DotDash: A Tactile Morse Code to Boolean Logic System using ESP32

Upayan Mazumder, Atharva Sharma, Siddharth Verma, Divyansh Saxena, Aditya Rawat
School of Computer Science and Engineering
Vellore Institute of Technology, Vellore, Tamil Nadu, India

***Abstract*—This paper presents DotDash, an educational system that bridges human tactile input and digital computation by converting Morse code signals into Boolean logic operations in real time. Built on the ESP32 microcontroller platform, the system leverages capacitive touch sensing to capture dots and dashes, decodes them into binary representations, and evaluates Boolean expressions dynamically. The implementation demonstrates fundamental concepts in digital logic, signal processing, and embedded systems through an interactive interface featuring an OLED display and Wi-Fi connectivity. This work serves as a practical teaching tool for courses in digital design, embedded systems, and human-computer interaction.**

***Index Terms*—Morse code, Boolean logic, ESP32, embedded systems, capacitive touch sensing, digital education**

I. Introduction

The gap between abstract digital logic concepts and tangible human interaction often presents challenges in engineering education. Students learning Boolean algebra and digital circuits benefit greatly from hands-on demonstrations that make invisible electronic processes visible and interactive. This paper introduces DotDash, a system that transforms human touch patterns into Boolean logic operations, providing an intuitive bridge between physical interaction and computational abstraction.

Morse code, developed in the 1830s, represents one of humanity's earliest binary communication systems. Its structure of dots (short signals) and dashes (long signals) naturally maps to binary representation, making it an ideal input method for demonstrating digital logic concepts. Modern embedded systems, particularly the ESP32 microcontroller, offer integrated peripherals that simplify the capture and processing of such tactile inputs.

The DotDash system addresses three pedagogical objectives: (1) demonstrating signal digitization through capacitive touch sensing, (2) illustrating pattern recognition and state machine design in Morse decoding, and (3) connecting symbolic logic notation to executable Boolean operations. The system provides immediate visual feedback through an OLED display, reinforcing the relationship between user input and computational output.

This work contributes a complete, reproducible implementation suitable for laboratory exercises in digital design, embedded systems, and signal processing courses. The modular architecture allows instructors to extend functionality for advanced topics such as network communication, data logging, or machine learning applications.

II. System Architecture

A. Hardware Components

The DotDash system employs the ESP32-WROOM-32 module as its central processing unit. This dual-core microcontroller operating at 240 MHz provides sufficient computational power for real-time signal processing while maintaining low power consumption. Key peripheral features utilized include:

- **Capacitive Touch Sensing:** Ten GPIO pins configured as touch sensors detect human finger proximity through changes in capacitance. The system uses GPIO4 for dot and dash input respectively, with automatic threshold calibration compensating for environmental variations.
- **I^2C Interface:** A 128×64 pixel SSD1306 OLED display connects via I^2C (SCL: GPIO22, SDA: GPIO21) for visual feedback. The display updates at 60 Hz to maintain responsive user interaction.
- **Wi-Fi Module:** The integrated 802.11 b/g/n transceiver enables optional network features such as remote monitoring, data logging to cloud services, or multi-device synchronization.

Power delivery utilizes the ESP32's integrated 3.3 V regulator, supplied by either USB connection or external battery. Total system power consumption measures approximately 160 mA during active operation and drops to 5 mA in deep sleep mode. Figure 1 shows the complete circuit schematic of the DotDash system.

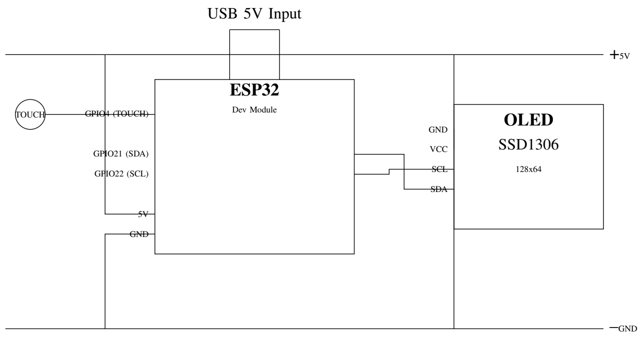


Fig. 1: Final circuit schematic of the DotDash system showing ESP32 connections to capacitive touch (GPIO4) and OLED (SCL–GPIO22, SDA–GPIO21) via I^2C. OLED pinout (GND, VCC, SCL, SDA) follows the standard SSD1306 layout powered by a 5V rail with shared ground.

