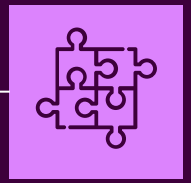
Morse Code Trainer

By: Ayin Pitman, Lorraine Graham, Mahnoor Ghani, and Vugar Amirov

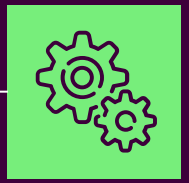
TABLE OF CONTENTS



01

Motivation + Functionality

Why we chose the project we chose, and how it works



02

Specification + Diagrams, and Code

More detailed explanations and diagrams of our system

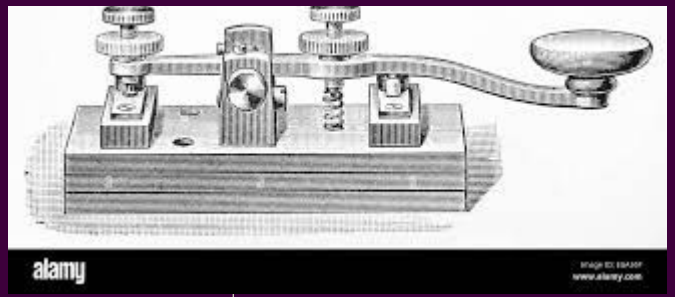
03

Reflection

What our project did well, and future improvements

01. Motivation

Create a simple translator that can continue the legacy of Morse Code and bring it's accessible features to a wider audience

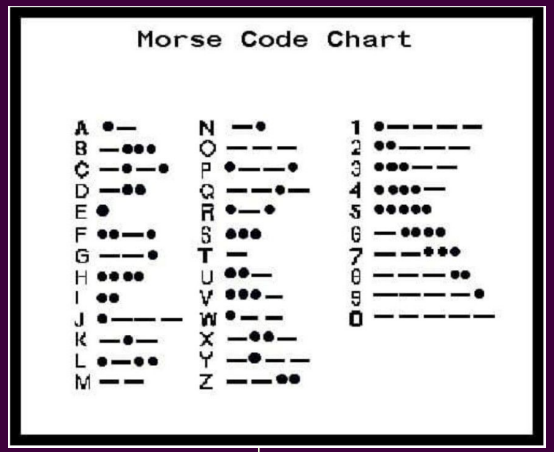


- The real-time translation creates an approachable learning environment for those new to Morse Code. It allows users to practice as the timing can be difficult to master.
- The fully tactile interface allows this tool to be used despite various environmental disturbances

01. Functionality

Real Time Translation

Our project's main goal is to translate from Morse Code into written english as you type!



