Simple Morse Code Sequence Translator and Detection

Instructor: Professor Gary Perks
Project Designer: Robin Simpson
Project Designer: Ronan Valadez

Fall 2023

Video Demo:

https://www.youtube.com/watch?v=Ppo3Xmh7wtc

Contents

1 Diagrams

1.1 HLBB

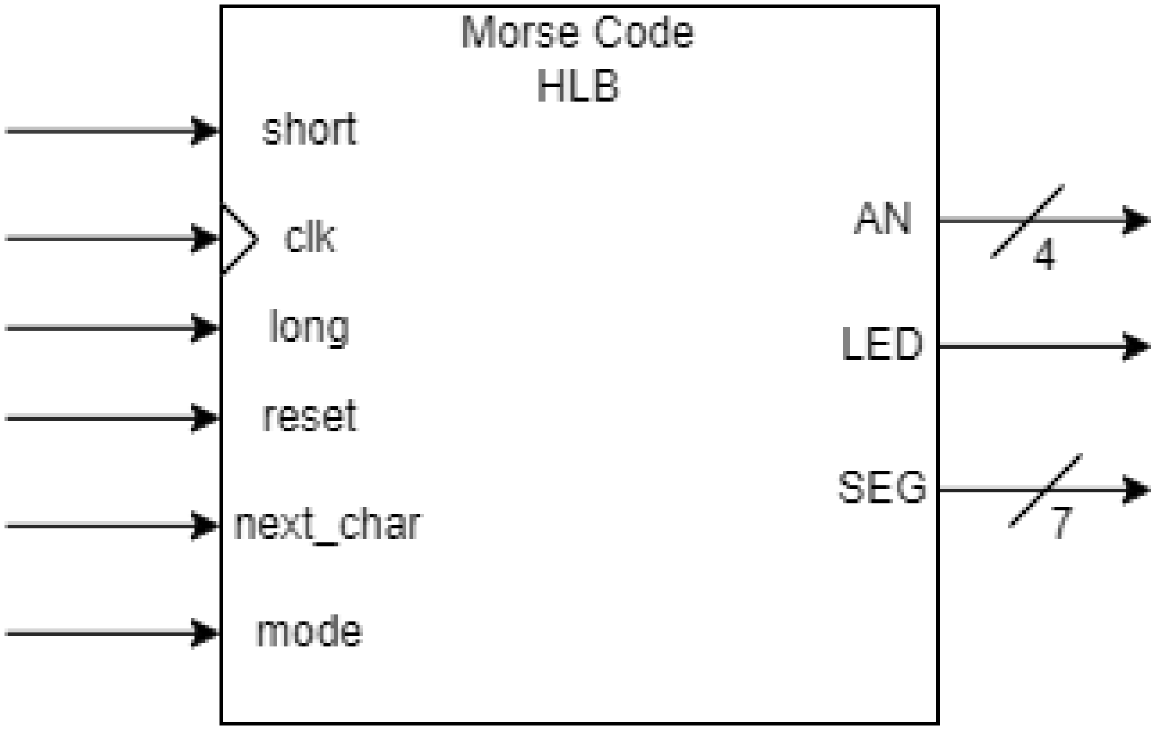


Figure 1: Draft of High Level Black Box of Morse Translator

1.2 Low Level Structural Diagram

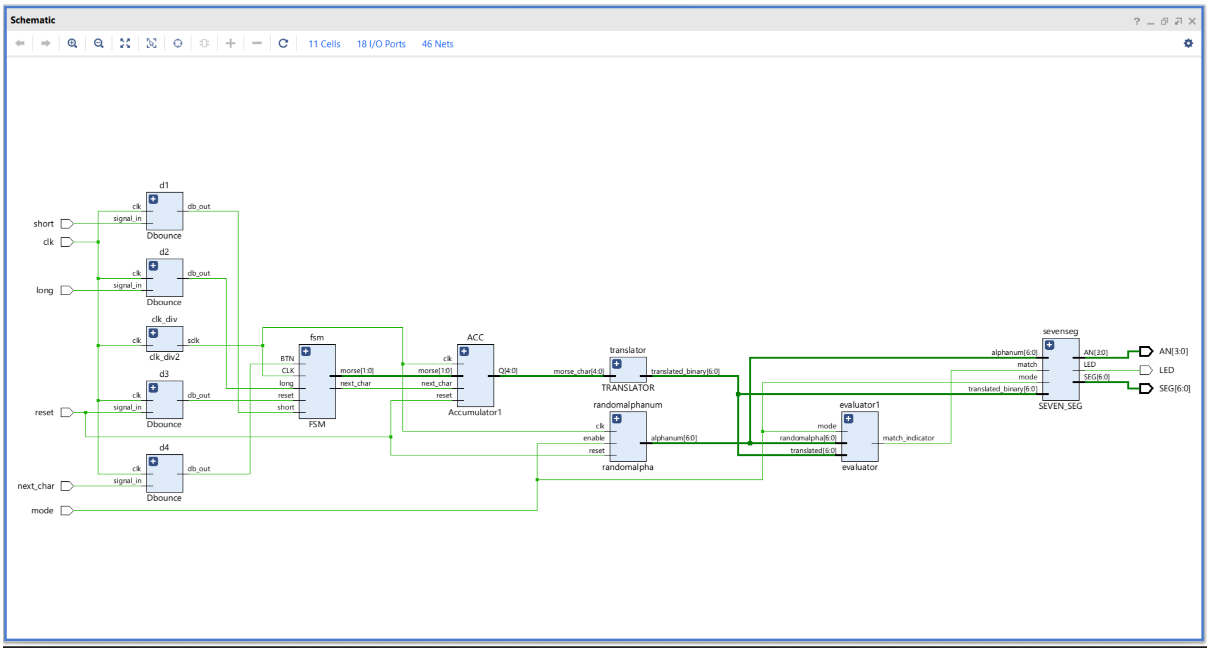


Figure 2: Structural Diagram **(Click to Enlarge)**

1.3 State Diagram

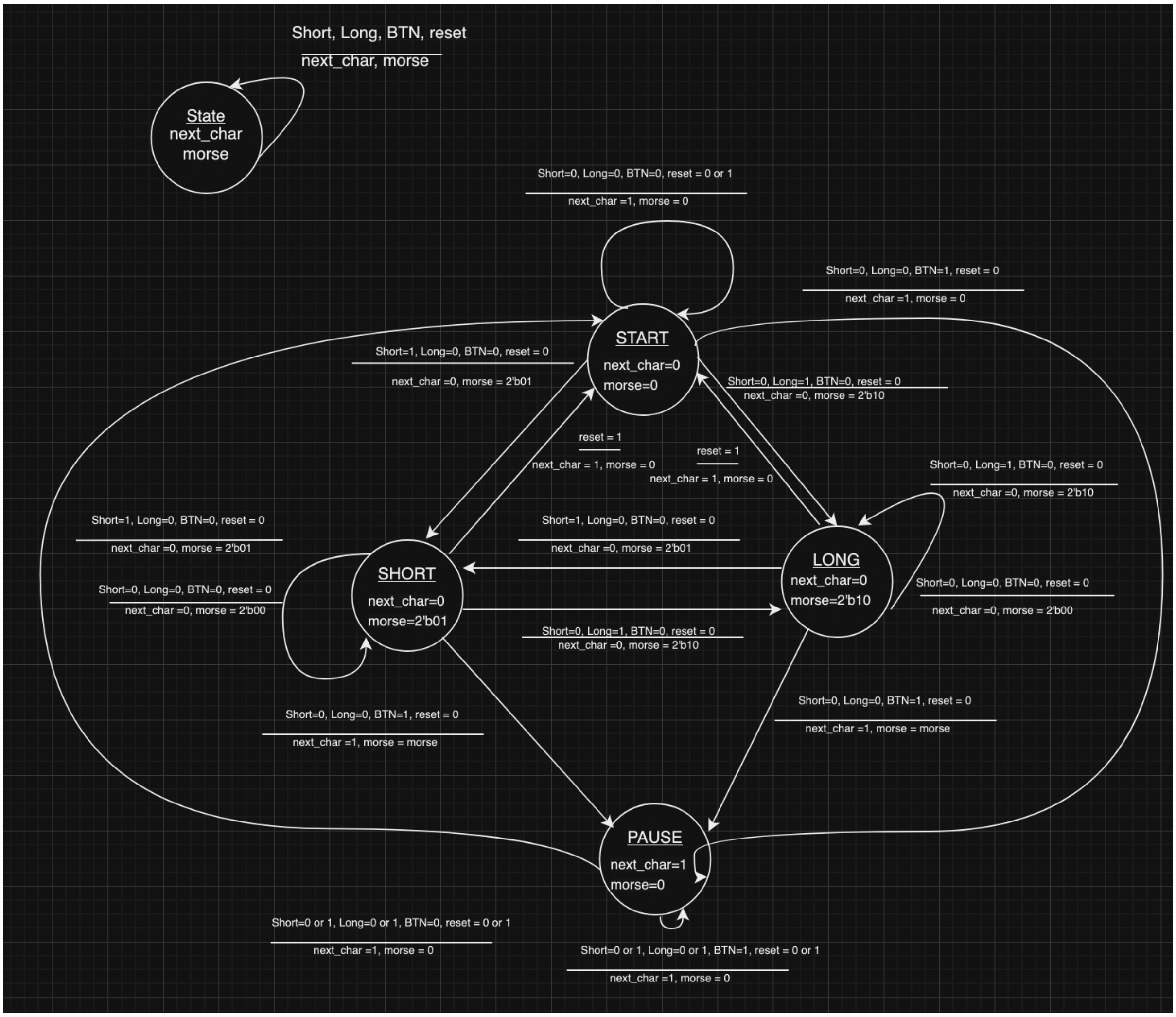


Figure 3: State Diagram **(Click to Enlarge)**

2 Simulation

2.1 Timing Diagram

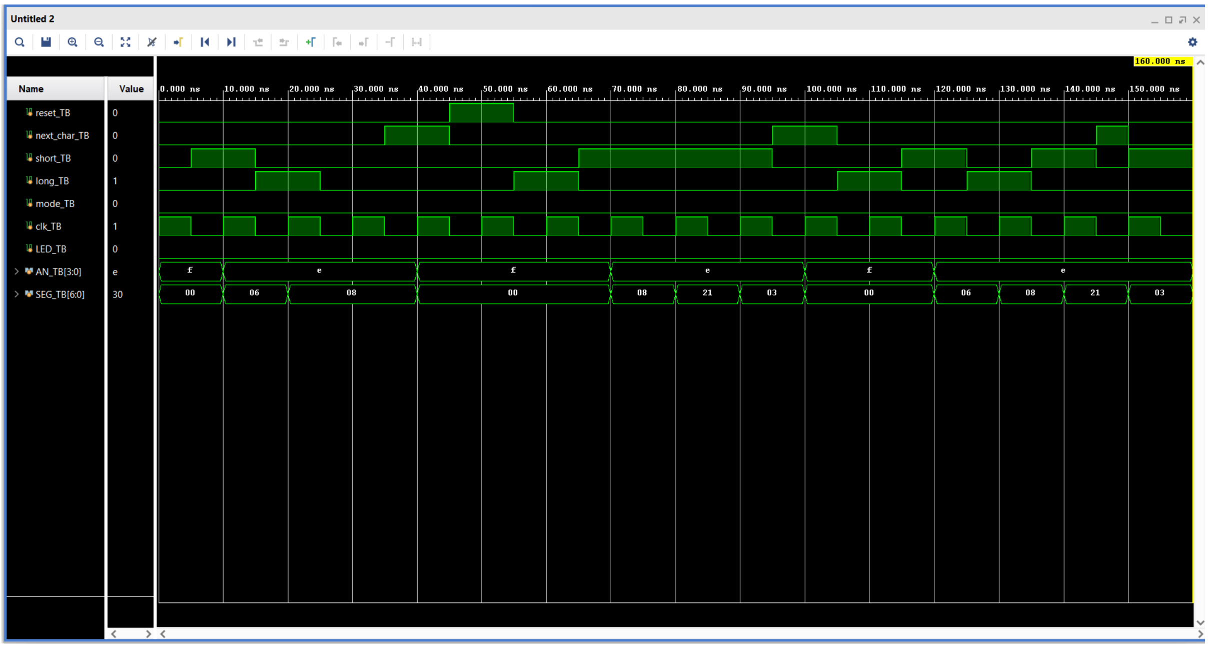


Figure 4: Simulation Timing Diagram **(Click to Enlarge)**

2.2 Code

Listing 1: TopLevelTB.sv Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////
// Company: Cal Poly
// Engineer: Robin Simpson + Ronan Valadez
//
// Create Date: 12/07/2023 07:26:58 PM
// Module Name: TOP_MODULE_TB
//////////////////////////////////////////////

module TOP_MODULE_TB();
logic reset_TB;
logic next_char_TB;
logic short_TB;
logic long_TB;
logic mode_TB;
logic clk_TB;
logic LED_TB;
logic  [3:0] AN_TB;
logic  [6:0] SEG_TB;
TOP_MODULE UUT (.reset(reset_TB), .next_char(next_char_TB),
.short(short_TB), .long(long_TB), .mode(mode_TB), .clk(clk_TB),
.LED(LED_TB), .AN(AN_TB), .SEG(SEG_TB));

initial begin
    reset_TB = 0;
    next_char_TB = 0;
    short_TB = 0;
    long_TB = 0;