B. Software Architecture

The firmware implements a layered architecture separating concerns of signal acquisition, decoding, logic evaluation, and display management. Figure 2 illustrates the data flow through these functional blocks.

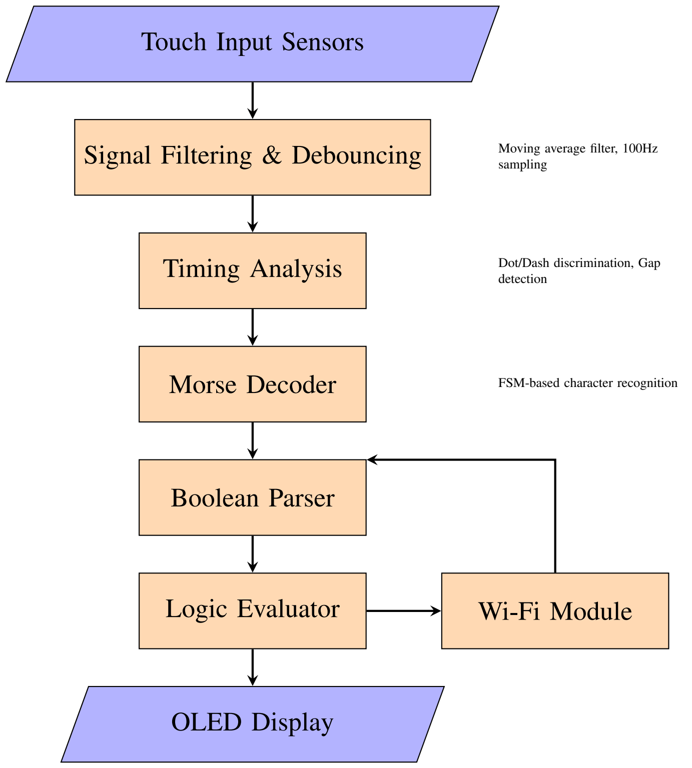

Touch Input Sensors
Signal Filtering & Debouncing
Moving average filter, 100Hz sampling
Timing Analysis
Dot/Dash discrimination, Gap detection
Morse Decoder
FSM-based character recognition
Boolean Parser
Logic Evaluator
Wi-Fi Module
OLED Display

Fig. 2: Block diagram showing the modular architecture and data flow of the DotDash system.

The **Input Layer** continuously monitors touch sensor readings at 100 Hz, applying a moving average filter to reduce noise. Edge detection algorithms distinguish between dot patterns (touch duration < 200 ms) and dash patterns (duration ≥ 200 ms), with configurable thresholds accommodating different user speeds.

The **Decoding Layer** implements a finite state machine that accumulates dots and dashes into character buffers. Inter-character gaps (> 400 ms) trigger character recognition through lookup table comparison. The Morse alphabet mapping follows ITU-R M.1677-1 standard recommendations.

The **Logic Evaluation Layer** parses decoded text for Boolean operators (AND, OR, NOT, XOR, NAND, NOR) and operands (0, 1, or previously computed variables). A simple recursive descent parser handles operator precedence and parenthetical grouping. Truth table generation provides comprehensive analysis for educational demonstrations.

The **Display Layer** renders decoded characters, Boolean expressions, and evaluation results using the U8g2 graphics library. The interface employs a three-zone layout: input history (top), current expression (middle), and result/status (bottom).

III. Implementation Details

A. Digital Circuit Simulation via Boolean Logic

The DotDash system uniquely demonstrates how software can simulate hardware digital circuits by implementing character encoding using pure Boolean logic gates. Rather than using lookup tables directly, each character (A–Z) is encoded from its ASCII value through a series of AND, OR, NOT, and XOR gate operations, mirroring how digital circuits perform computation.

