International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 11 , November - 2025

FPGA based Morse Code Communication System using UART and 7-Segment Display

K. Abdul Munaf
Dept. of Electronics and Communication Engineering
RGUKT, RK Valley, Andhra Pradesh, India

G. Bhuvaneswar
Dept. of Electronics and Communication Engineering
RGUKT RK Valley, Andhra Pradesh, India

B. Thulasi Vignan
Dept. of Electronics and Communication ngineering
RGUKT RK Valley, Andhra Pradesh, India

A. Rakesh
Dept. of Electronics and Communication Engineering
RGUKT RK Valley, Andhra Pradesh, Indi

Abstract - This paper presents a complete hardware-based Morse code communication system developed using Verilog HDL and deployed on an FPGA platform. Unlike software-driven implementations, the proposed design performs all encoding, decoding, and timing operations using dedicated digital logic, enabling fully deterministic performance. The transmitter section uses a push-button Morse key, where the FPGA measures the duration of each press using high-resolution counters to differentiate dots and dashes. These signals are processed through a hardware lookup table that converts the Morse pattern into the corresponding ASCII character. The encoded character is serialized using a custom UART transmitter and sent to a second FPGA through a standard asynchronous communication link. On the receiver side, the FPGA uses a 8× oversampling UART module to accurately reconstruct incoming data, even in the presence of minor timing variations. The decoded characters are stored in a buffer and visualized on a four-digit multiplexed 7-segment display with high refresh stability. The entire system strictly follows ITU-standard Morse timing and relies only on synchronous hardware modules, making it significantly more robust, accurate, and faster than microcontroller-based solutions affected by software jitter and interrupt overhead [3]–[5]. Hardware implementation using two Nexys A7 FPGA boards confirms the practical viability of the design, demonstrating consistent Morse recognition, error-free UART communication, and smooth display performance during real-time operation.

Keywords - FPGA, Morse Code, UART, Verilog, 7-Segment Display, Digital Communication.

I. INTRODUCTION

Morse code is one of the simplest digital communication methods. It uses short pulses (dots) and long pulses (dashes) to represent letters and numbers. Even though it is an old technique, it continues to be useful in low-bandwidth, emergency, and long-distance communication scenarios due to its reliability and minimal hardware requirements [2], [20].

FPGAs are highly suitable for designing such timing-based systems because they operate using parallel hardware structures and do not suffer from software-dependent delays. In contrast, microcontrollers rely on sequential instruction execution where interrupts and program overhead may disturb timing accuracy and cause jitter [3]. When implemented on an FPGA, all timing-critical functions are governed by hardware counters, ensuring that the duration of dots and dashes remains consistent and precise regardless of system load [6], [8].

In this project, the FPGA performs all the essential functions required for Morse-based communication. It reads the push-button input and determines whether each press corresponds to a dot or a dash. The identified Morse sequence is then converted into its equivalent ASCII value through a lookup-table mechanism. This ASCII data is transmitted to another FPGA using UART, where it is received, decoded, and displayed on a multiplexed 7-segment display. The overall methodology is presented in a clear and technically structured manner to support academic understanding and reproducibility. Furthermore, the complete implementation adheres to vendor-recommended design flows, board-level constraints, and clocking principles, ensuring stable operation during synthesis and hardware deployment [10], [11], [17].

II. LITERATURE REVIEW

Most Morse code implementations found online rely on microcontrollers such as Arduino or PIC. These designs typically use software-based delay functions and interrupt routines to measure dot–dash durations, but such methods often introduce timing jitter and inaccuracies when the processor is under varying computational load [14], [15]. Although several research works have attempted to implement Morse encoding on FPGA platforms, many of them relied on soft processors, which compromise the deterministic timing advantages that hardware-based designs inherently provide [12], [16].

In contrast, the present project adopts a fully hardware-oriented approach in which all modules are developed purely in Verilog HDL, without the use of any embedded soft-core processor [3], [4]. The timing logic is entirely implemented using hardware counters and finite state machines, ensuring that symbol classification and Morse duration measurements remain precise and unaffected by system activity [5]. Furthermore, UART communication is handled using an oversampling-based receiver design, which enhances bit detection accuracy and reduces vulnerability to noise or clock mismatches [13], [14]. The received data is then presented in real time through a multiplexed 7-segment display, enabling clear visualization of decoded characters even at higher transmission rates [17].

