An ESP32-Based Morse Code Transmission System Using Telegram Bot

Chandramouli Haldar *

Posted Date: 28 February 2026

doi: 10.20944/preprints202602.1999.v1

Keywords: morse code; ESP32; IoT; secret messaging; real-time communication

 doi:10.20944/preprints202602.1999.v1

Article

An ESP32-Based Morse Code Transmission System Using Telegram Bot

Chandramouli Haldar

Independent Researcher, NovaTech Innovative Solutions, Kolkata, India; chandramoulihaldar@gmail.com

Abstract

This paper introduces a Morse code transmission system utilizing an ESP32 microcontroller and displaying the decoded message via a Telegram bot in real-time. In contrast to the traditional Morse code inputting methods, which are normally based on a single button with timing distinction between dot and dash, the suggested design utilizes two special buttons for dot and dash, respectively, and another button for sending the entire message. This method simplifies the input of users, minimizes timing errors, and enhances accuracy in message delivery. The decoded text is then transmitted securely to a specified Telegram chat via the Bot API at high speed and reliability over Wi-Fi. The system is portable, lightweight, and compact, thereby ideal for covert or clandestine messaging without inviting unnecessary attention.

Keywords: morse code; ESP32; IoT; secret messaging; real-time communication

I. Introduction

Morse code is one of the earliest and most enduring forms of digital communication, developed by Samuel Morse and Alfred Vail. It represents letters, numbers, and symbols through sequences of short and long signals, commonly referred to as "dots" and "dashes." (Shown in the Figure 1.) Morse code, by convention, had been transmitted by telegraph keys, flashing lamps, or tone sounds, and for many decades had been the foundation of long-distance communication. Although formally invented historically, Morse code endures today due to the simplicity, the universality, and the fact that it conveys information despite limited resources.

In this work, for the purposes of providing a Morse code input and transmission system for a modern IoT application, a system employing an ESP32 microcontroller is presented. Contrary to the traditional model whereby a one-button configuration operates and the user must accurately time presses between the dot and dash, this system provides a user-friendlier input procedure. Dots and dashes each have a dedicated button: one for the dots and one for the dashes. A third acts as the transmit button for the compiled message. The methodology eliminates the need for accurate timing, reduces the risk for the human, and makes learning less intimidating for the less-skilled user employing the traditional Morse keying. The system operation is simple. The dot or dash button press respectively adds the associated symbol to the current Morse sequence. The sequence is decoded for an alphabetic character when a short idle time is sensed. The characters thus built up are added to give words or sentences. When the user depresses the send button, the decoded message is sent by a secure connection through the ESP32 via Wi-Fi to a Telegram bot, which immediately deposits the message in a chosen chat. For debugging purposes, all inputs and decoded characters also appear on the Serial Monitor.

The union of the Morse code with the Telegram communication app provides a few real-world applications. The device can act as a communication assistant for those with special needs for whom the use of buttons as opposed to full keyboards would be easier. The system may also find use when teaching students Morse code as a form of interactive and fun learning. Due to the compact structure of the hardware, the device allows for great mobility and has the capability of being used for passing

 doi:10.20944/preprints202602.1999.v1

secret information without raised attention, providing potential usage when it comes to matters of security or stealth communication.

In conclusion, the system marries the classic ease of Morse code with the present-day functionality of IoT and instant messaging platforms. Utilizing two special buttons for dot and dash improves usability, while the ESP32 guarantees smooth connectivity and inexpensive implementation. What emerges is a tiny, portable, and handy device that illustrates how old-fashioned communication techniques may be re-designed to suit modern purposes.

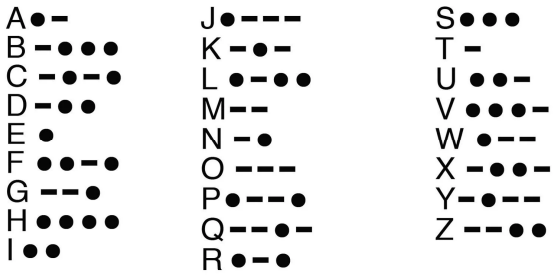


Figure 1. Morse Code Chart.

II. Background And Related Work

Morse code, invented by Alfred Vail and Samuel Morse during the 1830s, is a very early form of binary communication, conveying letters and digits by means of short and long signals, referred to as dashes and dots. Simple, with a low bandwidth usage, and reliable, Morse code became the standard of military, maritime, and aviation communication for over a century. Even today, Morse code enjoys a place of importance due not only to the robustness it imparts to emergency communication but also due to the fact that it operates with minimum hardware support [1].