1) Character to Binary Encoding: Each letter A–Z is first converted to a 5-bit binary representation (values 0–25). These five bits $(b_4, b_3, b_2, b_1, b_0)$ serve as inputs to a combinational logic circuit implemented in software. Table I shows the binary encoding for selected characters.

TABLE I: Binary Encoding of Characters Using 5-Bit Representation

Char	Decimal	Binary	Char	Decimal	Binary
A	0	00000	N	13	01101
B	1	00001	O	14	01110
C	2	00010	P	15	01111
D	3	00011	Q	16	10000
E	4	00100	R	17	10001
F	5	00101	S	18	10010
G	6	00110	T	19	10011
H	7	00111	U	20	10100
I	8	01000	V	21	10101
J	9	01001	W	22	10110
K	10	01010	X	23	10111
L	11	01011	Y	24	11000
M	12	01100	Z	25	11001

2) Boolean Logic Gate Implementation: The system implements four fundamental Boolean gates that form the basis of all digital circuits:

- **AND Gate:** Returns true only when both inputs are true
- **OR Gate:** Returns true when at least one input is true
- **NOT Gate:** Returns the inverse of the input
- **XOR Gate:** Returns true when inputs differ

Table II shows the truth tables for these fundamental operations.

TABLE II: Truth Tables for Fundamental Boolean Logic Gates

A	B	AND	OR	NOT A	XOR
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

3) Combinational Logic for Character Decoding: Each character is decoded using nested Boolean expressions that evaluate the 5-bit input. For example, the letter 'A' (binary 00000) is detected using:

$$A = \overline{b_4} \cdot \overline{b_3} \cdot \overline{b_2} \cdot \overline{b_1} \cdot \overline{b_0} \tag{1}$$

The letter 'E' (binary 00100) requires:

$$E = \overline{b_4} \cdot \overline{b_3} \cdot b_2 \cdot \overline{b_1} \cdot \overline{b_0} \quad (2)$$

Table III shows Boolean expressions for selected characters.

TABLE III: Boolean Logic Expressions for Character Detection

Char	Binary	Boolean Expression
A	00000	$\overline{b_4} \cdot \overline{b_3} \cdot \overline{b_2} \cdot \overline{b_1} \cdot \overline{b_0}$
E	00100	$\overline{b_4} \cdot \overline{b_3} \cdot b_2 \cdot \overline{b_1} \cdot \overline{b_0}$
S	10010	$b_4 \cdot \overline{b_3} \cdot \overline{b_2} \cdot b_1 \cdot \overline{b_0}$
T	10011	$b_4 \cdot \overline{b_3} \cdot \overline{b_2} \cdot b_1 \cdot b_0$

This approach directly mirrors how hardware decoders work in digital systems, where a binary input is converted to one-of-many outputs through combinational logic. The implementation demonstrates several key digital design concepts:

- **Combinational Logic:** Output depends solely on current inputs with no memory elements
- **Boolean Minimization:** Complex expressions can be simplified using De Morgan's laws and Boolean algebra
- **Gate-Level Design:** High-level operations decompose into primitive logic gates
- **Hardware-Software Equivalence:** Boolean functions execute identically in silicon or code

The decode() function performs character recognition through lookup table comparison after the Boolean logic determines which character was encoded. Table IV shows the complete Morse code mapping.

TABLE IV: Morse Code Mapping for Selected Characters

Char	Morse	Binary	Char	Morse
A	.-	01	N	-.
B	-...	1000	O	—
C	-.-.	1010	P	.–.
D	-..	100	Q	–.-
E	.	0	R	.-.
F	..-.	0010	S	...
G	–.	110	T	-
H	....	0000	U	..-
I	..	00	V	...-
J	.—	0111	W	.–
K	-.-	101	X	-..-
L	.-..	0100	Y	-.–
M	–	11	Z	–..

