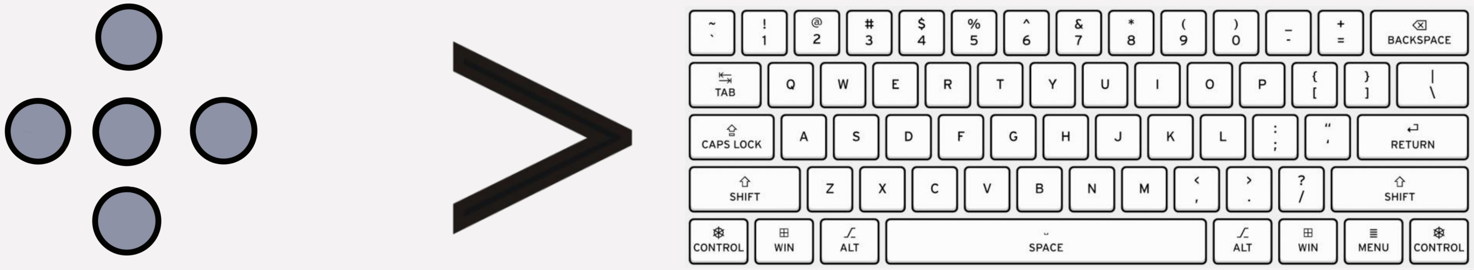
Team D●t Dash

Morse Code Translator

Team 2 Members: Ezan Khan, Keimaree Smith, Jonathan Thea, Tadiwanashe Zinyongo

Goals and Motivation

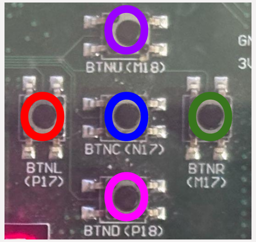
- We are developing a Morse code translator that allows users to input dots and dashes via push buttons on an FPGA.
  - Our project translates the inputs into English letters or numbers from 0 to 9.
- Purpose/Use cases:
  - Serves as a backup communication method during emergencies, allowing for the decoding of messages when other signaling systems are unavailable.
  - Additionally, can assist individuals with limited motor functions, providing an easier means of communication compares to a traditional keyboard

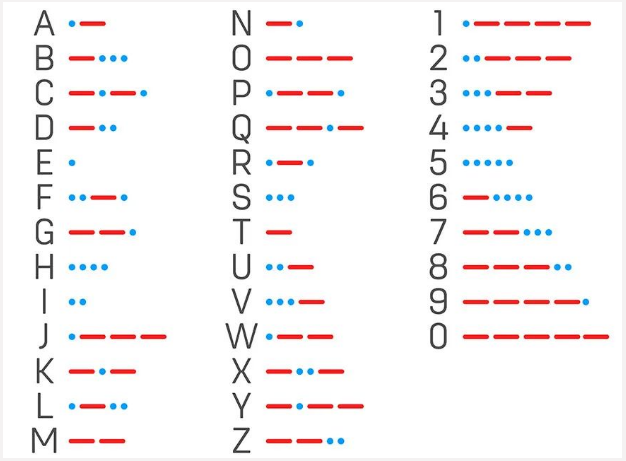


Functionality

You will have the following 5 input buttons on the FPGA where the:

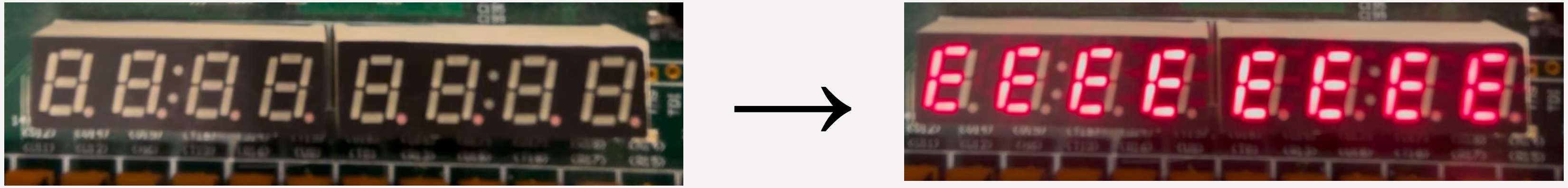
1. left button inputs a **dot**
2. center button inputs a **dash**
3. top button **clears** all characters
4. bottom button **deletes** one character
5. right button **enters** our sequence of dots & dashes