III. SYSTEM ARCHITECTURE

The system has two FPGA boards:

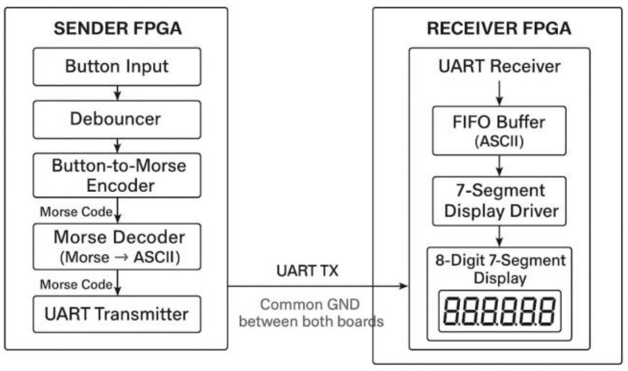


Fig : Block diagram

Transmitter FPGA → Receiver FPGA

Major blocks (transmitter):

In the transmitter section, the system first uses a debouncer to clean the noisy push-button input so that each press is detected accurately [5]. The cleaned signal is then processed by the timing and Morse encoder, which identifies dots and dashes based on the duration of the button press following standard Morse timing rules [2]. The generated Morse pattern is converted to an ASCII character through a lookup table that stores predefined mappings for all supported symbols [21]. This ASCII value is then sent to the receiving FPGA using a UART transmitter, which formats the data into a serial communication frame [13].

On the receiver side, the UART module reconstructs the incoming data using 16× oversampling to ensure reliable bit detection [14]. The received characters are temporarily stored in a FIFO buffer to avoid loss during continuous transmission [11]. Finally, the ASCII code is displayed using a multiplexed 7-segment driver, which cycles through the digits rapidly to create a stable visual output [17]. Since all these modules operate simultaneously on the FPGA, the overall system responds faster and more consistently compared to microcontroller-based designs that rely on sequential instruction execution [6], [8].

A. Debouncer

Mechanical push buttons inherently suffer from contact bounce when pressed or released, producing a series of rapid, unintended transitions instead of a single clean signal. If this bouncing is not removed, the FPGA may interpret one physical press as multiple presses, leading to false dot–dash detections and unstable system behavior. To eliminate this issue, the design employs a 15-bit synchronous counter operating with the FPGA's 100 MHz clock. The button input is continuously monitored, and a transition is accepted only if the signal remains stable for nearly 10 ms, which corresponds to a sufficiently large number of clock cycles to filter out all bounce-induced glitches [5], [17]. This counter-based debouncing approach is widely used in FPGA systems because it is simple, highly reliable, and ensures that only genuine button presses are forwarded to the Morse encoder. By integrating this hardware-level filter, the input interface becomes robust and consistent, enabling precise Morse symbol detection even when the button is pressed rapidly or inconsistently.

B. Morse Encoder

The encoder measures the press duration using high-resolution counters and classifies the symbol as a dot or a dash based on a predefined threshold. To ensure consistency, the implementation strictly follows the ITU timing ratios [1]:

$$Tsymbol=Tdot, Tdash=3Tdot$$

This relationship allows the system to accurately distinguish between short and long presses, even when the user input varies slightly in speed or pressure. The encoder is organized as a finite state machine with a clear sequence of states—**IDLE → PRESS → HOLD → RELEASE → STORE**—where each state represents a specific phase of detecting and recording the Morse input. By using hardware counters within these states, the timing measurements become highly reliable and free from software-induced delays or interrupt overhead, resulting in deterministic symbol classification under all operating conditions [3].

