Project Proposal: Morse Code Button Tapper and Decoder

Ziyao Xiong

***Abstract*—This project implements an interactive Morse code encoder and decoder using a push button and keypad interface. Users input Morse sequences via a push button, where short and long presses represent dots (.) and dashes (-), respectively. The sequences are displayed on an LCD in real-time. Upon pressing the 'D' key on the keypad, the system decodes the stored sequence by referencing a predefined lookup table and displays the resulting characters or numbers on the LCD.**

I. Introduction

MORSE code, developed by Samuel Morse in the 1830s, revolutionized communication by encoding messages as dots (.) and dashes (-) [1]. Historically, decoding Morse code was a manual process performed by skilled operators who interpreted the rhythmic patterns by ear or visually through telegraph signals. While effective, this method required extensive training and focus, limiting accessibility to non-specialists. This project aims to modernize and simplify the decoding process by combining a push button for Morse code input with an automated decoding system. Input sequences are displayed on an LCD in real-time, and a keypad key triggers decoding into readable text. A buffer stores sequences, while a lookup table ensures efficient and accurate pattern matching. By combining historical communication with modern embedded design, this project highlights Morse code principles in a user-friendly, interactive platform.

II. Design

The system is built around the PIC18F87K22 microcontroller, which features built-in RAM for temporary data handling, Flash memory for permanent data storage, versatile I/O ports for connecting peripherals, and timers for precise timing operations. These capabilities make it well-suited for managing Morse code sequences, decoding patterns, and interfacing with the push button, keypad, and LCD. Fig. 1 illustrates the system's high-level structure, detailing component connections and data flow.

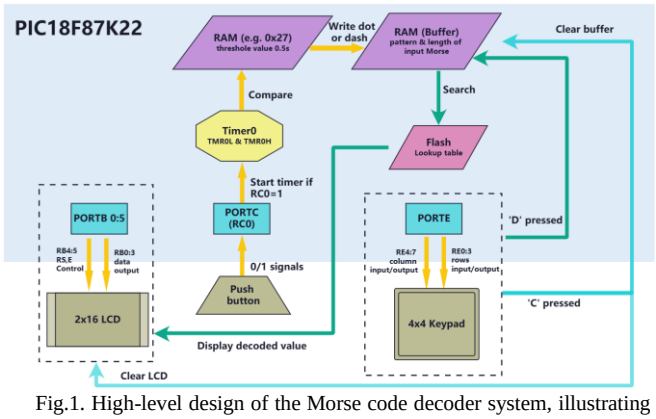

Fig.1. High-level design of the Morse code decoder system, illustrating the microcontroller's integration with peripherals and key components.

The hardware design is implemented using the EasyPIC PRO v7 development board, which incorporates the PIC18F87K22 microcontroller. A push button (model RJS-GME1-Y-12V-D) is connected to PORTC (RC0) for Morse code input, utilizing Timer0 to distinguish between dots and dashes based on signal duration. A 4x4 keypad, interfaced through PORTE, is used for command inputs such as clearing the buffer and triggering the decoding process. The 2x16 LCD display, connected via PORTB, provides real-time feedback for Morse code input and decoding results. The hardware schematic of the system is shown in Fig.2.

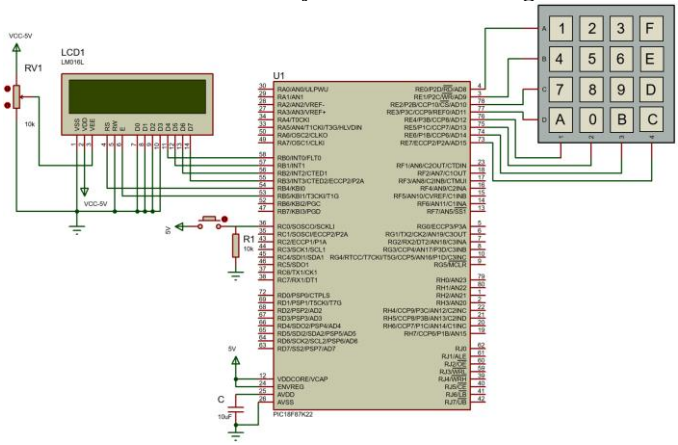

Fig. 2. Schematic diagram of the Morse code decoder system, highlighting the connections between the PIC18F87K22 microcontroller, push button, keypad, LCD, and supporting components.

The circuit enables the on-chip voltage regulator of the microprocessor to provide a stable 3.3V supply to its core logic while external components, such as the LCD operating at 5V, function reliably. The VDDCORE pin connects to a 10µF capacitor for stabilizing the regulator's output [2]. The ENVREG pin is tied to VDD to activate the regulator.