    mode_TB = 0;
end

always begin
    clk_TB = 1'b1;
    #5;
    clk_TB = 1'b0;
    #5;
    end
always begin
    // test for A
    #5 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 1;
        long_TB = 0;
        mode_TB = 0;
   #10 reset_TB = 0;
```

```
        next_char_TB = 0;
        short_TB = 0;
        long_TB = 1;
        mode_TB = 0;
    #10 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 0;
        long_TB = 0;
        mode_TB = 0;
    #10 reset_TB = 0;
        next_char_TB = 1;
        short_TB = 0;
        long_TB = 0;
        mode_TB = 0;
    // test for b
    #10 reset_TB = 1;
        next_char_TB = 0;
        short_TB = 0;
        long_TB = 0;
        mode_TB = 0;
    #10 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 0;
        long_TB = 1;
        mode_TB = 0;
   #10 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 1;
        long_TB = 0;
        mode_TB = 0;
    #10 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 1;
        long_TB = 0;
        mode_TB = 0;
   #10 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 1;
        long_TB = 0;
        mode_TB = 0;
    #10 reset_TB = 0;
        next_char_TB = 1;
        short_TB = 0;
        long_TB = 0;
        mode_TB = 0;
    //test for C
    #10 reset_TB = 0;
        next_char_TB = 0;
```

```
        short_TB = 0;
        long_TB = 1;
        mode_TB = 0;
    #10 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 1;