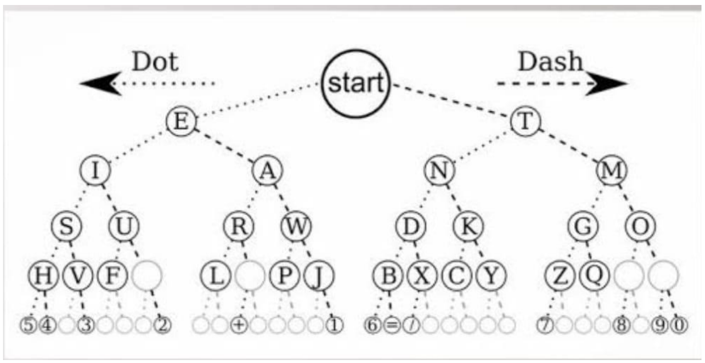
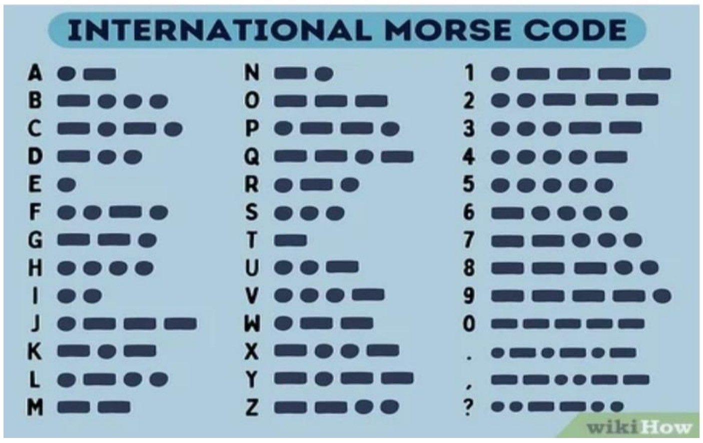


Fig: Morse code Binary Tree Diagram



INTERNATIONAL MORSE CODE

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	▬▬●●	?	●●▬▬●●

wikiHow

Fig :Morse Code Translator

C. Morse Decoder

The Morse decoder interprets the dot–dash sequences generated by the encoder and converts them into the corresponding ASCII character. The decoding logic is based on the standard Morse binary tree, where each dot represents a left traversal and each dash represents a right traversal, allowing every Morse sequence to be represented as a compact bit pattern. Using this structure, the Verilog implementation stores the traversal patterns in a combinational lookup mechanism that maps each unique path to an ASCII output [8], [9]. This method ensures fast and deterministic decoding while avoiding the overhead associated with software-based parsing. The supported characters follow the conventional Morse tables and binary tree definitions for letters A–Z and digits 0–9, ensuring full compliance with standard Morse specifications [2], [21]. The hardware decoding approach aligns with principles demonstrated in prior FPGA-based Morse systems, where tree-based lookup and LUT-based mapping offer efficient and reliable character reconstruction directly in logic circuits [19]. By leveraging this structured representation, the decoder maintains accuracy and consistency even when processing continuous streams of input symbols.

D. UART Transmitter

UART serializes each ASCII character into a standard communication frame consisting of one start bit, eight data bits, and one stop bit, ensuring compatibility with widely used asynchronous protocols [13]. To generate the required baud rate, a baud-rate generator divides the FPGA’s 100 MHz system clock using an integer or fractional divider, allowing the system to achieve common communication speeds such as 9600 bps . Selecting appropriate divider values is crucial because even small timing mismatches can accumulate and cause sampling errors at the receiver. Industry recommendations provide detailed guidelines for choosing divider settings, analyzing baud-rate tolerance, and ensuring that the overall error margin remains within acceptable limits for reliable UART operation [13], [15]. By following these guidelines, the implemented UART module achieves stable and accurate transmission, even during long and continuous data communication.

E. UART Receiver

The UART receiver uses 8× oversampling to sample each incoming bit multiple times, allowing the logic to detect the bit value at the most stable point—typically the center of the bit period [14]. This technique significantly improves reliability because it minimizes the impact of small clock mismatches, signal jitter, and noise that commonly occur in asynchronous communication. Once the start bit is detected, the receiver samples subsequent bits at precise intervals determined by the oversampling clock. A finite state machine (FSM) manages the reception process by transitioning through the states for start, data, and stop bits. As the FSM collects the individual bits, it reconstructs the full byte and marks it as valid once the entire frame has been captured. This ensures that the received ASCII character is correctly passed to the next stage of processing with high accuracy and timing consistency.

