     count_out = count_out + 1;
                  end
            end
      end
   endmodule
```

data_decoder module code:

```
module data_decoder(
input rst, send_data,
input [3:0] count, data_out,
output reg [4:0] out_morse,
output reg count_reset_sig);
      parameter A_m = 0; parameter B_m = 1; parameter C_m = 2; parameter D_m = 3; parameter E_m =
4; parameter F_m = 5; parameter G_m = 6; parameter H_m = 7; parameter I_m = 8;parameter J_m =
9;parameter K_m = 10;parameter L_m = 11;parameter M_m = 12;parameter N_m = 13;parameter O_m =
14;parameter P_m = 15;parameter Q_m = 16;parameter R_m = 17;parameter S_m = 18;parameter T_m =
19;parameter U_m = 20;parameter V_m = 21;parameter W_m = 22;parameter X_m = 23;parameter Y_m =
24;parameter Z_m = 25; parameter SPACE_m = 26;

   always@(*)
   begin
```

```
if(!rst)
   begin
      out_morse = 5'b00000;
      count_reset_sig = 1'b0;
   end
else
   begin
   if(send_data)
      begin
      case(count)
         1: begin
            case(data_out[0])
               0 : out_morse = E_m;
               1 : out_morse = T_m;
            endcase
           end
         2: begin
            case(data_out[1:0])
               0 : out_morse = I_m;
               1 : out_morse = A_m;
               2 : out_morse = N_m;
               3 : out_morse = M_m;
            endcase
           end
         3:  begin
               case(data_out[2:0])
                  0 : out_morse = S_m;
                  1 : out_morse = U_m;
                  2 : out_morse = R_m;
                  3 : out_morse = W_m;
                  4 : out_morse = D_m;