        long_TB = 0;
        mode_TB = 0;
     #10 reset_TB = 0;
        next_char_TB = 0;
        short_TB = 0;
        long_TB = 1;
        mode_TB = 0;
    #10 reset_TB = 0;
        short_TB = 1;
        long_TB = 0;
        mode_TB = 0;
        next_char_TB = 0;
    #10 reset_TB = 0;
        short_TB = 0;
        long_TB = 0;
        mode_TB = 0;
        next_char_TB = 1;
    end
endmodule
```

3 Code and Modules

3.1 Accumulator

Listing 2: Accumulator.sv Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////
// Engineer: Ronan Valadez
// Create Date: 12/05/2023 12:57:18 PM
// Description: 8 bit acculumator.  Adds new value to the
// current value when LD is 1.
//////////////////////////////////////////////////

module Accumulator1(
    input clk,            // Clock signal
    input reset,          // Reset signal
    input next_char,      // Signal to load the next character
    input [1:0] morse,    // 2-bit input representing Morse code
    output logic [4:0] Q = 0  // 5-bit output register initialized to 0
    );

    always_ff @ (posedge clk)
    begin
        if (reset || next_char)
            Q <= 0;  // Reset or load next character: set Q to 0
        else if(!next_char)
            Q <= Q + morse;  // If not loading next character,
                             // add morse value to Q
        else begin
            Q <= Q;  // Maintain the current value of Q
        end
    end
endmodule
```