Traditional Morse systems usually depended on one key such that the duration of the press would indicate whether the signal would be a dot or a dash (refer to Figure 2). Though effective with trained hands, this required training and attention to detail and thus remained less accessible to untrained operators. To simplify this, Nalajala et al. [2] suggested a Morse code generator based on an 8051 microcontroller and an alphanumeric keypad. Their system meant users could input letters as well as digits directly, which the microcontroller would translate into Morse code for transmission by means of LEDs, buzzers, and LCD displays. This eliminated the need for remembering Morse sequences while carrying the reliability of Morse-based communication.

Figure 2. Traditional Morse Code Machine.

Recent research has also taken Morse code into new input and assistive technologies. Banerjee et al. [3] built a Morse code generator using the Arduino UNO at a low cost as part of a demonstration of educational and emergency communication. Tevaramani et al. [4] developed a Morse code decoder

 doi:10.20944/preprints202602.1999.v1

using an Arduino UNO and LCD, making decoding easier for students. Gueorguieva et al. [5] discussed Morse code as an input for limited and headless computer systems, revealing the flexibility of Morse code for human-computer interaction. Likewise, Bhatt et al. [6] suggested the concept of "Blink-to-Code," wherein eye blinking could be monitored and converted into Morse sequences, revealing the potential for Morse code-based assistive communication.

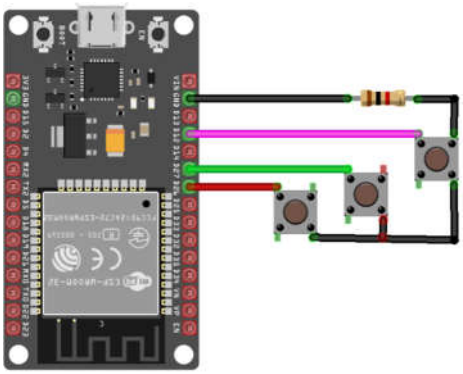
Along with this, IoT- and microcontroller-based communication platforms have expanded exponentially. ESP32, possessing Wi-Fi and secure communication features, became a popular platform for IoT-based applications. The researchers utilized them with message platforms such as Telegram for creating efficient real-time communication tools. Hutajulu et al. [7] presented a smart home automation system using ESP32 with a Telegram bot, which supports remote monitoring as well as control of appliances.

The research suggests the usability of the Telegram as a lightweight, secure, and globally accessible message platform for IoT applications. The current work advances these beginnings by integrating Morse code with up-to-date IoT communication. As opposed to traditional single-button input, our system incorporates two special buttons, one for dots and one for dashes, as well as a third one for sending the finished message. This configuration diminishes input complexity, lessens timing mistakes, and enhances usability relative to prior methods. In turn, by integrating with Telegram, the system provides decoded output instantaneously over the internet, widening the range of Morse communication from local buzzer/LCD output [2,4] up to the level of global message platforms. The compact, inexpensive structure allows for appropriate usage not only for emergency communication but also for stealthy and covert communication.

III. Proposed System

The proposed system is a compact communication device that merges simple Morse code with the networking capabilities of a typical IoT device. Conventional Morse code transmission requires a single button in which the user holds down the button for the duration of the time they want to signal dot or dash, and to complicate this, Morse uses spaces as well, either between letters or words. This proposed method provides three buttons. One button will generate the dot letter indicator, one will generate the dash letter indicator, and the third will signal the completed message signal to be sent. This variation of transmission will reduce the need for timing precision and reduces the opportunity of a user being incorrect. Therefore, it is more accessible for someone who has never used a typical telegraph key to input Morse code.

The ESP32 microprocessor is the processing unit, receiving input from the user buttons, decoding the Morse into letters, and sending the messages. Once the user has generated a message, the text generated from the Morse is sent securely and over Wi-Fi to a bot connected to telegram which delivers it instantly to a designated chat. Therefore, there is access to communications anywhere in the world because the user has Telegram on their device to receive the decoded Morse message. The whole system is low-cost, energy-efficient, and portable, making a great solution for anything from emergencies, assisting technology, education, and easily communicating discreet or covert messages.



 doi:10.20944/preprints202602.1999.v1

Figure 3. Graphical Circuit Diagram.