1) Push Button

The push button in the system serves as the primary input device for generating Morse code signals. It is connected to the microcontroller via the RC0 pin. The button's COM (common) terminal is connected to a 3.3V source, while the NO (normally open) terminal is connected to RC0 and pulled down to ground through a 10 kΩ resistor. When the button is not pressed, the pull-down resistor ensures that RC0 reads a logic-low (0) signal. When the button is pressed, the circuit is completed, and RC0 detects a logic-high (1) signal.

2) Timer

The timer is crucial for accurately measuring the duration of Morse code signals to distinguish dots (<0.5 s) and dashes (>0.5 s). Timer0 is configured in 16-bit mode to maximize its counting range (2^{16} ticks) and reduce interrupt frequency, as 8-bit mode would require frequent handling of overflows. The internal clock was chosen for its simplicity and reliability, eliminating the need for external components.

A 1:256 prescaler is selected, providing an increment interval of 16 µs, which results in an overflow time of:

$$t_{overflow} = 16\,\mu s \cdot 2^{16} = 1.0485 s. \quad (1)$$

This interval exceeds the threshold of 0.5 seconds used to differentiate dots and dashes while maintaining sufficient resolution to capture timing details accurately.

For durations exceeding 1.0485s, an overflow counter is implemented. When an overflow occurs, an interrupt service routine (ISR) increments the counter, and the total elapsed time when there is an overflow count n is calculated as:

$$t_{elapsed} = n \cdot t_{overflow} + 16\,\mu s \cdot t_{current}, \quad (2)$$

where $t_{current}$ is the timer's value at the time of measurement.

3) Buffer and Lookup Table

Following the Timer's role in distinguishing dots and dashes, the buffer serves as temporary storage for processed Morse code sequences. It is a 6-byte memory block stored in

RAM, designed to accommodate the longest Morse code sequence of 5 symbols, with 1 byte reserved for storing the sequence length. The buffer supports interactions from both the 'C' and 'D' keypad commands—'C' clears it, while 'D' triggers decoding. The buffer ensures fast, dynamic updates, essential for real-time Morse code input and decoding.

The lookup table is stored in Flash memory and defines the Morse code patterns and corresponding lengths for all supported letters and numbers. During decoding, this structure enables efficient comparisons by first matching the length of the input sequence and then verifying the Morse code pattern. This two-step approach reduces computational overhead and ensures accurate decoding, making it well-suited for real-time applications. By storing the lookup table in Flash, the system benefits from persistent, non-volatile data storage, ensuring reliability across power cycles.

4) Keypad

The keypad in the system is a 4x4 matrix connected to PORTE of the PIC18F87K22 microcontroller. Rows (RE0-RE3) and columns (RE4-RE7) dynamically switch roles between inputs and outputs during scanning. The microcontroller drives one set (e.g., columns) low and reads the other set (e.g., rows) as inputs with internal pull-ups enabled. When a key is pressed, the corresponding row and column are detected, and the system executes the associated function. For example, pressing 'C' clears the buffer and LCD, while pressing 'D' triggers the decoding process.

5) LCD (Liquid Crystal Display)

A 2x16 character LCD is connected to PORTB of PIC18F87K22 in 4-bit data mode. Communication involves sending data or commands in two nibbles via the data lines (RB0-RB3), while the RS line specifies whether the input is a command (RS=0) or data (RS=1), and the E line is pulsed to latch the input. VEE is connected to a potentiometer to adjust the display contrast for optimal readability.

The LCD provides programmatic control over the cursor position, ensuring characters are displayed in the correct sequence and position. During Morse code input, the full sequence of dots and dashes is displayed in real time, with the cursor moving to the next position after each entry. This ensures that the input sequence remains visually clear and easy to follow. When the 'C' key is pressed to clear the buffer, the cursor resets to the starting position. Upon pressing the 'D' key, the decoded character is displayed on the second row of the LCD to prevent any overlap or confusion in the display.

Building on the functionality of these components, the software design unifies them into a cohesive system. Fig. 3 illustrates the detailed flow of operations of the system.