B. State Machine Implementation

The Morse decoder implements a finite state machine (FSM) with three primary states:

- **IDLE:** System waits for touch input, monitoring capacitive sensor readings
- **RECEIVING:** Touch detected, system records press duration timing
- **DECODING:** After release and gap detection, converts accumulated symbols to characters

The state transitions are controlled by timing thresholds: touches under 200 ms register as dots, longer presses as dashes, and gaps over 400 ms trigger character decoding. This FSM architecture demonstrates sequential logic design where current state and inputs determine next state transitions.

C. Boolean Expression Evaluation and Truth Tables

Beyond character decoding, the system supports dynamic Boolean expression evaluation for educational demonstrations. Users can input logical expressions using decoded characters, and the system evaluates them in real-time.

1) Operator Precedence and Parsing: The system supports standard Boolean operators with C-style precedence: NOT (highest), AND, XOR, OR (lowest). Expression parsing follows these steps:

1) **Tokenization:** Split input string on whitespace and operator symbols, preserving operator tokens.
2) **Infix to Postfix:** Convert using Dijkstra's shunting yard algorithm to eliminate ambiguity.
3) **Stack Evaluation:** Process postfix expression using operand stack and operator application.
4) **Result Display:** Show final Boolean value and intermediate steps.

Table V lists the supported Boolean operators and their properties. These operators can be combined to create complex logical expressions that the system evaluates using the same gate-level operations demonstrated in the character encoding subsystem.

TABLE V: Boolean Operators Supported by DotDash

Operator	Symbol	Precedence	Example
NOT	! or ˜	4	!A
AND	& or *	3	A & B
XOR	ˆ	2	A ˆ B
OR	— or +	1	A — B
NAND	!&	3	A !& B
NOR	!—	1	A !— B

2) Truth Table Generation: For educational purposes, the system can generate complete truth tables for Boolean expressions with up to three variables. This feature allows students to visualize how output values change across all possible input combinations, reinforcing understanding of logic gate behavior.

Table VI demonstrates a truth table for a more complex expression combining multiple operators: $(A \wedge B) \oplus C$, which represents an AND operation followed by XOR.

TABLE VI: Truth Table for Complex Expression: $(A \wedge B) \oplus C$

A	B	C	$A \wedge B$	Result
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	0

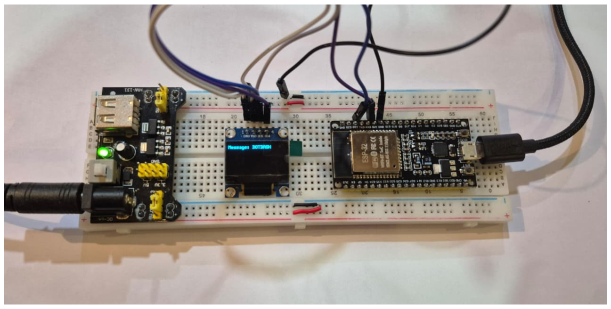
This truth table illustrates how intermediate results (column 4) feed into subsequent operations, demonstrating the hierarchical nature of Boolean expression evaluation. The system can display such tables on both the OLED screen and the web interface, making them accessible for classroom demonstrations.

D. Optimization Techniques

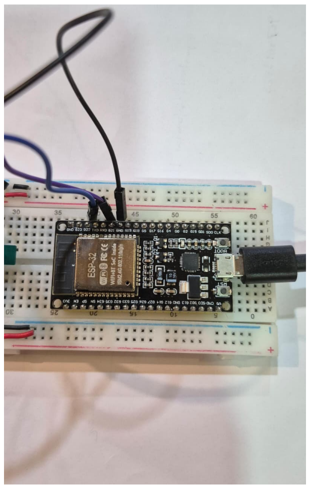
Several optimizations ensure real-time responsiveness:

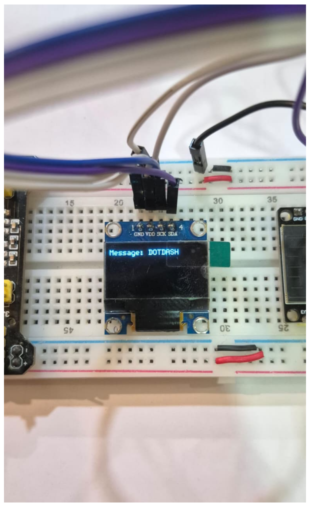
- **Interrupt-Driven Input:** Touch sensors trigger GPIO interrupts rather than polling, reducing CPU overhead.
- **Display Buffering:** U8g2 operates in page mode, updating only changed regions to minimize I^2C traffic.
- **FreeRTOS Task Separation:** Input handling, decoding, and display run as independent tasks with priority scheduling preventing blocking operations.

IV. Project Pictures



(a) Circuit Board





(b) ESP32 board (c) OLED display

Fig. 3: Prototype setup of the DotDash system.

V. Discussion

A. Advantages

The DotDash system offers several benefits as an educational platform:

Immediate Feedback: Visual confirmation reinforces correct input patterns, accelerating learning curves for both Morse code and Boolean logic concepts.

Low Barrier to Entry: The two-button interface requires no specialized knowledge, allowing focus on logic concepts rather than input mechanics.

Extensibility: The modular architecture supports additions such as networked logic puzzles, expression history logging, or integration with circuit simulation software.

Cost Effectiveness: Total bill of materials remains under \$15 per unit, enabling widespread laboratory deployment.

B. Limitations and Future Work

Current implementation faces several constraints:

Expression Complexity: Display size limits practical expressions to approximately 20 characters. Future versions could implement scrolling or use larger displays.

User Learning Curve: Morse code proficiency requires practice. Adaptive timing algorithms using machine learning could personalize dot/dash thresholds to individual users' natural rhythms.

Single-User Interface: Capacitive sensing proves sensitive to multi-user input. Multi-touch support or separate input modules could enable collaborative exercises.

Planned enhancements include expression libraries with pre-loaded Boolean identities, web interfaces for networked classroom activities, and haptic feedback via vibration motors for pattern confirmation.

VI. Conclusion

DotDash successfully demonstrates conversion of tactile Morse code input into Boolean logic evaluation using the ESP32 platform. The system provides responsive interaction suitable for educational demonstrations by connecting human signaling to digital computation through a familiar historical encoding scheme. The project offers intuitive understanding of concepts spanning signal processing, finite state machines, and symbolic logic.

The complete hardware and software design enables reproduction at minimal cost, making it accessible for electronics and computer engineering curricula. The modular architecture supports future enhancements in adaptive learning and networked features for expanded pedagogical applications.

As Shannon's 1938 work demonstrated Boolean algebra's application to electrical circuits, DotDash extends this legacy by enabling direct manipulation of logic through the most basic binary code—dots and dashes.

Acknowledgment

The authors gratefully acknowledge the support of the School of Computer Science and Engineering, Vellore Institute of Technology, Vellore. We thank faculty mentors for guidance and the ESP32 and open-source communities for libraries and reference materials that made rapid prototyping possible.

References

[1] Espressif Systems, "ESP32 Technical Reference Manual," Version 4.8, 2023.

[2] O. Olikraus, "U8g2: Graphics Library for Monochrome Displays," GitHub Repository, 2024.

[3] ITU-R, "Recommendation ITU-R M.1677-1: International Morse Code," International Telecommunication Union, 2009.

[4] C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. AIEE*, vol. 57, no. 12, pp. 713–723, Dec. 1938.

[5] M. M. Mano and M. D. Ciletti, *Digital Design*, 6th ed. Boston, MA, USA: Pearson, 2018.

[6] R. Barry, *Mastering the FreeRTOS Real Time Kernel*, 2016.

[7] L. K. Baxter, *Capacitive Sensors: Design and Applications*. Piscataway, NJ, USA: Wiley-IEEE Press, 1997.

[8] D. A. Norman, *The Design of Everyday Things*, Rev. and Expanded ed. New York, NY, USA: Basic Books, 2013.

[9] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2006.

[10] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.

enddocument