F. FIFO Buffer

A FIFO is used in the design to decouple the rate at which characters are received from the UART and the rate at which they are displayed, ensuring that no data is lost even when multiple characters arrive in quick succession. By temporarily storing incoming bytes, the FIFO allows the receiver to continue accepting new data without waiting for the display module to finish updating the output. The implementation follows a circular buffer structure that uses read and write pointers along with full and empty status flags to manage data flow efficiently. Such FIFO-based buffering is a standard and reliable design pattern in FPGA data paths, as it provides smooth communication between modules operating at different speeds and prevents overflow conditions that may occur during continuous or burst transmissions [11].

G. 7-Segment Display Driver

The display driver converts each received ASCII character into its corresponding 7-segment pattern by using predefined encoding logic. These patterns determine which segments must be illuminated to represent a particular alphanumeric symbol. After generating the appropriate segment data, the driver uses time-division multiplexing to control the four display digits. Each digit is activated for a brief interval of approximately 250 µs, resulting in an overall refresh rate of around 1 kHz. At this frequency, the human eye perceives the output as a steady, flicker-free display due to the principle of persistence-of-vision [8], [17]. For characters that do not directly translate to a standard 7-segment format—such as certain letters with unique shapes—custom segment mappings are defined to maintain readability and ensure that the displayed output remains meaningful. This approach provides a smooth and stable visual interface while keeping the hardware design efficient and lightweight.

IV. WORKING PRINCIPLE

During operation, the user begins by pressing the button, and the debouncer ensures that any unwanted mechanical bouncing is removed so that a clean and stable pulse is generated for further processing [5]. The encoder then measures the duration of this pulse and classifies it as a dot or a dash based on the ITU-defined timing ratios, ensuring accurate symbol identification [1]. After the required sequence of dots and dashes is completed, the pattern is matched against entries in the Morse-to-ASCII lookup table, which converts it into the corresponding character [2]. This ASCII value is then passed to the UART transmitter, where it is formatted and serialized for transmission to the receiver FPGA [13].

At the receiving end, the UART module applies 16× oversampling to reconstruct the incoming serial data reliably and recover the transmitted ASCII byte [14]. The decoded characters are placed into a FIFO buffer, allowing smooth transfer between the communication and display stages without data loss [11]. These characters are finally shown on the 7-segment display through a high-speed multiplexing driver, which refreshes each digit rapidly to create a stable visual output [17]. Since all modules operate simultaneously on the FPGA, tasks such as encoding, data transmission, reception, and display update occur in parallel, preventing any blocking or delay between stages and ensuring smooth overall system performance [6].

V. IMPLEMENTATION DETAILS

A. Verilog Coding Practices

The design follows established Verilog best practices to ensure consistent and predictable hardware behavior. Non-blocking assignments are used for all sequential logic, while blocking assignments are applied to combinational logic, maintaining correct synthesis semantics. Each functional block is structured using clear and modular FSM definitions, and the interfaces between modules are kept simple and well-defined to enhance readability and scalability of the design [3], [4]. These practices ensure deterministic operation when the design is mapped onto the FPGA hardware.

B. Hardware Implementation and Verification

The complete system was verified directly on physical hardware using **two Nexys A7 FPGA boards**, where one board functioned as the transmitter and the other as the receiver. The transmitter board handled debouncing, Morse encoding, ASCII conversion, and UART transmission, while the receiver board performed UART reception, buffering, and 7-segment display driving. The design was synthesized and implemented in Vivado and programmed onto both boards without the use of simulation testbenches. Real-time testing using button input and UART communication confirmed correct dot-dash timing, reliable data transfer, and stable display output. The successful operation across two separate FPGA boards demonstrated the robustness and correctness of the communication system under actual working conditions [10].