III. Assessment of Performance

The performance of the constructed system will be assessed to ensure the system meets design specifications. Functional tests will focus on evaluating the system's stability and recovery under stress conditions, including rapid consecutive inputs, prolonged pauses, very short or extended keypresses, and varying sequence lengths. These tests will verify the system's ability to process diverse Morse code sequences accurately. Invalid Morse code patterns will also be tested to validate robust error handling. Timing accuracy and decoding reliability will be evaluated to confirm correct classification of dots and dashes near the 0.5-second threshold and accurate outputs. Additionally, the response time from input detection to display update will be measured to ensure smooth and real-time interaction. Memory usage will also be evaluated to confirm efficient utilization of the microcontroller's resources, ensuring the system operates reliably without exceeding hardware constraints.

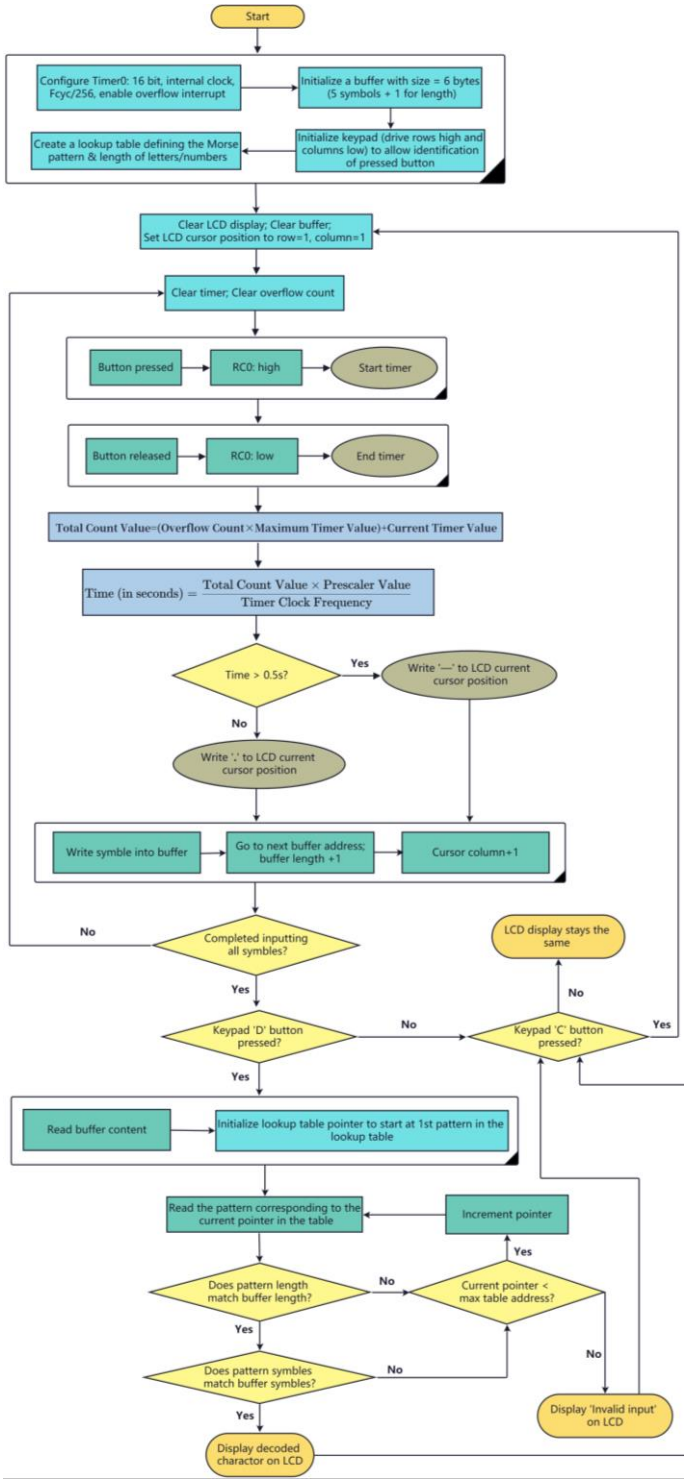

Fig. 3. Flowchart illustrating the software design, including initialization, input processing, and result display.

IV. Conclusion

This proposal outlines a real-time Morse code decoding system using the PIC18F87K22 microcontroller, focusing on efficient timing, memory management, and user interaction via a push button, keypad, and LCD. It aims to process and display Morse code sequences in real-time. Future work could enhance functionality by supporting multi-character decoding, enabling users to input and decode entire words or phrases, expanding the system's versatility and applications.

REFERENCES

[1] Department of the Army, International Morse Code (Instructions), Washington, DC, USA: U.S. Government Printing Office, 1968, pp. 6–7.

[2] Microchip Technology Inc., PIC18F87K22 Family Data Sheet, 2009. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/39960d.pdf. [Accessed: Nov. 23, 2024].