3.2 Evaluator

Listing 3: Evaluator.sv Code

```
`timescale 1ns / 1ps
////////////////////////////////////////////
// Company: Cal Poly
// Engineer: Robin Simpson
// Create Date: 12/05/2023 03:05:47 PM
// Design Name: EVL
// Module Name: evaluator
// Project Name: Morse
// Target Devices: Basys3
/////////////////////////////////////////////

module evaluator(
    input logic [6:0] randomalpha,  // 7-bit binary from randomalpha
    input logic [6:0] translated,   // User's 7-bit binary input
    input logic mode,   // Mode switch (enable/disable)
    output logic match_indicator  // Output match indicator
);
    // Compare logic
    always_comb begin
        if (mode) begin
            // If mode is enabled, compare the binary values
            match_indicator = (randomalpha == translated) ? 1 : 0;
        end
        else begin
            // If mode is disabled, do not perform comparison
            match_indicator = 1'bz; // Thus I make it red/blue
        end
    end

endmodule
```

3.3 Random Alpha Num Generator

Listing 4: randomalpha.sv Code

```
`timescale 1ns / 1ps
/////////////////////////////////////////////
// Company: CalPoly
// Engineer: Robin Simpson
// Create Date: 12/05/2023 01:24:29 PM
// Design Name: PRNGA
// Module Name: randomalpha
// Project Name: Morse
// Target Devices: Basys3
//////////////////////////////////////////

module randomalpha (
    input logic clk,          // Clock input
    input logic reset,        // Asynchronous reset
    input logic enable,       // Enable signal
    output logic [6:0] alphanum  // 7-bit output character
);

// PROFESSOR HUMMEL ASSISTED CODE
logic [31:0] r_random = SEED;  // LFSR register
logic s_feedback;
// PROFESSOR HUMMEL ASSISTED CODE


// Robin Code
logic generated_once;  // Flag to indicate generated

// PROFESSOR HUMMEL ASSISTED CODE
const logic [31:0] SEED = 32'h6B1CCA14;
// PROFESSOR HUMMEL ASSISTED CODE


// PROFESSOR HUMMEL ASSISTED CODE
assign s_feedback = ~(r_random[31] ^ r_random[21]
                    ^ r_random[1] ^ r_random[0]);
// PROFESSOR HUMMEL ASSISTED CODE


// Pseudo-random number generator - Robin Code
always_ff @(posedge clk) begin
    if (enable && reset) begin
        // Shift with feedback and set generated_once flag if enabled
        r_random <= {r_random[30:0], s_feedback};
    end
end
```

```
// Robin Code
logic [3:0] random_value;
assign random_value = r_random[3:0]; // Taking the 4 LSBs

// Robin Code
// Map the random 4-bit value to alphanumeric characters (0-9, A-F)
// For future additions/different input mode (if we get there)
always_comb begin
    // XXXXX means disabled character (future implementations)
    case(random_value)
        4'b0000: alphanum = 7'b1000000; // 0
        4'b0001: alphanum = 7'b1111001; // 1
        4'b0010: alphanum = 7'b0100100; // 2
        4'b0011: alphanum = 7'b0110000; // 3
        4'b0100: alphanum = 7'bxxxxxxx; // 4
        4'b0101: alphanum = 7'bxxxxxxx; // 5
        4'b0110: alphanum = 7'bxxxxxxx; // 6
        4'b0111: alphanum = 7'bxxxxxxx; // 7
        4'b1000: alphanum = 7'bxxxxxxx; // 8
        4'b1001: alphanum = 7'bxxxxxxx; // 9
        4'b1010: alphanum = 7'b0001000; // A
        4'b1011: alphanum = 7'b0000011; // b
        4'b1100: alphanum = 7'b1000110; // C
        4'b1101: alphanum = 7'b0100001; // d
        4'b1110: alphanum = 7'b0000110; // E
        4'b1111: alphanum = 7'bxxxxxxx; // F
        default: alphanum = 7'bXXXXXXX; // Undefined
    endcase
end

endmodule
```

