Blocks o' Code (v3): A Modular Tangible Programming System for Interactive Learning

Jordan Merkel, Destiny Ellenwood, Annesley Kolb, Camilla Torres

Department of Electrical and Computer Engineering
University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — Blocks o' Code (v3) is a modular hands-on programming platform designed to introduce children to programming concepts through interactive physical blocks. The system uses ESP32-based processing, inter-block communication, display feedback, and audio output to create an engaging educational environment. Each block acts as part of a larger embedded system, allowing users to build logical sequences through physical interaction instead of relying only on screen-based coding. The project emphasizes modularity, ease of use, and classroom adaptability while addressing hardware integration challenges such as communication reliability, power distribution, and mechanical connectivity. Quantitative validation confirms strong system performance: LED selection latency averaged 13.1 ms against a 50 ms target, end-to-end communication latency averaged 49.6 ms against a 2000 ms target, and I^2C rise times averaged 289 ns, well within the 1000 ns standard-mode limit.**

***Index Terms* — Educational technology, embedded systems, ESP32, I^2C communication, modular electronics, tangible programming.**

I. Introduction

Teaching programming concepts to young learners can be challenging when traditional tools rely heavily on abstract syntax, text-based coding, and screen-based interaction. These approaches often create a barrier for beginners who benefit from more hands-on and interactive learning environments. As a result, many educational tools have been developed to introduce computational thinking through physical interaction and visual feedback [3], [4].

The Blocks o' Code (v3) system addresses this challenge by providing a modular, hands-on programming platform designed for classroom environments. The system allows users to construct simple programs by physically connecting electronic blocks, where each block represents a specific programming function or instruction. When connected together, the blocks communicate through embedded microcontrollers and shared communication protocols to execute programmed behavior such as showing visual feedback, generating sound, or controlling lighting effects.

The hardware architecture is built around the ESP32-WROOM-32 module, which provides wireless connectivity, processing capability, and support for multiple peripheral interfaces. Each block contains its own electronics and connects to neighboring blocks through pogo-pin connectors, allowing rapid reconfiguration without cables or complex setup. Integrated peripherals, such as displays, speakers, and LEDs, provide immediate feedback, reinforcing the relationship between program structure and system output. This paper presents the design and implementation of the Blocks o' Code (v3) platform, including system architecture, hardware design, communication methods, and peripheral subsystems. Experimental testing and system validation are also discussed to evaluate performance and reliability.

The goal of this project is to design a robust, portable, and educationally effective system that enables students to learn fundamental programming concepts through physical interaction. By combining embedded systems design, modular hardware architecture, and interactive feedback mechanisms, Blocks o' Code (v3) aims to bridge the gap between abstract programming logic and tangible learning experiences.

This paper details the following primary contributions to the Blocks o' Code platform:

- A highly modular hardware platform utilizing I^2C communication.
- A FreeRTOS-based firmware architecture for real-time block detection.
- A Flutter-based companion app with 3D visualization and syntax validation.
- Quantitative latency and electrical validation of the interconnected system.

A. Goals & Constraints

The project goals were organized into three levels. The basic goals were to build a modular set of physical programming blocks, support magnetic block-to-block connections carrying power and I^2C data, and allow a central Brain block to detect and interpret the connected configuration. The advanced goals focused on efficiency and scalability, including reducing per-block cost and limiting power consumption during active operation. The stretch direction aimed to expand the platform into a richer educational system through stronger multimedia interaction, improved app support, and future hardware enhancements. In the current prototype, the team achieved the core functional goals of the system: the Brain block can scan connected child blocks, communicate with the companion app, validate the detected sequence through the app

pipeline, and execute working LED and audio behaviors. Some of the longer-term goals, especially aggressive cost, power, and advanced hardware expansion targets, remain future work rather than fully validated outcomes in the present prototype.

Several high-level constraints shaped the design. Cost remained important because the system was intended for educational use and needed to stay practical for replication. The platform also had to be kid-friendly, which required blocks that were durable, easy to handle, and simple to connect correctly. In addition, latency was a major system constraint, since delayed feedback would weaken the connection between a user's action and the resulting system response. For that reason, the firmware and communication architecture also needed to support predictable, repeatable, and as much as practicable deterministic timing behavior in the local control path. Power limits further influenced the design because each block had to support embedded processing, communication, display, lighting, and audio within realistic portable hardware constraints. Therefore, the final design balanced responsiveness, deterministic timing needs, usability, safety, and portability while preserving the modular goals of the platform.