A	•—	N	—•	1	•————
B	—•••	O	———	2	••———
C	—•—•	P	•——•	3	•••——
D	—••	Q	——•—	4	••••—
E	•	R	•—•	5	•••••
F	••—•	S	•••	6	—••••
G	——•	T	—	7	——•••
H	••••	U	••—	8	———••
I	••	V	•••—	9	————•
J	•———	W	•——	0	—————
K	—•—	X	—••—		
L	•—••	Y	—•——		
M	——	Z	——••		

The output will be displayed on our FPGAs 7 segment display, as shown below:



Specifications

Requirements

On the FPGA/**hardware** side, we will need:

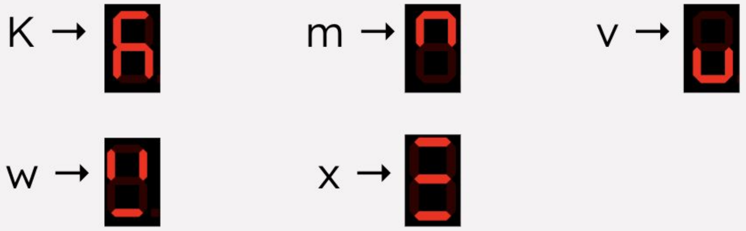
1. 5 buttons (dot, dash, clear, delete, enter)
2. Eight character 7-segment display

On the Verilog/**software** side, we will need:

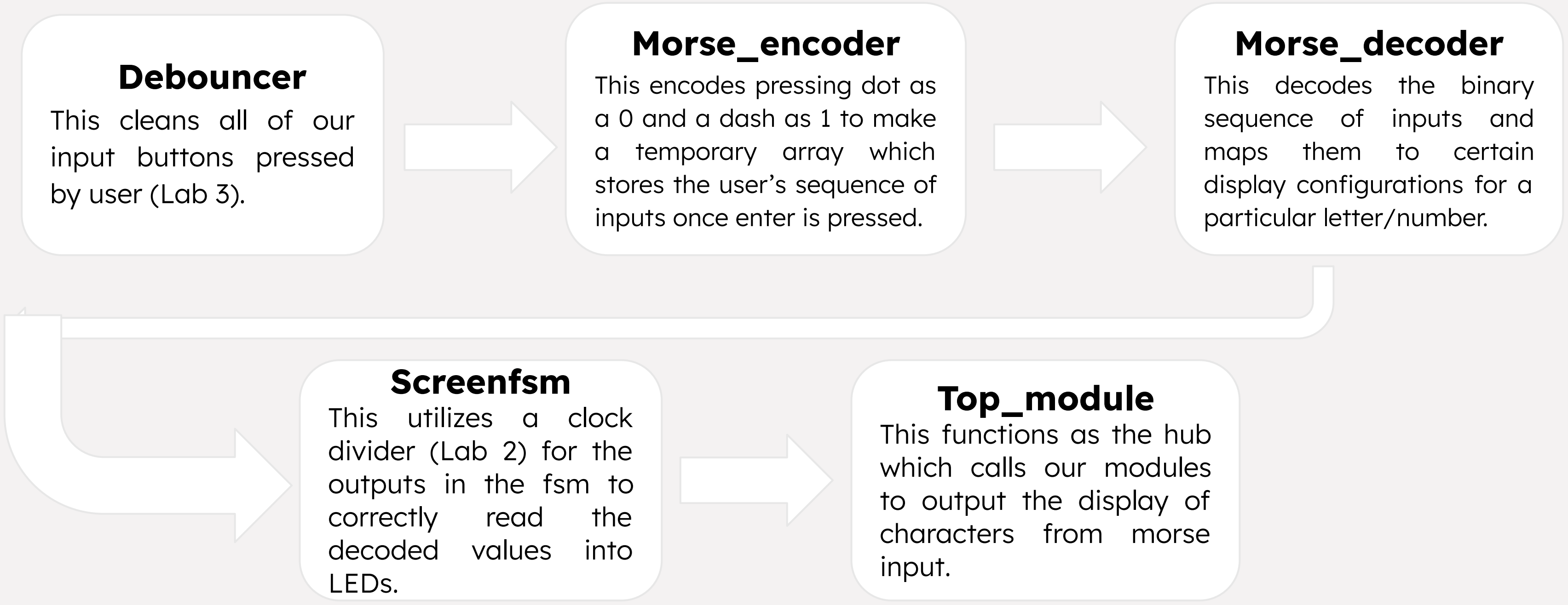
1. Debouncer (5 button inputs)
2. Morse Encoder (store position & input seq)
3. Morse Decoder (turn seq into character)
4. FSM (output characters to 7-seg display)

Constraints

The 7-segment display consists of four vertical segments (two on the left and two on the right) and three horizontal segments (forming the center lines). Since diagonal lines can't be lit, representing letters such as X and M is challenging. To address this, we designed our own versions of those letters as shown below:



State Machine Diagram



Debouncer

This cleans all of our input buttons pressed by user (Lab 3).