3.4 Translator

Listing 5: translator.sv Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////
// Company: Cal Poly
// Engineer: Ronan Valadez
// Create Date: 12/06/2023 03:51:16 PM
//////////////////////////////////////////////
module TRANSLATOR(
input [4:0] morse_char,
output logic [6:0] translated_binary);
// Translates accumulated values into respective 7seg values
always_comb begin
    case(morse_char)
        5'b00011: begin  //A
            translated_binary = 7'b0001000;
        end
        5'b00101: begin  //b
            translated_binary = 7'b0000011;
        end
        5'b00110: begin  //C
            translated_binary = 7'b1000110;
        end
        5'b00100: begin  //d
            translated_binary = 7'b0100001;
        end
        5'b00001: begin  //E
            translated_binary = 7'b0000110;
        end
        5'b01010: begin  //0
            translated_binary = 7'b1000000;
        end
        5'b01001: begin  //1
            translated_binary = 7'b1111001;
        end
        5'b01000: begin  //2
            translated_binary = 7'b0100100;
        end
        5'b00111: begin  //3
            translated_binary = 7'b0110000;
        end
        default: begin //default to keep sevenseg off if no match
            translated_binary = 7'b0000000;
        end
   endcase
end
endmodule
```

3.5 7 Seg

Listing 6: sevenseg.sv Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////
// Company: Cal Poly
// Engineer: Ronan Valadez
// Create Date: 12/05/2023 04:25:52 PM
// Design Name: SEGS
// Module Name: SEVEN_SEG
// Project Name: Morse
//////////////////////////////////////////////////

module SEVEN_SEG(
    input [6:0] translated_binary, // 7-bit inpu
    input mode, // Mode selector
    input [6:0] alphanum, // 7-bit input representing the alphanum
    input  match, // Input signal to control the LED
    output logic [3:0] AN, // 4-bit output to anode
    output logic [6:0] SEG, // 7-bit output for the segments
    output logic LED // Output for the LED indicator
);

always_comb begin
    if(mode) begin
        SEG = alphanum; // In learn mode, display the alphanum value
        AN = 4'b1110; // Select the first digit to be active
        if(match) begin
            LED = 1; // If there is a match, turn on the LED
        end else begin
            LED = 0; // If there is no match, turn off the LED
        end
    end else begin
    // Standard mode
        if(translated_binary == 7'b0000000) begin
            AN = 4'b1111; // Deactivate all digits
            SEG = 0; // Clear the segments
            LED = 0; // Turn off the LED
        end else begin
            LED = 0; // Ensure the LED is off
            SEG = translated_binary; // Display the value
            AN = 4'b1110; // Select the first digit to be active
        end
    end
end

endmodule
```

3.6 FSM

Listing 7: FSM.sv Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////
// Company: Cal Poly
// Engineer: Ronan Valadez + Robin Simpson
// Create Date: 12/04/2023 04:28:45 PM
// Module Name: FSM
// Project Name: Morse
//////////////////////////////////////
module FSM(
    input CLK,       // Clock input from clk_div2, set at 10ms
    input short,     // Button input from debouncer for a short press
    input long,      // Button input from debouncer for a long press
    input reset,     // Reset input
    input BTN,       // Button input to trigger the next char
    output logic [1:0] morse,  // 2-bit output to represent Morse code
    output logic next_char  // Output to signal to process the next char
);

    // Enumerate the FSM states
    typedef enum {START, SHORT, LONG, PAUSE} state;

    // Declare and initialize current and next state variables
    state NS; // Next state
    state PS = START; // Present state, initialized to START

    always_ff @(posedge CLK) begin
        PS <= NS; // Assign the next state to the current state
    end

    always_comb begin
        // Initialize outputs
        morse = 0;
        next_char = 0;

        // State transition logic
        case(PS)
            START: begin // Default state of the FSM
                if(short) begin
                    morse = 2'b01; // Output Morse code for short press
                    NS = SHORT; // Transition to SHORT state
                end
                else if(long) begin
                    morse = 2'b10; // Output Morse code for long press
                    NS = LONG; // Transition to LONG state
                end
```