                  5 : out_morse = K_m;
                  6 : out_morse = G_m;
                  7 : out_morse = O_m;
               endcase
            end
         4:  begin
               case(data_out[3:0])
                  0 : out_morse = H_m;
                  1 : out_morse = V_m;
```

```
                    2 : out_morse = F_m;
                    4 : out_morse = L_m;
                    6 : out_morse = P_m;
                    7 : out_morse = J_m;
                    8 : out_morse = B_m;
                    9 : out_morse = X_m;
                    10 : out_morse = C_m;
                    11 : out_morse = Y_m;
                    12 : out_morse = Z_m;
                    13 : out_morse = Q_m;
                    15 : out_morse = SPACE_m;
                  endcase
                end
              default: out_morse = 27;
          endcase
          count_reset_sig = 1'b1;
          end
        else
          begin
            count_reset_sig = 1'b0;
          end
        end
    end

endmodule
```

3. RESULT

Device Utilization:

Logic Utilization: 700 / 32,070 (2%)
Total Registers: 349
Total Pins: 68 / 457 (15%)

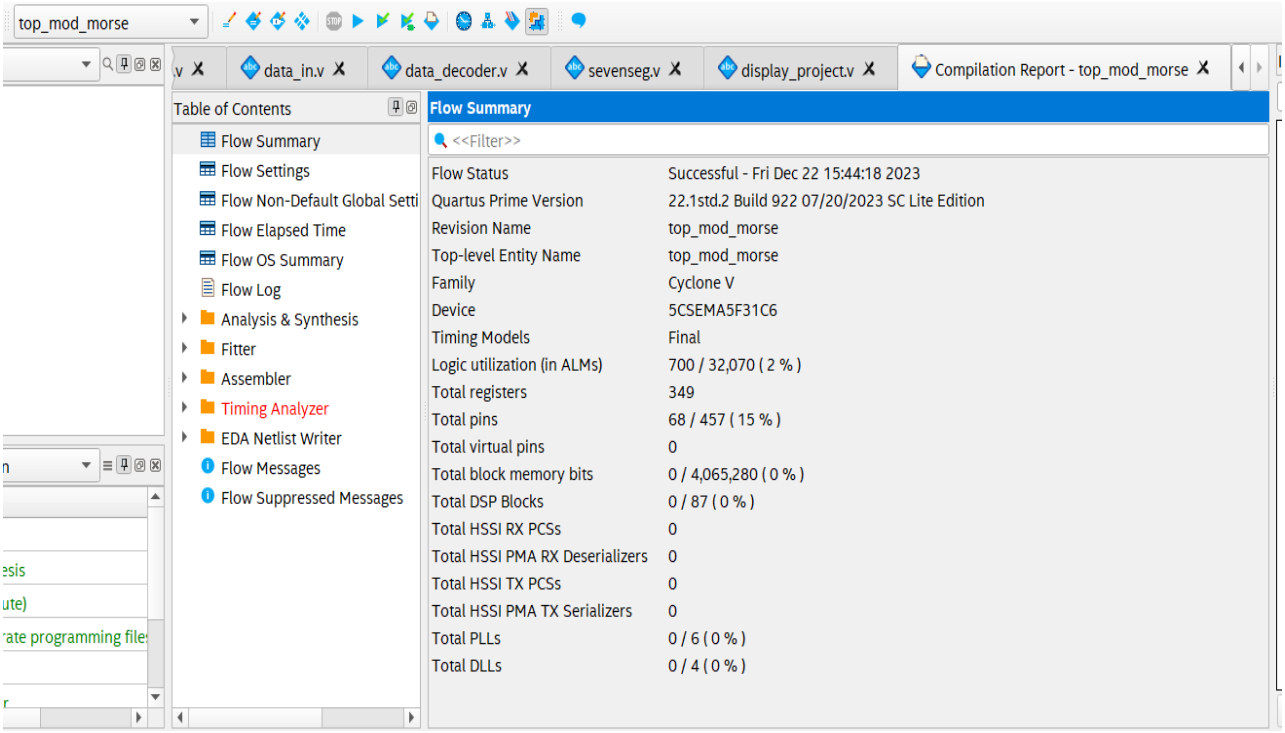

Figure 9 - Device Utilization statistics

The Output obtained from DE1 SoC FPGA board and VGA module is shown below for various alphabets.

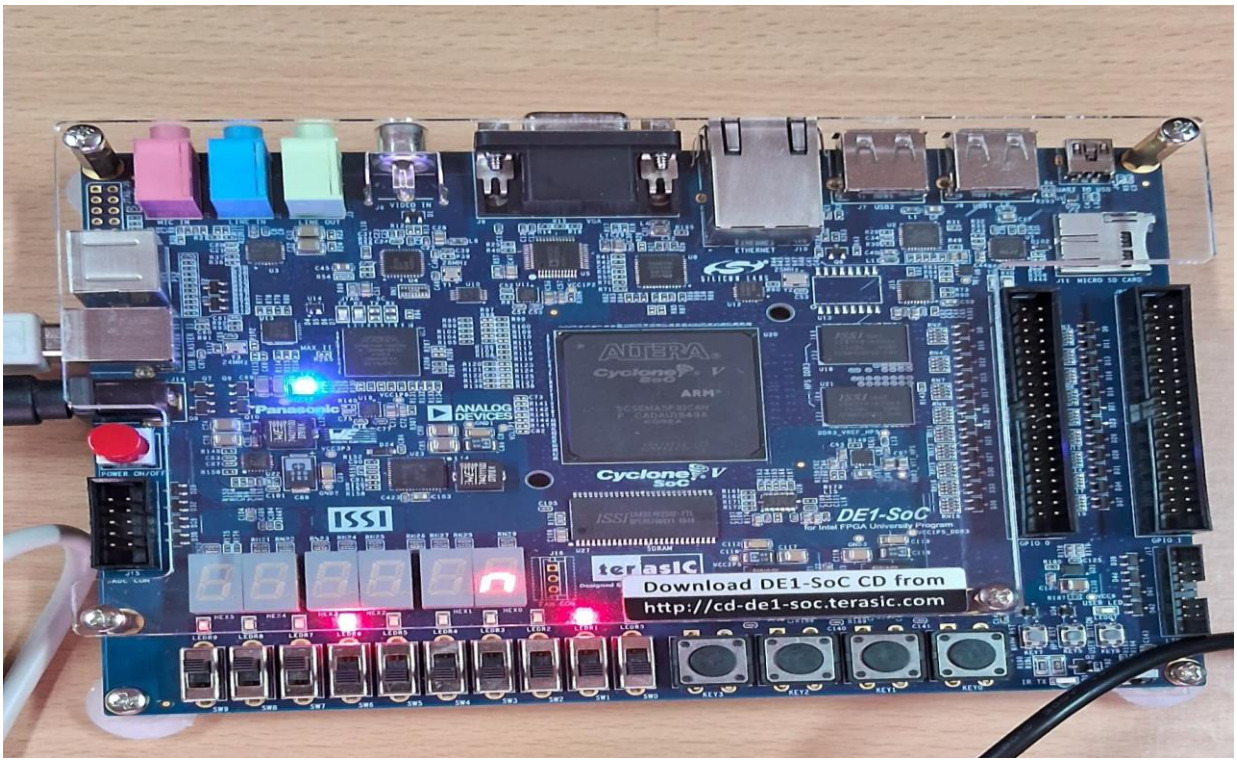

Figure 10 – 7-Segment showing Alphabet 'N'

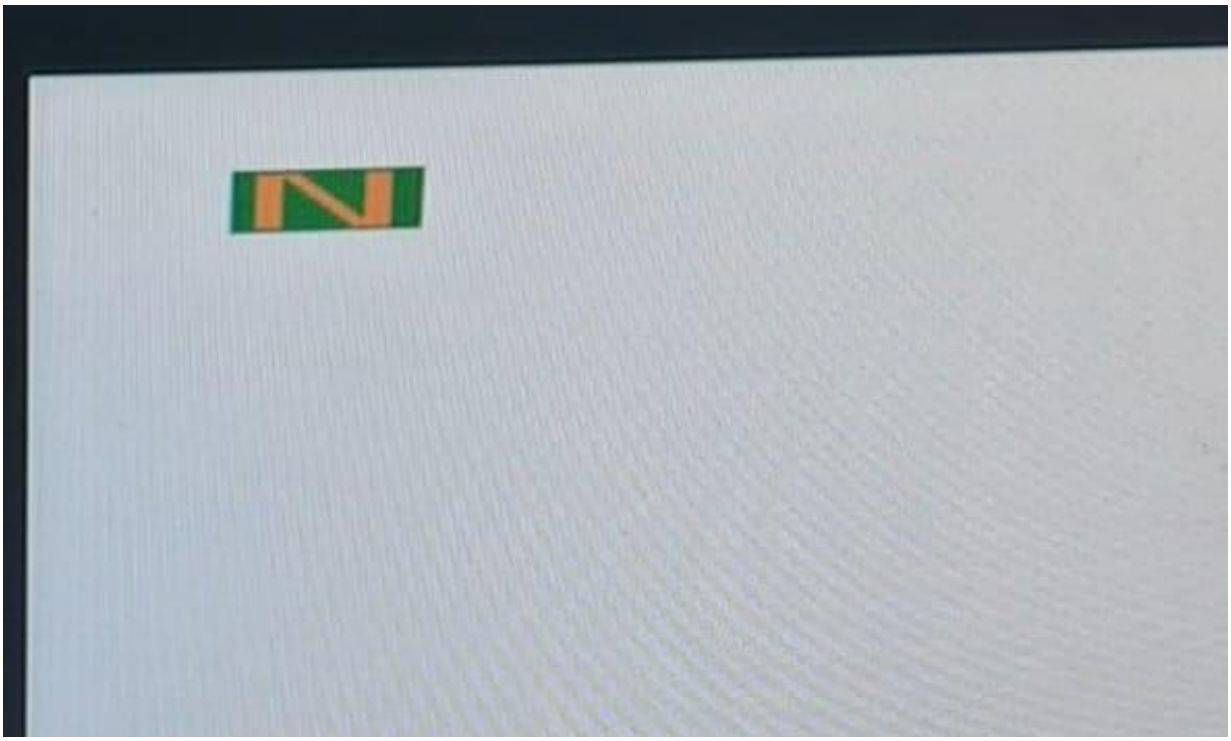

Figure 11 – VGA Interfaced Monitor showing Alphabet 'N'

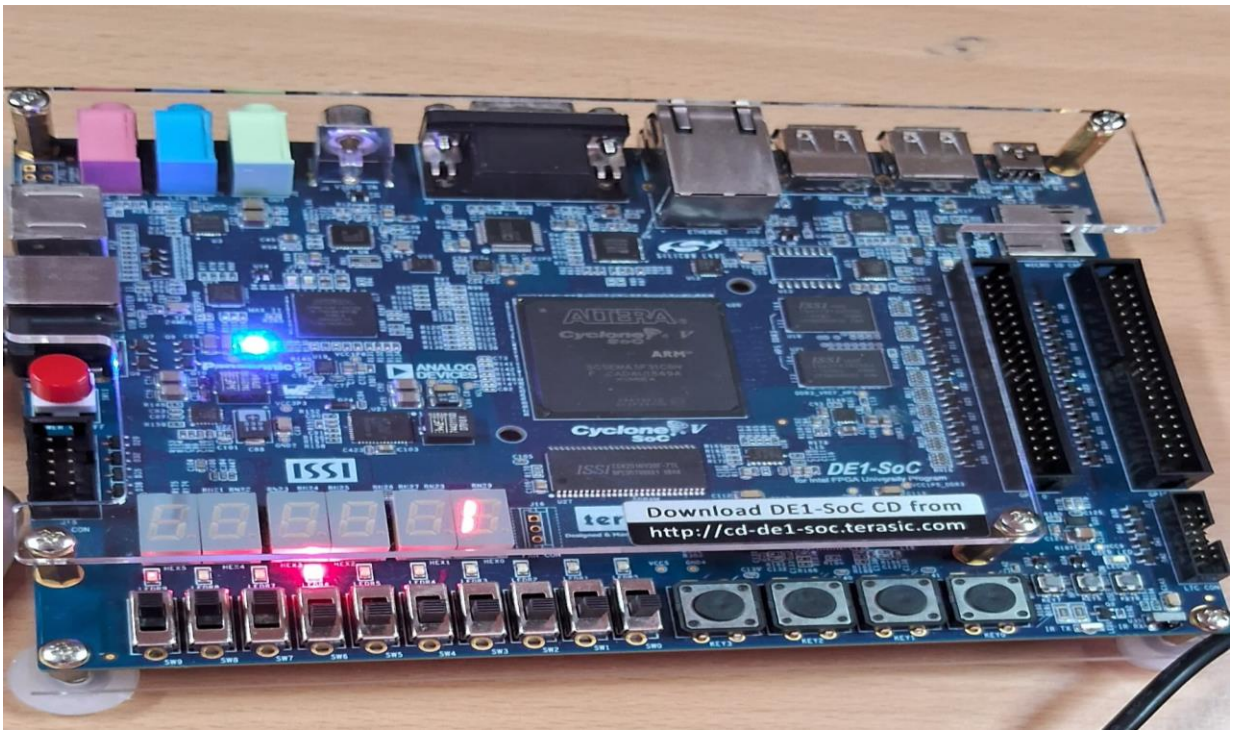

Figure 12 – 7-Segment showing Alphabet ‘I’