Morse_encoder

This encodes pressing dot as a 0 and a dash as 1 to make a temporary array which stores the user's sequence of inputs once enter is pressed.

Morse_decoder

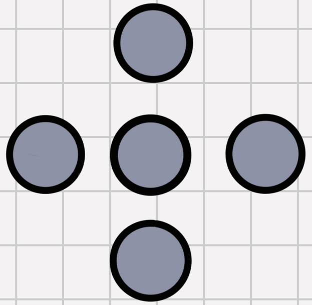
This decodes the binary sequence of inputs and maps them to certain display configurations for a particular letter/number.

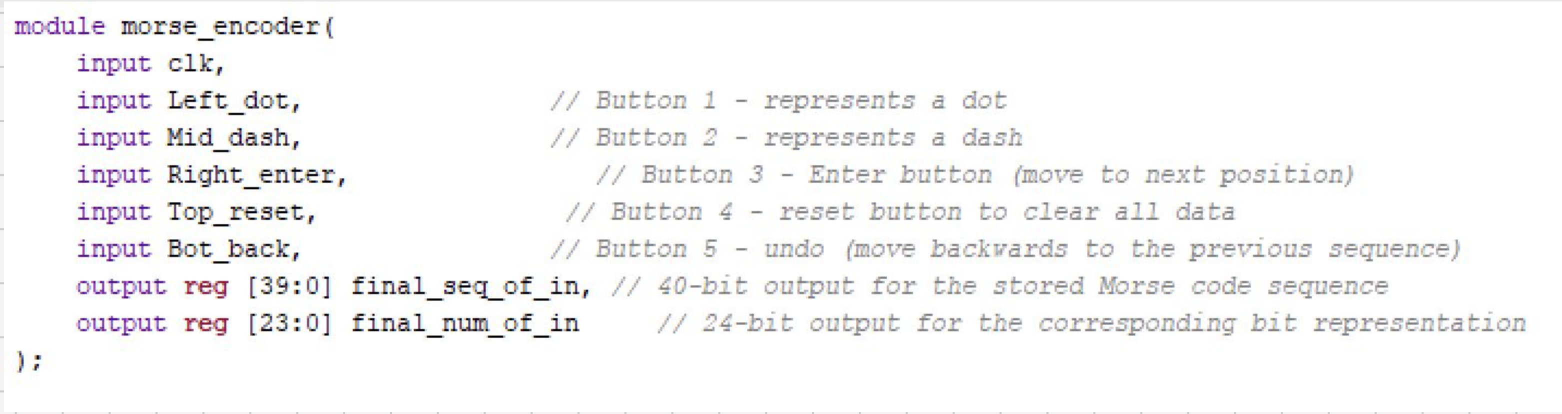
Screenfsm

This utilizes a clock divider (Lab 2) for the outputs in the fsm to correctly read the decoded values into LEDs.