```
                else if(BTN) begin
                    next_char = 1; // Signal to process the next character
                    NS = PAUSE; // Transition to PAUSE state
                end
                else begin
                    NS = START; // Stay if no input is detected
                end
            end
            SHORT: begin // State for a short button press
                // Maintain unless conditions change
                morse = 2'b01;
                if(!short && !long && !BTN && !reset) begin
                    NS = START; // No inputs, go back to START
                end
                else if(reset) begin
                    NS = START; // If reset is pressed, go back to START
                end
                // Add additional conditions here if needed
            end
            LONG: begin // State for a long button press
                // Maintain unless conditions change
                morse = 2'b10;
                if(!short && !long && !BTN && !reset) begin
                    NS = START; // No inputs, go back to START
                end
                else if(reset) begin
                    NS = START; // If reset is pressed, go back to START
                end
                // Add additional conditions here if needed
            end
            PAUSE: begin // Pause between characters
                if(!BTN) begin
                    NS = START; // BTN released, go back to START
                end
                else begin
                    NS = PAUSE; // BTN still pressed, stay in PAUSE
                end
            end
            default: NS = START; // Undefined states
        endcase
    end
endmodule
```

3.7 Top Level Module

Listing 8: toplevel.sv Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////
// Company: Cal Poly
// Engineer: Robin Simpson + Ronan Valadez
// Create Date: 12/07/2023 07:02:44 PM
// Module Name: TOP_MODULE
////////////////////////////////////////////////

// Top level module definition for a project
module TOP_MODULE(
    // Description of inputs and outputs
    input reset,        // Button input for system reset
    input next_char,    // Button input to proceed to the next character
    input short,        // Button input for a short press (morse code dot)
    input long,         // Button input for a long press (morse code dash)
    input mode,         // Switch input to toggle modes
    input clk,          // System clock input
    output logic LED,     // LED output to indicate match (learn mode)
    output logic [3:0] AN,  // Outputs to control anode
    output logic [6:0] SEG   // Outputs to control segment
);

// Internal signals declaration
logic clkdiv_out; // Clock signal post division
logic [4:0] acc_out; // Output from the Accumulator
logic [1:0] morse_out; // Morse code output from the FSM module
logic next_char_signal; // Process the next character
logic [6:0] translated_out_signal; // Output from TRANSLATOR module
logic [6:0] alphanum_out_signal; // Output from randomalpha module
logic evaluator_out_signal; // Output from evaluator module

// Debounced signals for button inputs
logic short_dbounce, long_dbounce, reset_dbounce, next_char_dbounce;

// Instantiations of modules and mapping of inputs and outputs

// Clock divider to generate a slower clock signal
clk_div2 clk_div (.clk(clk), .sclk(clkdiv_out));

// Debouncer for short button press
Dbounce d1 (.clk(clk), .signal_in(short), .db_out(short_dbounce));

// Debouncer for long button press
Dbounce d2 (.clk(clk), .signal_in(long), .db_out(long_dbounce));
```

```
// Debouncer for reset button
Dbounce d3 (.clk(clk), .signal_in(reset), .db_out(reset_dbounce));

// Debouncer for next_char button
Dbounce d4 (.clk(clk), .signal_in(next_char), .db_out(next_char_dbounce));

// FSM for morse code processing
FSM fsm (.CLK(clkdiv_out), .short(short_dbounce), .long(long_dbounce),
.BTN(next_char_dbounce), .reset(reset_dbounce), .morse(morse_out),
.next_char(next_char_signal));

// Accumulator module to sum morse code inputs
Accumulator1 ACC (.clk(clkdiv_out), .reset(reset),
.next_char(next_char_signal),.morse(morse_out), .Q(acc_out));

// Translator module to convert morse code
TRANSLATOR translator (.morse_char(acc_out),
.translated_binary(translated_out_signal));

// Module to generate a random alphanumeric value
randomalpha randomalphanum (.clk(clkdiv_out), .reset(reset),.enable(mode),
.alphanum(alphanum_out_signal));

// Evaluator module to compare random alphanumeric value with translated
evaluator evaluator1 (.randomalpha(alphanum_out_signal),
.translated(translated_out_signal),.mode(mode),
.match_indicator(evaluator_out_signal));

// 7-segment display driver module
SEVEN_SEG sevenseg (.translated_binary(translated_out_signal),.mode(mode),
.alphanum(alphanum_out_signal), .match(evaluator_out_signal),
.AN(AN), .SEG(SEG), .LED(LED));

endmodule
```

3.8 Dbouncer

The module Dbounce is a debouncer for digital input signals. It is essential for stabilizing inputs from buttons. This module is used in its original form written by Prof. Ratner, with no modifications made.

3.9 Clock Divider

Standard Clock Divider module provided from Dr. Mealy; modified so signal is 10ms long.

Project Name: Simple Morse Code Translator

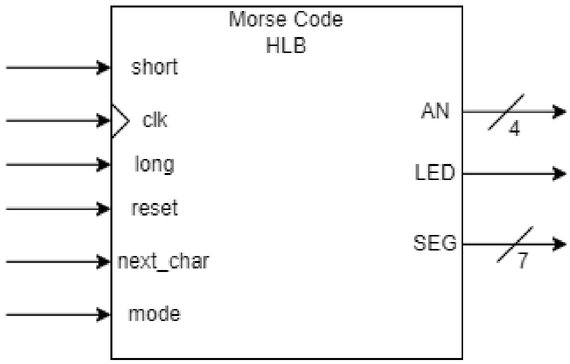
Project Designers: Robin Simpson, Ronan Valadez