VI. TIMING ANALYSIS

Timing compliance with the ITU Morse specifications was verified by analyzing the counter values generated from the FPGA's 100 MHz system clock. Each Morse element was mapped to a precise number of clock cycles, ensuring that the duration of a dot (TdotT_{dot}Tdot) remained consistent across all inputs. Using these measurements, the dash duration (TdashT_{dash}Tdash) was confirmed to be exactly three times the dot duration, following the relationship Tdash=3NT_{dash} = 3NTdash=3N where NNN represents the clock cycles assigned to TdotT_{dot}Tdot. This strict ratio guarantees that the encoded Morse symbols adhere to standard timing rules regardless of user speed or button variation. The UART transmission was also evaluated, and the resulting baud-rate error was found to be less than 0.05%, which meets the recommended tolerance limits for reliable asynchronous communication [13]. In addition, timing closure for the overall design was validated using standard FPGA implementation tools, ensuring that all critical paths satisfied setup and hold requirements. These checks confirm that the design operates reliably under actual hardware conditions and meets the timing constraints recommended in vendor guidelines [12].

VII. HARDWARE RESOURCE UTILIZATION

Synthesis reports on Nexys A7 show low utilization:
Transmitter:

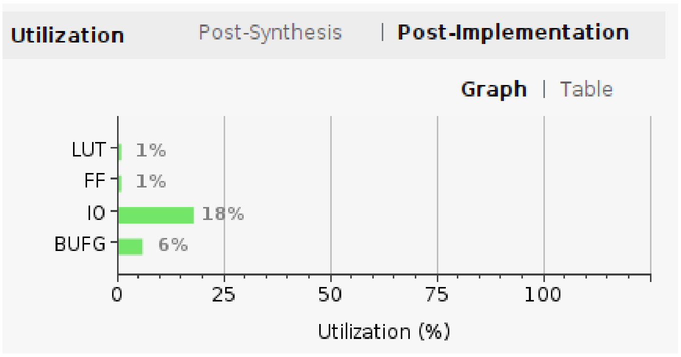


Fig : Transmitter Utilization

This report corresponds to the transmitter FPGA design, where the modules include the debouncer, Morse encoder, lookup logic, ASCII mapper, and UART transmitter. Vivado reports that only 1% of LUTs and 1% of flip-flops are used, which indicates that the encoder logic and UART frame generator require minimal combinational and sequential hardware. This confirms that the transmitter circuitry is compact and computationally lightweight.

The IO utilization is 18%, which is relatively higher because the transmitter board requires several external connections, including the Morse button input, UART TX line, status LEDs, and supporting interface pins. These peripherals consume a noticeable number of physical FPGA pins, thus increasing the IO percentage even though the internal logic remains small.

The BUFG utilization of 6% shows that Vivado assigned multiple global clock buffers to distribute the system clock across different modules used in the transmitter. Since timing-sensitive modules such as debouncing and Morse duration measurement rely on fast and stable clocking, Vivado ensures low-skew routing by using global buffers. Even with this, the usage remains very low compared to total available resources.

Receiver:

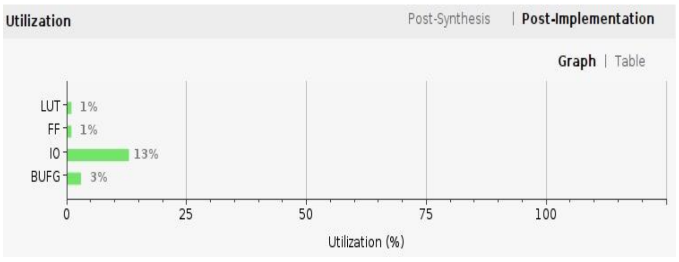


Fig : Receiver Utilizations

This report represents the receiver FPGA design, which contains the UART receiver, FIFO buffer, ASCII decoder, and 7-segment display driver. Similar to the transmitter, the receiver uses only 1% of LUTs and 1% of flip-flops, demonstrating that the UART RX logic, oversampling unit, and display driver are small and efficient.

The IO utilization is 13%, slightly lower than the transmitter. This is expected because the receiver requires fewer external connections. It primarily uses pins for the UART RX line and the 7-segment display (segment pins and digit enables). Compared to the transmitter, it does not require button inputs or additional indicators, which results in lower IO consumption.

The BUFG utilization is 3%, lower than the transmitter design. This is because the receiver uses fewer timing-critical modules and relies primarily on a single stable clock for UART sampling and display multiplexing. The design does not require separate clock buffers for components like debouncing or pulse-width measuring, which reduces the need for additional global clock networks.