Top_module

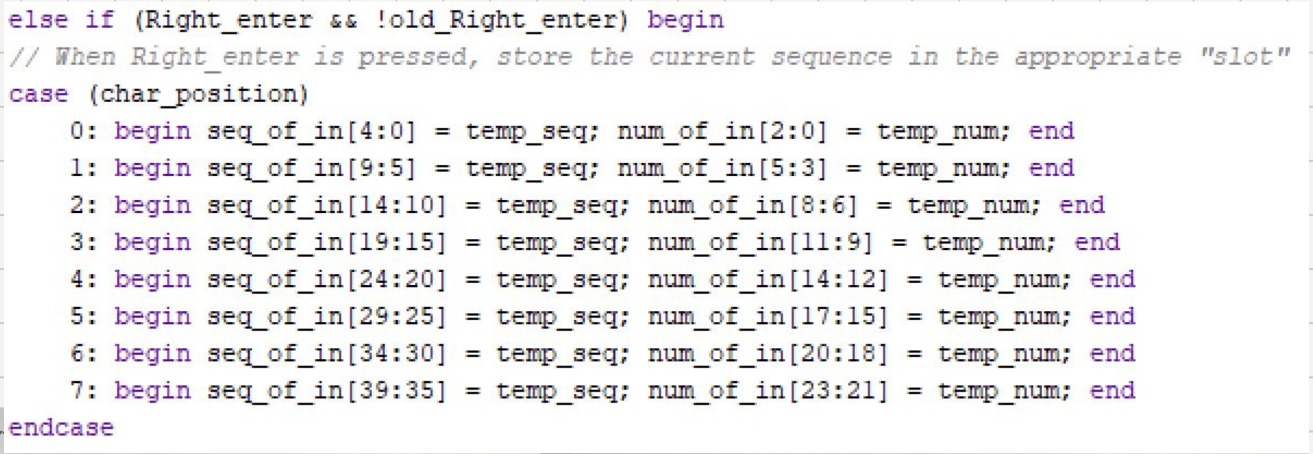
This functions as the hub which calls our modules to output the display of characters from morse input.



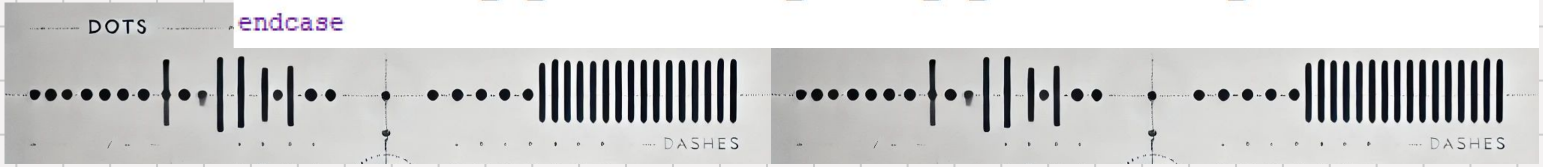


```
module morse_encoder(
    input clk,
    input Left_dot,                // Button 1 - represents a dot
    input Mid_dash,                // Button 2 - represents a dash
    input Right_enter,                // Button 3 - Enter button (move to next position)
    input Top_reset,                // Button 4 - reset button to clear all data
    input Bot_back,                // Button 5 - undo (move backwards to the previous sequence)
    output reg [39:0] final_seq_of_in, // 40-bit output for the stored Morse code sequence
    output reg [23:0] final_num_of_in     // 24-bit output for the corresponding bit representation
);
```

Snippet of Morse Encoder



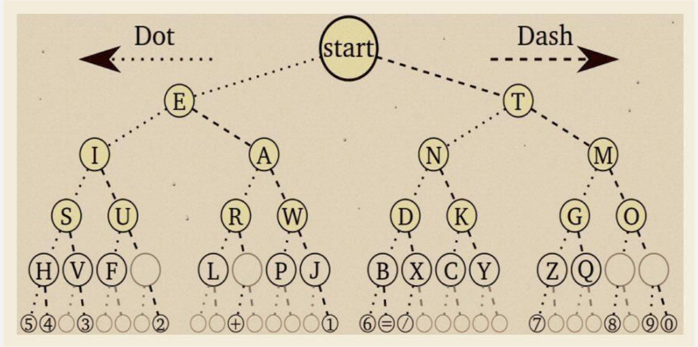
```
else if (Right_enter && !old_Right_enter) begin
// When Right_enter is pressed, store the current sequence in the appropriate "slot"
case (char_position)
    0: begin seq_of_in[4:0] = temp_seq; num_of_in[2:0] = temp_num; end
    1: begin seq_of_in[9:5] = temp_seq; num_of_in[5:3] = temp_num; end
    2: begin seq_of_in[14:10] = temp_seq; num_of_in[8:6] = temp_num; end
    3: begin seq_of_in[19:15] = temp_seq; num_of_in[11:9] = temp_num; end
    4: begin seq_of_in[24:20] = temp_seq; num_of_in[14:12] = temp_num; end
    5: begin seq_of_in[29:25] = temp_seq; num_of_in[17:15] = temp_num; end
    6: begin seq_of_in[34:30] = temp_seq; num_of_in[20:18] = temp_num; end
    7: begin seq_of_in[39:35] = temp_seq; num_of_in[23:21] = temp_num; end
endcase
```