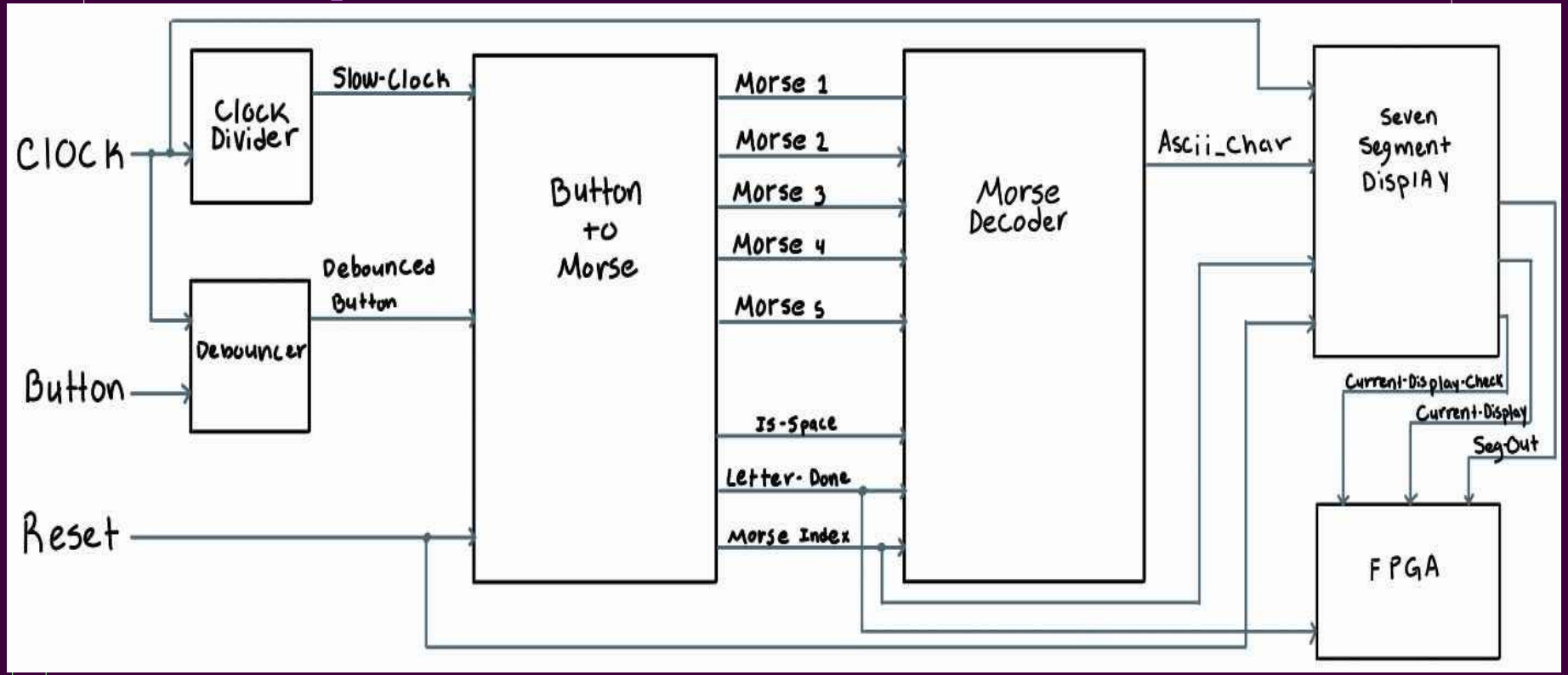
Text Display

and display the translated content onto a display for easy readability.

02. Specifications

- Process human input into Morse Code using one button
- Translate Morse Code into written English
- Have translated English appear on the display as it is being translated
- Include a delete button that deletes the last translated character
- Include a reset button
- Implement correct timing for Morse Code (dots, dashes, spaces, etc.)

02. Block Diagram



02. Code Snippets

Button_to_morse
and morse_decoder modules

```
if (counter >= one_time_unit) begin
    // Determine dot or dash based on press duration
    if (counter >= three_time_units) begin
        case (morse_index) // Dash
            3'b000: latched_morse[0] <= 2'b10;
            3'b001: latched_morse[1] <= 2'b10;
            3'b010: latched_morse[2] <= 2'b10;
            3'b011: latched_morse[3] <= 2'b10;
            3'b100: latched_morse[4] <= 2'b10;
            default begin
                latched_morse[0] <= 2'b00; latched_morse[1] <= 2'b00; latched_morse[2] <= 2'b00;
                latched_morse[3] <= 2'b00; latched_morse[4] <= 2'b00;
            end
        endcase
    end else begin
        case (morse_index) // Dot
            3'b000: latched_morse[0] <= 2'b01;
            3'b001: latched_morse[1] <= 2'b01;
            3'b010: latched_morse[2] <= 2'b01;
            3'b011: latched_morse[3] <= 2'b01;
            3'b100: latched_morse[4] <= 2'b01;
            default begin
                latched_morse[0] <= 2'b00; latched_morse[1] <= 2'b00; latched_morse[2] <= 2'b00;
                latched_morse[3] <= 2'b00; latched_morse[4] <= 2'b00;
            end
        endcase
    end
end
```

Dot or Dash?

The code determines if the user input is a dot or dash depending on timing.

If held for a minimum of one clock cycle, the code detects a dot. If held for a minimum of 3 clock cycles, the code detects a dash.

Once the symbol is correctly identified, the index is incremented so it can detect the next symbol.

User Input ⟶ Morse Code

```
 if (inactivity_counter >= three_time_units && !latched_done && !is_space)
begin
            morse_one <= latched_morse[0];
            morse_two <= latched_morse[1];
            morse_three <= latched_morse[2];
            morse_four <= latched_morse[3];
            morse_five <= latched_morse[4];
            letter_done <= 1; // Detects finished letter
            latched_done <= 1; // Detects when latching is done
. . .

 if (inactivity_counter >= seven_time_units && latched_done) begin
            is_space <= 1; // Set is_space signal high
            letter_done <= 0; // Reset letter_done to avoid processing new
symbols
            morse_index <= 0;
            latched_morse[0] <= 2'b00;
            latched_morse[1] <= 2'b00;
            latched_morse[2] <= 2'b00;
            latched_morse[3] <= 2'b00;
            latched_morse[4] <= 2'b00;
        end
```

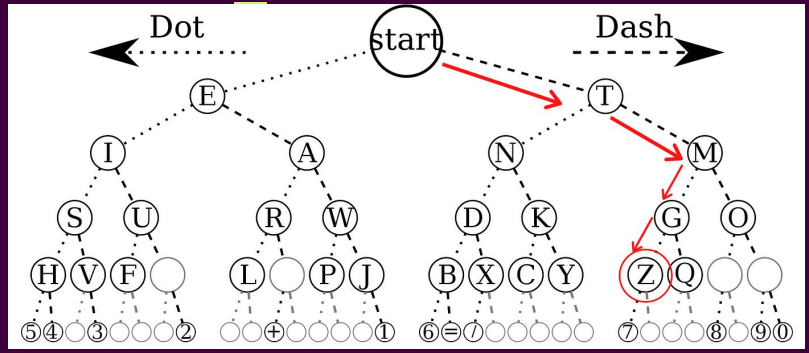
If button is inactive for at least three time units, code detects that the letter is done