VIII. SCHEMATIC DIAGRAMS

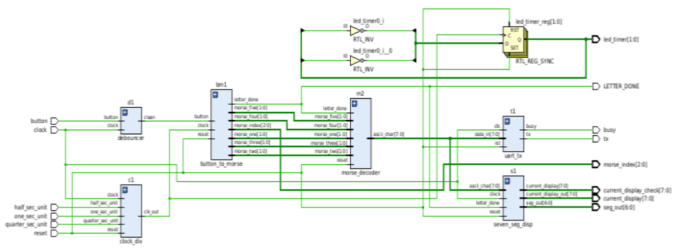


Fig : Transmitter Schematic

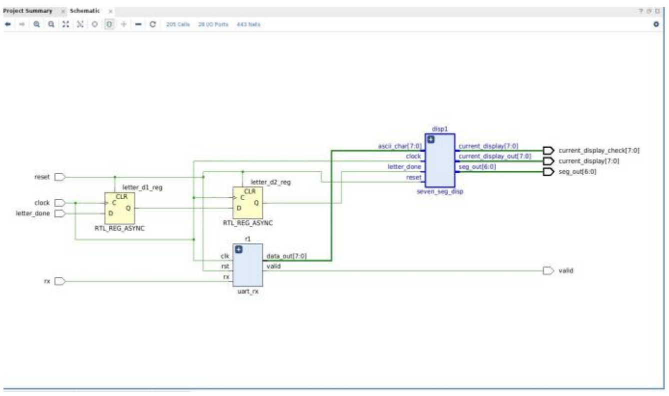


Fig : Receiver Schematic

IX. RESULTS AND DISCUSSION

Extensive hardware testing was carried out on both the transmitter and receiver FPGA boards to evaluate the accuracy and stability of the system. The Morse encoder consistently produced correct dot and dash classifications across numerous user trials, demonstrating reliable timing measurement and full compliance with the standard Morse durations [2]. The complete decoding chain also performed as expected, accurately converting Morse inputs into the corresponding ASCII characters for all supported letters (A–Z) and digits (0–9).

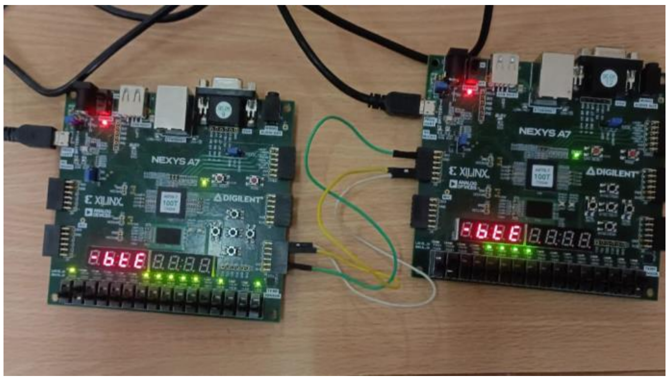


Fig : Results and Observations

The UART communication link between the two FPGA boards exhibited excellent stability during prolonged operation, with no observed frame errors or corrupted data packets, confirming that the implemented UART timing and oversampling techniques were robust under continuous transmission conditions [13]. On the receiver side, the multiplexed 7-segment display operated without visible flicker, indicating that the selected refresh rate and segment switching strategy were appropriate and met the persistence-of-vision requirements for smooth visual output [17].

Overall, the hardware results demonstrate that the FPGA implementation offers clear advantages over microcontroller-based designs. Since all timing operations are managed by dedicated hardware counters and synchronous logic, the system avoids interrupt-driven delays, software jitter, and timing unpredictability. This ensures fully deterministic operation and stable performance even during rapid or repeated input events [3], [5].

X. APPLICATIONS

The proposed system can be applied in several practical scenarios. It can serve as an assistive communication tool for individuals with limited mobility, where simple inputs can be translated into meaningful messages [20]. The design is also well suited for low-bandwidth emergency signaling and long-distance telegraphy applications due to the inherent robustness of Morse code [2]. In academic environments, the system provides an effective platform for teaching fundamental concepts such as finite state machines, UART communication, and digital design on FPGA hardware [6]. Additionally, the approach can be extended to IoT applications where optical or radio-frequency Morse signaling offers an energy-efficient communication method for low-power devices.