A. System Design and Architecture

The system architecture consists of three layers: input, processing, and communication. Three physical push buttons in the input layer read user input: dot, dash, and send. These are connected to the ESP32's GPIO pins with internal pull-up resistors to ensure stable signal detection.

In the processing layer, the ESP32 interprets button presses and builds Morse sequences. A time-based idle mechanism (2 seconds) identifies the end of each character and triggers the decoding of the Morse sequence into its alphanumeric counterpart. The decoded character is appended to a message buffer, which is refreshed constantly as additional symbols are input by the user.

The ESP32 sends messages encrypted using its onboard Wi-Fi module and the Universal Telegram Bot API in the communication layer. HTTPS protocol takes care of the data integrity since the Telegram bot can send the message directly into a given chat ID. For transparency, the system prints debugging data to the Serial Monitor as well. This information includes any button clicks, Morse codes, decoded characters, and even confirmation of sent messages.

The layer architecture demonstrates modularity; for instance, the input device could be easily replaced by a different sensor (e.g., touchpads or blink sensors) and would not even necessitate changing the communication layer.

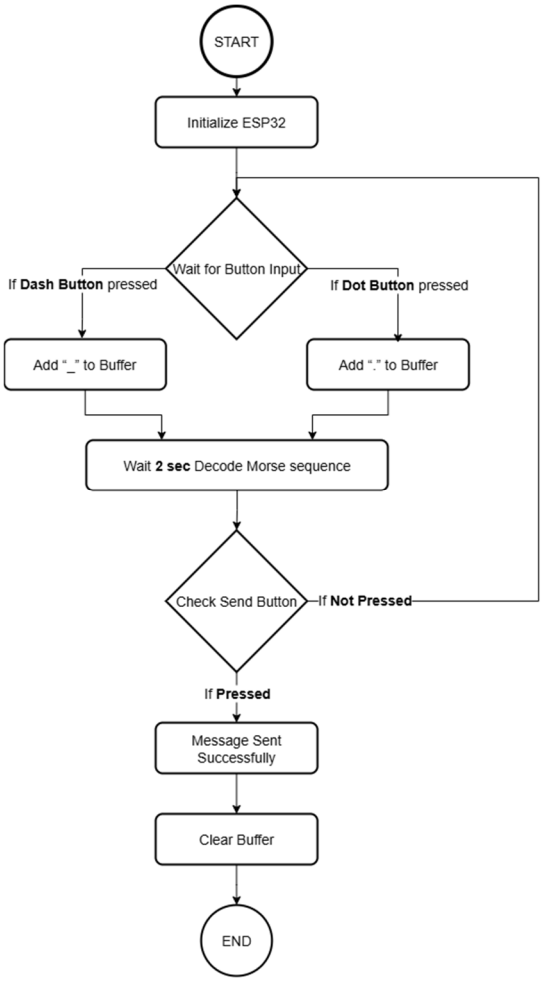


Figure 4. Flowchart of the System.

B. Implementation

The system was implemented with the Arduino IDE supporting ESP32 development. Libraries WiFi.h, WiFiClientSecure.h, UniversalTelegramBot.h, and ArduinoJson.h were utilized for

 doi:10.20944/preprints202602.1999.v1

connectivity and messaging purposes. The secure bot token and chat ID, retrieved from Telegram's BotFather, were included in the code for direct communication purpose.

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>

// === WiFi & Telegram Setup ===
const char* ssid = "SSID";
const char* password = "password";
#define BOTtoken "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
#define CHAT_ID "xxxxxxxxxxxxx"

WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);
```

Figure 5. Snippet of Library function and Bot Configuration.

For button input handling, presses were read with digitalRead() and debounced with a 100 ms wait to prevent multiple triggers. A press added either "." or "–" to the Morse sequence, with direct logging to the Serial Monitor for debugging purposes. In absence of a button press within 2 seconds, the system treated it as the end of a character and decoded the sequence by calling a lookup function for International Morse Code. Any unrecognized sequence was substituted with "?".