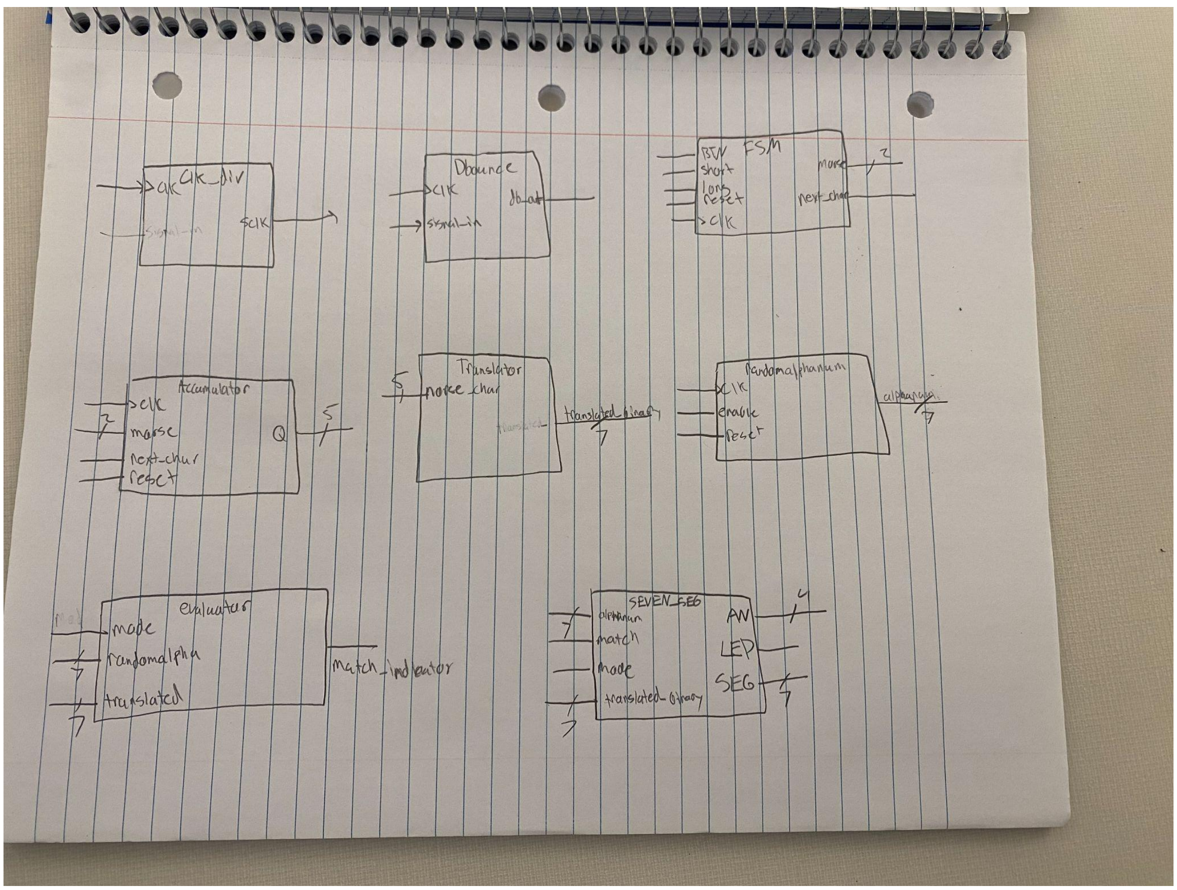
Project Description:

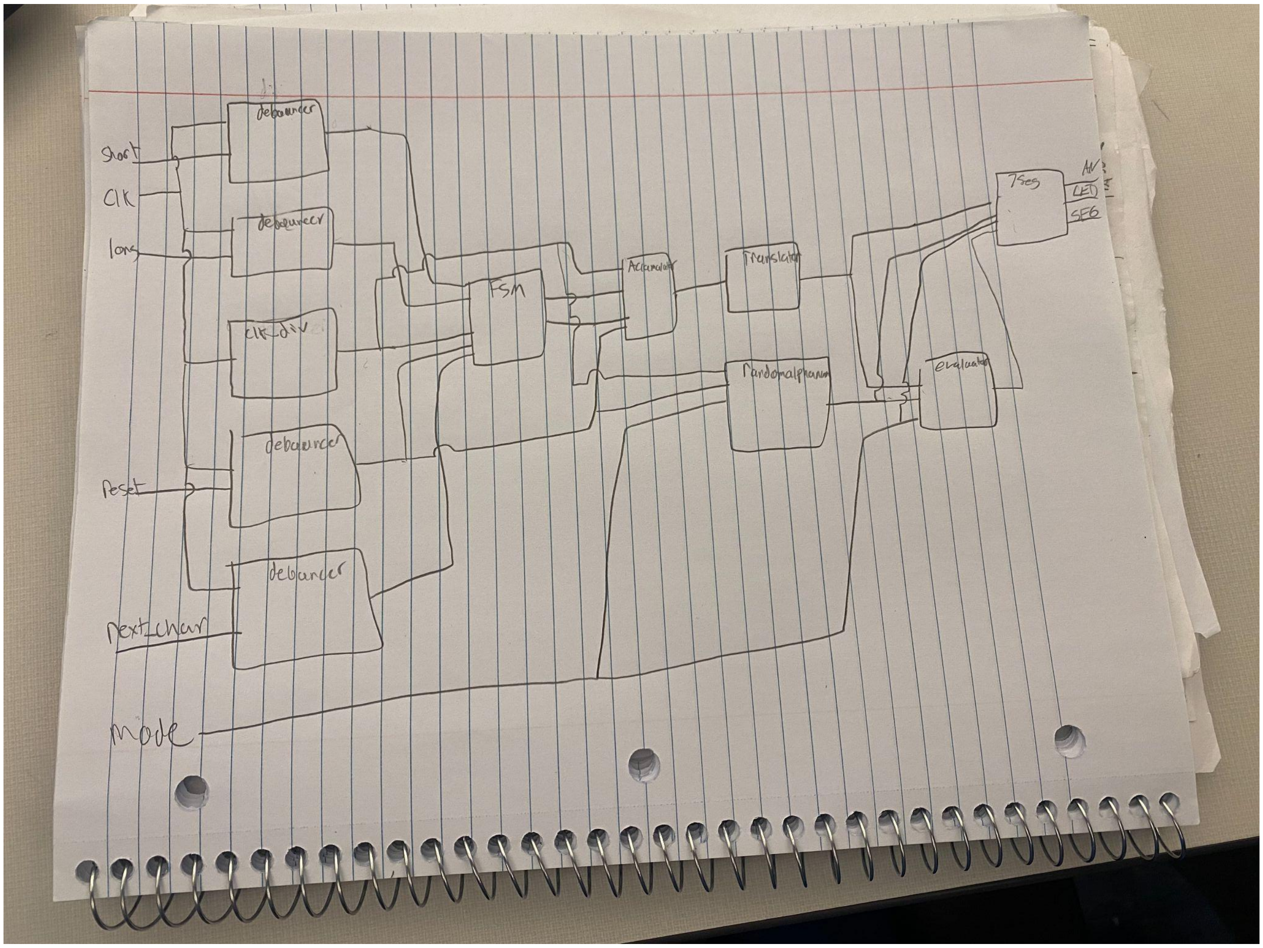
A switch will be used to distinguish between the two different modes. In the first mode, the user will be able to translate Morse code into a single alphanumeric character that will be displayed on the rightmost position of the Seven Segment Display. These characters will range from A-E, and 0-3. We use this range of characters because each character is given a unique sum based on their inputted morse. These were the only characters we could display on the Seven Segment Display, and have a unique sum using our input method. To enter a short, the user needs to press the short button input once. This button is located at T18. To input a long, press the long button which is located at W19. On each input, an accumulator will be adding the corresponding Morse value to a total. A short input adds one in binary, and a long input adds two in binary. These Morse binary numbers will be sent to an accumulator that will add up the sum. This sum will be initialized to zero, and be five bits. The 5 bit binary number will then be sent to a translator that will convert it to a 7 bit binary number for the Seven Segment Display module. The Seven Segment Display module will then display your corresponding alpha numeric character on the rightmost position of the seven Segment Display. If the user wants to reset their Morse input, they would press the reset button, located at V18, or the next_char button, located at T17.