If inactive for at least seven units, a space is detected and index is reset to allow for the user to input a new character

Dots/Dashes to Ascii Binary

```
if (is_space) begin
      ascii_char = " ";
      last_valid_char = ascii_char;
      end else if (letter_done) begin
      // Start traversal of binary tree
      if (morse_one == 2'b01) begin // Dot (E branch)
          if (morse_two == 2'b01) begin // Dot (I branch)
              if (morse_three == 2'b01) begin // Dot (S branch)
                  if (morse_four == 2'b01) begin // Dot (H branch)
                      if (morse_five == 2'b01) begin // Dot
                          ascii_char = "5"; // .....
                      end else if (morse_five == 2'b10) ascii_char = "4";


ascii_char = "A"; →  . -
ascii_char = "B"; →  - . . .
ascii_char = "C"; → - . - .
```

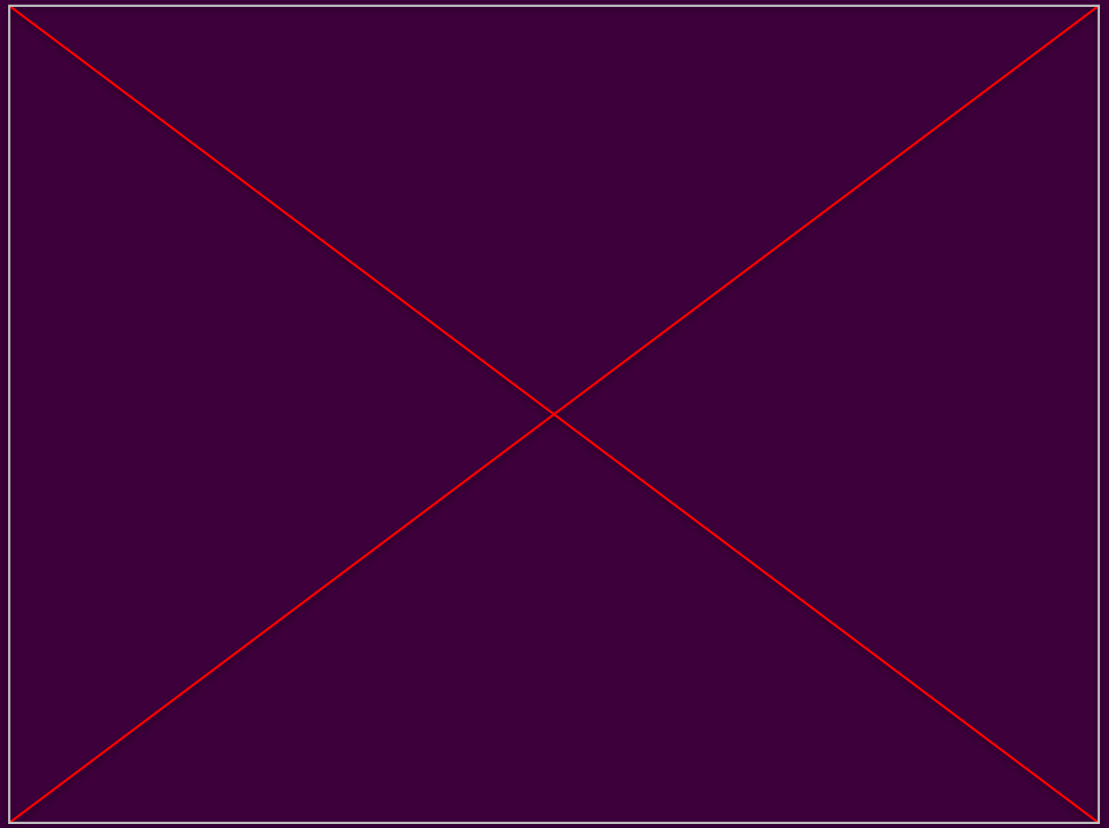


Using IF statements to implement the binary tree

Any letter or number can have maximum of 5 dots or dashes therefore the code has 5 variables

After finding the correct letter, it returns its ASCII value

Video



03. Successes

1. Only one button used for both dot and dash inputs

2. Mapping morse code to character is correct

3. All modules work together in the top module

4. 7-Segment display properly displays letters

5. Clock divider properly changes unit time

03. Failures

1. UART + VGA display
2. Didn’t accomplish adding the delete button
3. Space is not yet functioning on the display.

How We Would Improve

(with more time)

- Implementing the off-FPGA display
- Implementing the delete button
- Adding dynamic error correction
- Adding high contrast modes for display
- Adding text to speech functionality

03. What we wish to accomplish

- Space is implemented after 3 time units of button inactivity
- Switch inputs to change timing of clock
  - currently one clock speed of 4Hz (1 unit = ¼ s))

Thanks for listening!

Questions?