```
  if (morseSequence.length() > 0 && millis() - lastInputTime > idleTime) {
    char decodedChar = decodeMorse(morseSequence);
    typedMessage += decodedChar;
    Serial.print("Decoded Letter: ");
    Serial.println(decodedChar);
    morseSequence = "";
  }

// === Morse decoding function ===
char decodeMorse(String code) {
  if (code == ".-") return 'A';
  if (code == "-...") return 'B';
  if (code == "-.-.") return 'C';
  if (code == "-..") return 'D';
  if (code == ".") return 'E';
  if (code == "..-.") return 'F';
  if (code == "--.") return 'G';
  if (code == "....") return 'H';
  if (code == "..") return 'I';
  if (code == ".---") return 'J';
  if (code == "-.-") return 'K';
  if (code == ".-..") return 'L';
  if (code == "--") return 'M';
  if (code == "-.") return 'N';
  if (code == "---") return 'O';
  if (code == ".--.") return 'P';
  if (code == "--.-") return 'Q';
  if (code == ".-.") return 'R';
  if (code == "...") return 'S';
  if (code == "-") return 'T';
  if (code == "..-") return 'U';
  if (code == "...-") return 'V';
  if (code == ".--") return 'W';
  if (code == "-..-") return 'X';
  if (code == "-.--") return 'Y';
  if (code == "--..") return 'Z';
  return '?';
}
```

Figure 6. Snippet of morse code function.

If the send button was clicked, the message in the buffer was sent by the API and Wi-Fi client to the Telegram bot. The transfer time of the message would typically be between 1-2 seconds based on network conditions, and the message delivered would be shown in the Serial Monitor. Once the message had been sent off, the message was flushed from the buffer in anticipation of new input.

The prototype validated that the system offers proper Morse decoding and guaranteed message transmission via Telegram. The hardware's compactness and simplicity, along with real-time global messaging functionality, make this proposed system feasible for day-to-day and emergency communication.

 doi:10.20944/preprints202602.1999.v1

IV. Results and Analysis

The proposed Morse code transmission system in this paper incorporates hardware input, software decoding, and wireless transmission over Telegram. The three push buttons for dot, dash, and send message are constantly monitored by an ESP32 microcontroller. The state of each press is immediately debounced and buffered into a temporary Morse sequence. Real-time indicators of the pressed buttons are also displayed on the Serial Monitor. For instance, typing ".-" shows up as "Dot pressed" and "Dash pressed," which, after a brief idle time, gets decoded into the character A. Several characters get buffered into a message buffer, and clicking the send button causes the ESP32 to package and send the message over the Telegram Bot API securely over a HTTPS connection. The decoded text shows up as a message in the receiver's Telegram chat within a few seconds, providing fast and reliable communication.

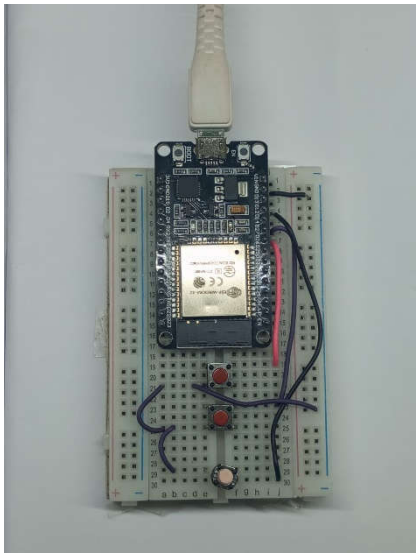


Figure 7. Hardwire Prototype.

Figure 6 depicts the system's hardware prototype consisting of the ESP32 development board interfaced with three tactile push buttons as shown on a breadboard. The compact form illustrates the system's potential for miniaturization and eventual integration into a handheld unit.

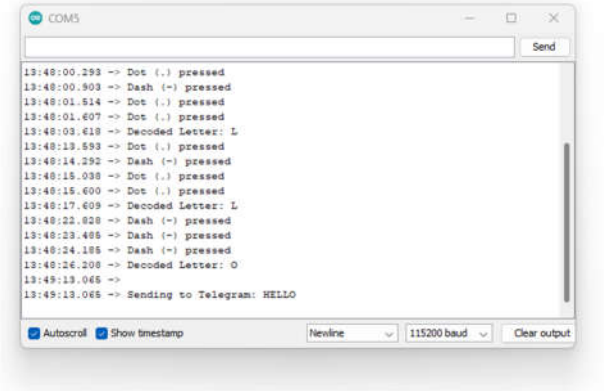


Figure 8. Serial Monitor output.

Figure 7 shows an example of the output on the Arduino Serial Monitor. In this case, the user pressed the dot button once, and the dash button once, thus creating a sequence of ".-". After two seconds of no input (there is a pause), the machine subsequently interpreted the sequence and displayed the text "Character decoded: A." You are able to type sequences one after the other to form