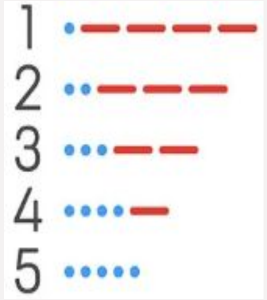
Morse Decoder Code Snippet

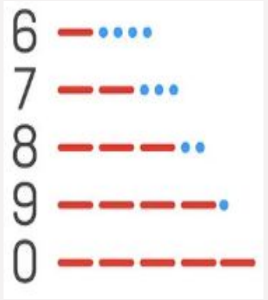
```
module morse_decoder (
    input [2:0] possible_inputs,       // takes in a 3-bit input to account for 6 cases of dot/dash inputs (2^3 = 8 > 6)
    input [4:0] possible_chars,        // takes in a 5-bit input since numbers can have up to 5 dashes/dots
    output reg [6:0] display;          // outputs a 7-bit register which represents the 7-segment display output
```

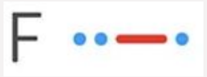
The 6 cases of dots & dash inputs are:
Case 1: no inputs (outputs a space),
Case 2: 1 input (can output 2 letters),
Case 3: 2 inputs (can output 4 letters),
Case 4: 3 inputs (can output 8 letters),
Case 5: 4 inputs (can output 12 letters)
Case 6: 5 inputs (can output numbers 0-9)



The most inputs we can have are 5 as all numbers need at least 5 total dots and dashes in Morse Code

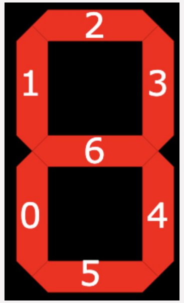


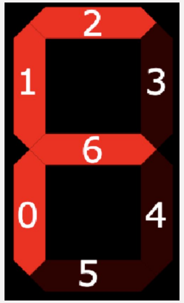


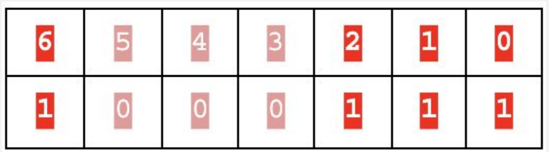


read right to left with 1 blank

{one_blank, 4'b0100}: out = 7'b1000111;