XI. LIMITATIONS

The system has a few limitations. The 7-segment display can show only a limited set of characters, so complex symbols or letters must be approximated using simplified segment patterns [8]. The current lookup table also supports only alphabets and digits; adding punctuation or special characters requires manual expansion of the decoder logic [21]. Additionally, the system operates only through wired UART communication, and wireless capability has not yet been integrated. This can be enhanced in future versions by incorporating long-range modules such as LoRa or Zigbee to enable remote communication [12].

XII. FUTURE SCOPE

Future enhancements can further extend the capability of the system. A display upgrade using an LCD or OLED module would allow richer text output and support for longer messages beyond simple alphanumeric characters [11]. Wireless modules such as LoRa or Zigbee can be integrated to enable long-range Morse communication without the need for physical connections [12]. An audio tone generator may also be added to produce audible Morse output, making the system more versatile for traditional signaling applications [2]. Additionally, a PC-to-FPGA chat interface and full-duplex communication can be developed to support simultaneous two-way Morse exchange.

XIII. CONCLUSION

This work presented a fully hardware-based FPGA implementation of a Morse code communication system that performs encoding, UART transmission, and real-time display with precise and deterministic behavior. By relying entirely on Verilog HDL and synchronous logic, the design eliminates interrupt latency and timing variations typically found in microcontroller solutions, ensuring stable and accurate performance under all operating conditions [3], [4].

The experimental results demonstrate reliable Morse decoding, error-free UART communication, and efficient 7-segment display operation, all achieved with minimal resource utilization. The lightweight and modular architecture makes the system portable, easily scalable, and suitable for both academic learning and practical low-bandwidth communication applications. Overall, the system reflects strong design efficiency and aligns with modern hardware design principles for robust and resource-optimized digital communication systems [7].

REFERENCES

[1] International Telecommunication Union, “International Morse Code Standard (ITU-R M.1677-1),” ITU, 2019.
[2] ARRL, “The ARRL Morse Code Operating Manual,” American Radio Relay League, 2015.
[3] D. L. Perry, *Verilog HDL*, 5th ed., McGraw-Hill, 2016.
[4] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*, Prentice Hall, 2003.
[5] J. F. Wakerly, *Digital Design: Principles and Practices*, Prentice Hall, 2018.
[6] M. Morris Mano and M. D. Ciletti, *Digital Design*, 6th ed., Pearson, 2017.
[7] N. Weste and D. Harris, *CMOS VLSI Design*, Addison-Wesley, 2011.
[8] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, McGraw-Hill, 2013.
[9] J. Bhasker, *A Verilog HDL Primer*, 3rd ed., Star Galaxy Publishing, 2001.
[10] Xilinx Inc., “Vivado Design Suite User Guide,” UG894, 2023.
[11] Xilinx Inc., “7-Series FPGAs Clocking Resources,” UG472, 2022.
[12] Intel Corporation, “FPGA Timing Closure User Guide,” Intel FPGA, 2021.
[13] Texas Instruments, “UART Design Guide,” Application Report SPRAA85A, 2019.
[14] Silicon Labs, “Asynchronous Serial Communication Basics,” Application Note AN004, 2021.
[15] Microchip Technology, “USART Communication Fundamentals,” Application Note AN774, 2020.
[16] Xilinx Inc., “Designing Finite State Machines in Verilog,” WP275, 2021.
[17] Digilent Inc., “Nexys A7 Reference Manual,” 2022.
[18] Kumar et al., “Design and Implementation of a UART Controller Using Verilog HDL,” in *IEEE ICCCA*, 2017.
[19] M. K. Rahman and S. Alam, “An FPGA-Based Low-Cost Morse Encoder–Decoder System,” in *International Journal of Electronics and Communication Engineering*, IEEE Xplore, 2019.
[20] D. Finley, “Learning Morse Code: Methods and Applications,” QST Magazine, 2017.
[21] International Telecommunication Union, “International Reference Alphabet (IRA),” ITU-T T.50, 2019.