 doi:10.20944/preprints202602.1999.v1

words such as “HELLO.” Each action that you perform can be seen on the Serial Monitor, step by step.

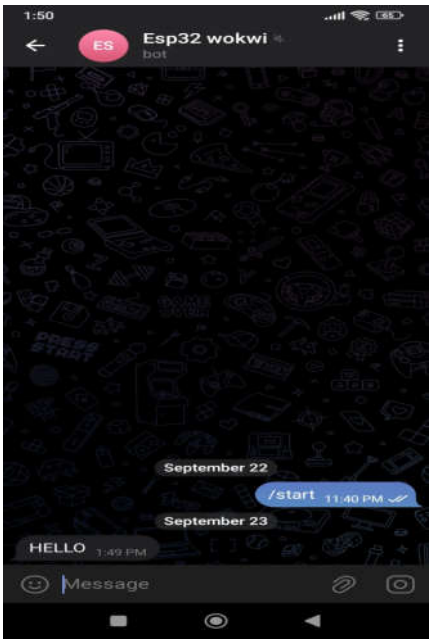


Figure 3. Telegram chat output.

Figure 8 shows the Telegram chat output, which resulted from the successful transmission of the Morse code typed last. For instance, after typing the Morse sequence for the word “HELP” and clicking the send button, the word “HELP” appeared in the Telegram chat window in less than two seconds. This ensures not only local Morse input processing by the system but also successful delivery of the Morse input to a remote platform.

The end-to-end operation of the system reduces the complexity. From the moment the button is pressed until the last message reaches the user, the whole sequence of operations is intuitive, extremely fast, and extremely accurate. The system does away with the time complexities inherent in the older Morse keying, and the Telegram integration makes the system operational from any place with internet connectivity. This aspect makes the device highly effective for applications like emergency alerts, secret communication, and assistive communication for those who may not easily use the keyboard.

V. Conclusion and Future Prospects

Morse code transmission using ESP32 system presented herein solely illustrates a transportable, effective, and uncomplicated mode of message transmission through Telegram. The system reduces the timing errors of the Morse keying by the inclusion of special dot, dash, and send buttons, thus making communication less intimidating for beginners. The Telegram compatibility ensures real-time message delivery anywhere in the world, thus rendering the device fit for the case of emergency communication, assistive technology, learning, as well as covert or surreptitious message communication by virtue of the compactness.

To further the future potential, the system may also be improved by the inclusion of LoRa communication for enabling long-distance transmission without the use of the internet, thus becoming effective when used in remote or disaster-affected communities. The system may also be further advanced by the use of AI/ML algorithms for the incorporation of an autocorrect capability that identifies and corrects Morse input mistakes for increased correctness and usability. These would further widen the applicability of the system reliability in the real world.

 doi:10.20944/preprints202602.1999.v1

References

1. L. P. Carron, Morse Code: The Essential Language. Newington, CT: American Radio Relay League, 1986.
2. P. Nalajala, B. Godavarthi, M. L. Raviteja, and D. Simhadri, "Morse Code Generator Using Microcontroller with Alphanumeric Keypad," in Proc. Int. Conf. on Electrical, Electronics, and Optimization Techniques (ICEEOT), pp. 762–766, 2016.
3. A. Banerjee, R. Singh, and S. Ghosh, "A Novel Approach of Arduino UNO Based Morse Code Generation," International Journal of Engineering Research & Technology (IJERT), vol. 10, no. 5, pp. 112–116, 2021.
4. S. S. Tevaramani, P. R. Naik, and R. Kulkarni, "Morse Code Decoder Using Arduino," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), vol. 13, no. 4, pp. 45–49, 2024.
5. A. Gueorguieva, G. Rakhmetulla, and A. S. Arif, "Enabling Input on Tiny/Headless Systems Using Morse Code," arXiv preprint, arXiv:2012.06708, 2020.
6. A. Bhatt, "Blink-to-Code: Real-Time Morse Code Communication via Eye Blink Detection and Classification," arXiv preprint, arXiv:2508.09344, 2025.
7. O. Y. Hutajulu, M. D. Mendoza, and Y. Simamora, "Feasibility Study of Smart House Design with ESP32 and Telegram in a Simple House," in Proc. Int. Conf. on Emerging Applications and Innovations (EAI), pp. 112–118, 2022.