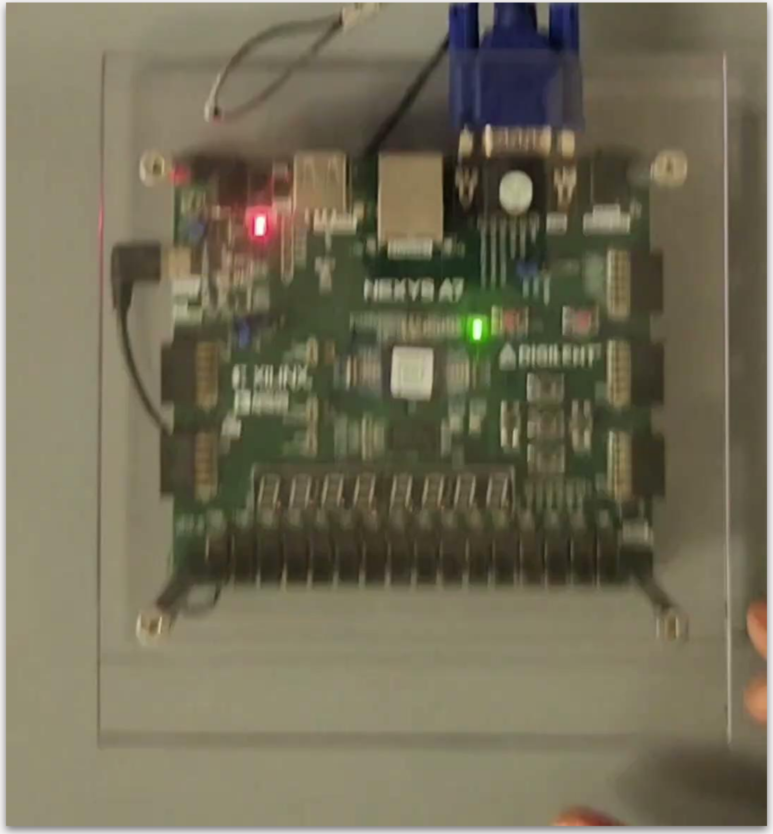






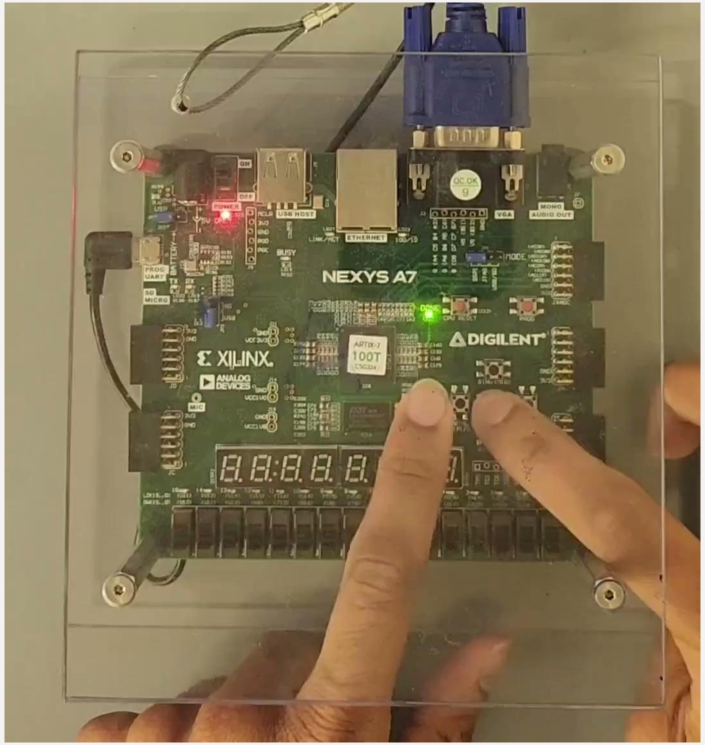
6	5	4	3	2	1	0
1	0	0	0	1	1	1

Demo



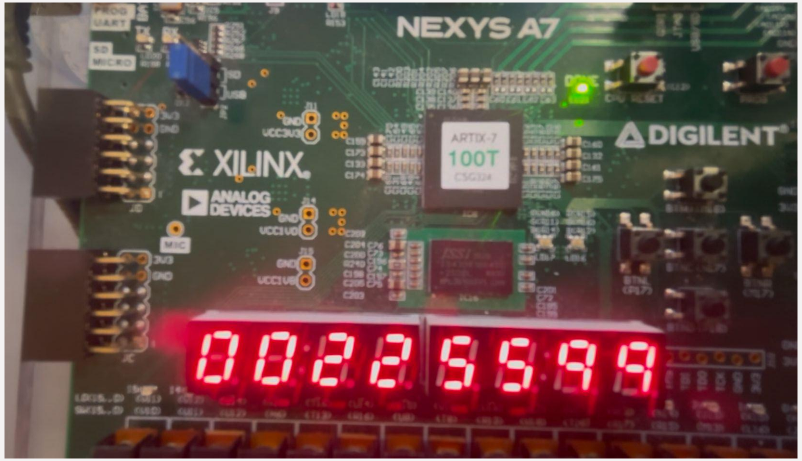
Successes

- Dot, dash, delete, reset, and enter buttons work as intended
- We can output all 26 letters of the English alphabet
- We can output all 10 numbers ranging from 0 to 9
- We can correctly index to the next position on the display per character
- We can correctly return to the first character position once the display is full



Weaknesses

- **Same symbol for numbers and letters**
  - Due to the inability to have diagonals on the 7 signal display, certain letters and numbers look exactly the same and the user would have no way to determine the difference.
    - **Example**: 2 & Z; 5 & S; q & 9
  - For next steps we want to incorporate the synchronization of LED lights, starting from the leftmost LED, to correspond with the placement of a number on the output display.



Sources and References

Clock divider (FSM Module): EC311 Fall 2024 Lab 2

Debouncer (Buttons): EC311 Fall 2024 Lab 3

Indexing Display Output (Morse Encoder Module): Verilog Arrays and Memories (Article)

Siekoo Alphabet/Number Design (Morse Decoder Module): The Siekoo Alphabet (Article)

Designing Characters (Morse Decoder Module): How to Control 7-Segment Displays Using Verilog (Video)

7 Segment Display (FSM Module Module): Up-Down-Counter---7-segment-display (Github)

Initializing Display Segments (FSM Module): FSM Module for AI Project (Github)

7 Segment Display Reference (Constraints File): 7-Segment LED Display Animator - 4-Digit (Github)

Questions?