Figure 13 – VGA Interfaced Monitor showing Alphabet ‘N’

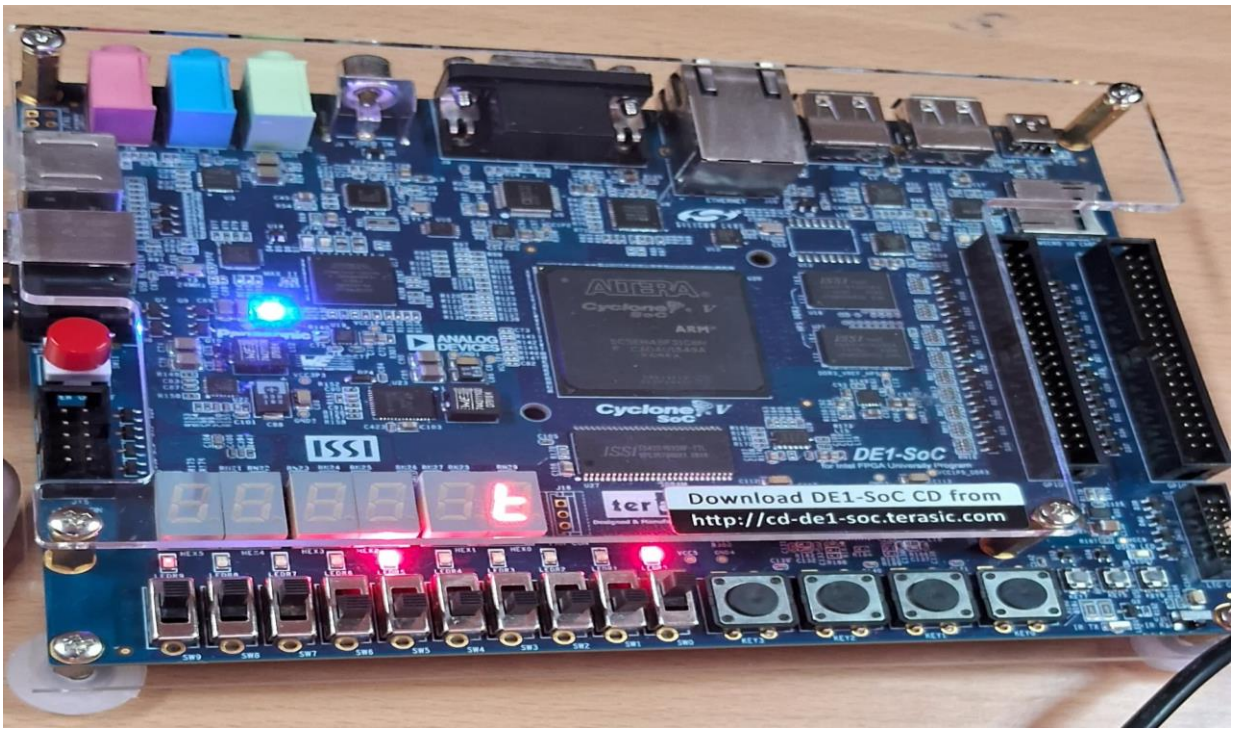

Figure 14 – 7-Segment showing Alphabet ‘T’

Figure 15 – VGA Interfaced Monitor showing Alphabet ‘T’

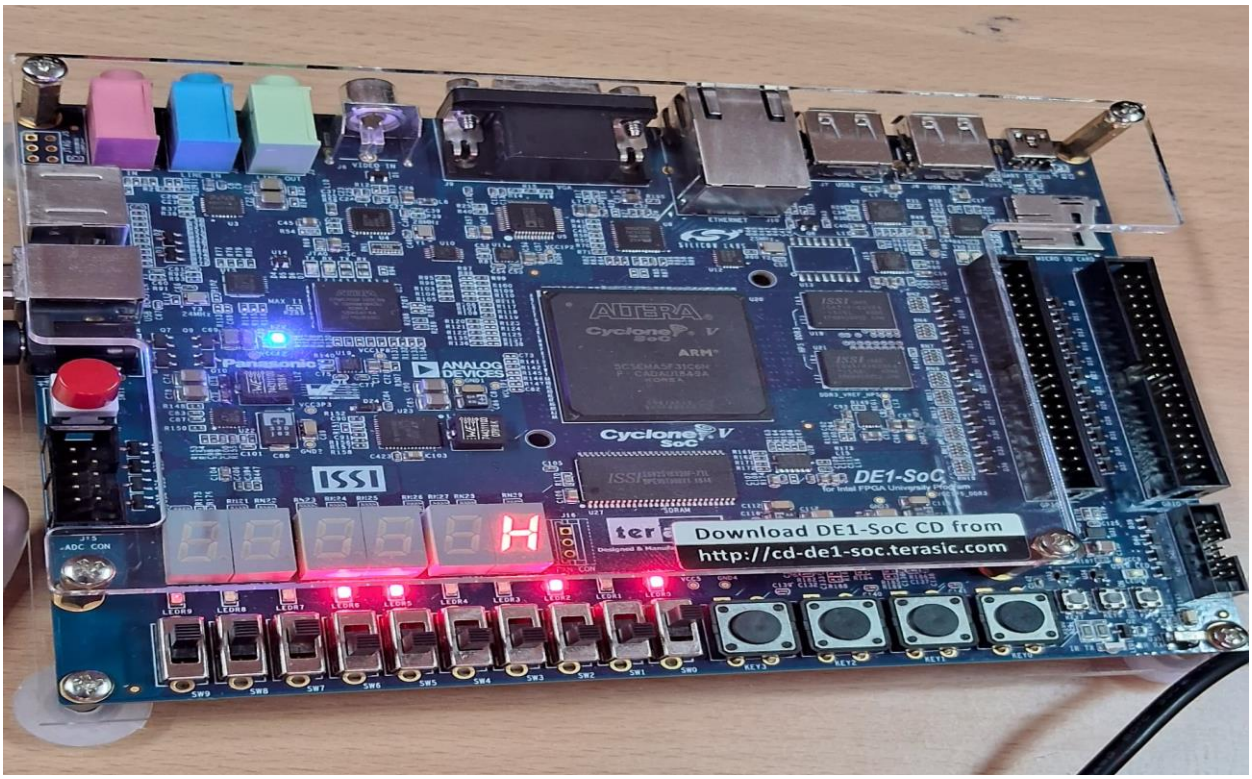

Figure 16 – 7-Segment showing Alphabet ‘K’

Figure 17 – VGA Interfaced Monitor showing Alphabet ‘K’

CONCLUSION

This design accomplishes FPGA based morse code decoder using DE1 SoC board. It utilizes on-board Slide switches and push buttons as input with LCD monitor and 7-segment as output display to represent Morse code in Text format. The design results in successful conversion of Morse code to corresponding Alphabet in text format. The alphabet is displayed in the monitor. Further this design can be extended as a decoder for various morse code combinations.