II. System Architecture

The Blocks o' Code (v3) platform is organized as a distributed embedded system centered around a Brain block that coordinates a network of child blocks and a companion Flutter application. At the lowest level, each child block exposes a uniform I^2C interface for configuration and execution, while internally managing its own peripherals such as a TFT Display, LED strip and matrix, and audio hardware. Each cube connects to its neighbors through a four-pin magnetic pogo connector carrying VCC, GND, SDA, and SCL. The SDA and SCL lines form the shared 100 kHz I^2C bus, while the GND pin provides a common reference for all digital communication. The VCC line is used as a low-current reference supply for the bus and detection circuitry, but the primary power for each block remains local to its own battery and regulation stages. This four-wire interface allows the system to share a stable logic and communication reference across the chain without relying on a high-current shared power rail, preserving both modularity and fault isolation.

From a system perspective, the Brain block acts as the primary controller on this bus. It periodically scans for active devices, discovers the physical order of the connected blocks, and builds a logical program representation from that configuration. This representation is serialized into a compact JSON structure that describes each block by type and position within the chain. The Brain then uses FreeRTOS tasks to separate three main responsibilities: continuous scanning and configuration management, local program execution, and network communication.

Above the hardware layer, the Brain maintains a persistent Wi-Fi connection to a dedicated TCP server running on the local network. The server exposes a simple message relay interface between the embedded system and the Flutter companion application. Whenever the physical configuration changes, the Brain transmits an updated JSON configuration through the server to the app, which validates the sequence and provides real-time visualization. Validation results, execution commands, and heartbeat messages flow back down the same path to the Brain. This architecture cleanly decouples physical block assembly, embedded control, and user-facing visualization while still achieving low end-to-end latency from user action to on-screen and physical feedback.

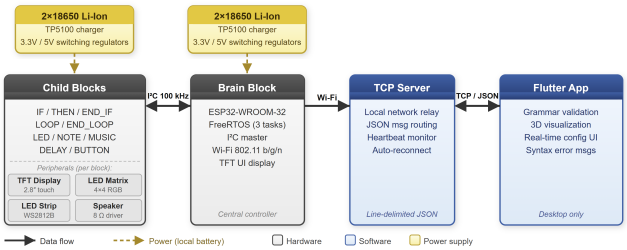

Fig. 1. System block diagram. Solid arrows indicate data flow (I^2C and Wi-Fi/TCP); dashed arrows indicate local power delivery from on-board batteries.

III. Hardware Design

A. ESP32 Processing Core

The ESP32-WROOM-32 module [1] is used as the primary controller in the Brain block and serves as the central processing unit for the system. It was selected because it provides sufficient processing capability, integrated Wi-Fi, and support for multiple communication interfaces required by the design.

The ESP32 manages communication with all connected child blocks over the I^2C bus. It continuously scans for active devices, maintains the current block configuration, and interprets the physical arrangement as a program sequence. Based on this configuration, the ESP32 controls system behavior by coordinating input handling, control flow, and output execution.

In addition to local control, the ESP32 handles communication with the companion application over Wi-Fi. Configuration data is transmitted to the app for validation, and execution commands are received to control system operation.

The ESP32 runs FreeRTOS [6] to separate scanning, execution, and communication tasks. This structure im-

proves overall system responsiveness and helps maintain consistent timing behavior during operation.

B. Child Block Hardware & I^2C Bus

Each child block uses a standardized hardware platform to ensure compatibility, simplify manufacturing, and maintain consistent electrical and mechanical interfaces across the system. Each block includes a 2.8-inch TFT touchscreen display, which serves as the primary user interface for visualizing program elements and interacting with block-specific functions.

The display allows students to visualize program elements and interact with block-specific features depending on the role of the block within the program sequence. For example, users may select actions, trigger functions, or view feedback directly through the touchscreen interface.

Additional interaction methods are integrated into each block to provide multiple forms of feedback and engagement. A 4×4 LED matrix can display simple animations, icons, or patterns that represent system outputs or program behavior, allowing students to quickly observe how their programmed sequence affects the system. A programmable LED strip provides immediate visual feedback regarding program execution: the strip illuminates red or green to indicate whether the current block configuration is correct or incorrect, helping guide students toward the proper logical order of operations. Once a valid configuration is detected, the LED strip also functions as an interactive output element, enabling students to select colors, brightness levels, and lighting patterns that correspond to the executed program.

Audio feedback is provided through the integrated amplifier and speaker subsystem, enabling the system to play sounds, musical notes, or simple songs as part of program execution. This introduces an additional sensory interaction method that can be triggered through specific block functions, allowing students to explore how different commands generate audio responses.

Inter-block communication is achieved using a 100 kHz I^2C bus [2] routed through magnetic pogo-pin connectors. These connectors enable both mechanical attachment and electrical connectivity. The bus utilizes pull-up resistors and unique device addressing to ensure reliable communication and prevent contention across multiple connected blocks.