In the second mode, the learner/testing mode, a random alphanumeric character will be displayed on the rightmost position of the Seven Segment Display. These characters will range from A-E, and 0-3. The user will then attempt to enter the correct Morse code for the given character. If their answer is correct, an LED located at LD0 will light up. If their answer is incorrect, the LED will stay unlit, and the user must cycle to the next random character by pressing either the reset or next_char buttons. To determine if the Morse input matches the random alphanumeric code, the Morse input will be sent to an evaluator that will determine if the LED should be on or off.

Draft of high-level black box diagram and structural low-level diagram:







Explanation of how FMS, Accumulator and new module criteria is met:

The system comprises the following main components:

- **Finite State Machine (FSM):** Serves as the central input processing unit, handling user interactions and Morse code signal interpretation.

- **Debouncer:** Plays a crucial role in stabilizing input signals, eliminating the effects of signal bouncing typically associated with physical button presses. Its design and implementation are key to ensuring the reliability and accuracy of user inputs in the system.

- **Accumulator :** Collects and compiles Morse code signals into a unified format for further processing.

- **Translator Module:** Converts the compiled Morse code from the accumulator into alphanumeric characters.
- **Seven Segment Display:** Displays the alphanumeric characters or prompts in both operational modes.
- **Random Alphanumeric Generator:** Generates random characters for the Learning Mode.
- **Evaluator:** Assesses the correctness of user inputs in Learning Mode and triggers the LED indicator.

Detailed Breakdown:

Finite State Machine (FSM) - FSM.sv:

Module Structure and Logic:

- State Definitions: Defines states such as START, SHORT, LONG, and PAUSE for processing Morse code inputs.
- Input Handling: Includes logic for interpreting short and long button presses, translating these into Morse code signals.
- State Transitions: Utilizes logic gates and conditional statements for transitioning between states based on button inputs.
- Output Generation: Produces a 2-bit output representing the Morse code signal.

Key Functionalities:

- Morse Code Interpretation: Converts user button presses into standard Morse code signals (dot and dash).
- State Management: Ensures accurate tracking and transitioning of states in response to user inputs.

Accumulator - Accumulator1.sv

Signal Processing and Storage:

- Morse Code Aggregation: Accumulates sequential Morse code signals into a complete character.
- Binary Representation: Maintains a binary format for the accumulated Morse code.
- Reset and Next Character Logic: Handles signals for resetting the accumulated data and moving to the next character.

Evaluator - evaluator.sv

Comparison and Validation Logic

- Input Assessment: Compares the user's Morse code input against a pre-defined standard from Random Alphanumeric Generator.
- Accuracy Determination: Provides a mechanism to assess the correctness of user inputs, especially in learning scenarios. This comes in the form of asserting whether user input is equal to the generated character.

Random Alphanumeric Generator - randomalpha.sv

Random Character Generation Mechanism

- Pseudo-random Generation: Implements an algorithm for generating random characters. Provided by Professor Hummel. Modified for 4 bit randomization.
- Morse Code Mapping: Maps generated characters to their corresponding Morse code.

Seven Segment Display - SEVEN SEG.sv

Display Control Logic

- Character Visualization: Manages the display of Morse code translations or generated characters on the seven-segment display.

- Mode-Dependent Display: Adjusts the display output based on the operational mode (Standard or Learning).

Top Module - TOP MODULE.sv

System Integration and Management

- Central Control: Orchestrates the interaction and data flow between all sub-modules.

- Input and Output Management: Defines and manages the system's inputs and outputs, ensuring cohesive operation.

Translator - TRANSLATOR.sv

Translation Mechanics

- Morse Code to Text Conversion: Translates binary Morse code into alphanumeric characters.

- Look-up Table or Mapping Logic: Employs a system to associate Morse code patterns with specific characters.