When blocks are connected together, configuration data is transmitted through the I^2C network to the Brain block. The Brain processes the block sequence and communicates with the companion application to verify whether the connected blocks form a valid program structure. If the configuration is correct, an execution signal is returned to the system, allowing the blocks to perform their programmed actions. This architecture maintains modular flexibility while ensuring reliable communication and synchronized behavior across all connected blocks.

C. Power & Charging Architecture

Each block in the system (15 total) is designed with an identical, self-contained power architecture to ensure modularity and independent operation. Every block is powered by two 18650 lithium-ion batteries, providing a stable and rechargeable energy source.

Charging is handled locally on each block using the TP5100 charging IC configured for 2-cell (2S) lithium-ion charging [8]. USB-C input is used as the primary charging interface, paired with a PD trigger module to negotiate a 9 V input supply. This ensures compatibility with modern USB-C power sources while maintaining a simple and reliable charging profile.

Voltage regulation is performed onboard to supply the required 3.3 V and 5 V rail for the ESP32-WROOM microcontroller, display, and communication circuitry. Power is distributed locally within each block, meaning there is no shared power bus between blocks. This prevents cascading failures and improves system robustness.

Protection considerations include proper grounding, regulated voltage rails, and careful routing of power traces to minimize voltage drop and noise. Since each block is electrically independent, fault isolation is inherently achieved.

The custom PCBs, developed in KiCad [10], utilize slotted pads to ensure robust mechanical connections for the power and data lines, minimizing voltage drops across the modular chain.

D. Audio Subsystem Architecture

The audio subsystem of the Blocks o' Code (v3) platform provides auditory feedback to enhance the interactive learning experience. Audio output is generated via the ESP32's I^2S digital audio interface (Fig. 2), which provides superior signal quality compared to PWM or DAC-based methods. The audio subsystem supports interactive features: children can select preloaded songs or activate individual notes, experimenting with simple melodies while learning how block connections map to program behavior.

The I^2S signal is AC-coupled into the amplifier input to remove any DC offset and stabilize the input stage. An LM386 low-voltage amplifier was chosen for its low component count and single-supply operation [9]. A resistor-capacitor network between pins 1 and 5 enables bass boost, external passive components raise the voltage gain above the default of 20, and the amplified output drives an $8\,\Omega$ speaker. Users can trigger preloaded songs or individual notes through block connections, reinforcing the cause-and-effect relationship between physical programming and audible output.

This design provides a compact and efficient audio solution capable of producing clear tones and sound effects while maintaining low power consumption. By allowing users to trigger songs or individual musical notes through block interactions, the audio subsystem reinforces the cause-and-effect relationship between physical programming inputs and system outputs. The use of the LM386 bass boost configuration further improves perceived audio quality, ensuring that the sound output remains clearly audible in classroom environments while still maintaining a simple and reliable circuit design.

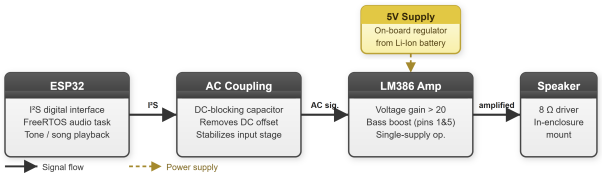

Fig. 2. Audio subsystem signal chain: ESP32 I^2S output $\rightarrow$ AC coupling $\rightarrow$ LM386 amplifier $\rightarrow$ 8 Ω speaker. Dashed arrow indicates amplifier power supply.

E. Implementation - Power Details

Key PCB layout decisions improved reliability and thermal performance. A decoupling capacitor in the TP5100 charging circuit was relocated closer to the IC pins to reduce noise. Stitching vias were added under the voltage regulator footprint to distribute heat through the PCB under load. Ninety-degree pin headers accommodated module placement within the cube enclosure dimensions. The PCB schematic, shown in Fig. 3, highlights the charging IC, the power delivery, and the I^2C pull-up resistors (4.7 kΩ) on SDA and SCL.

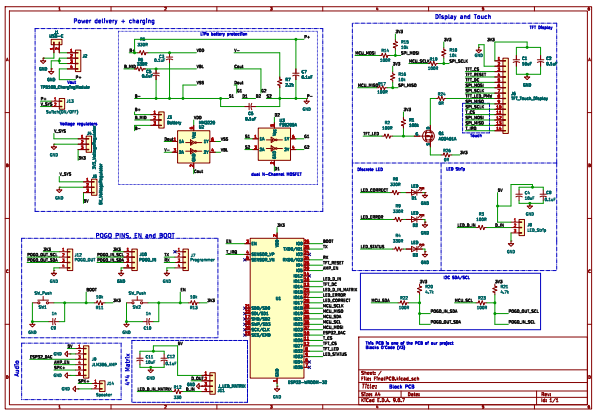

Fig. 3. PCB schematic for the Blocks o' Code (v3) block. Power enters through the USB-C connector (J1) and is conditioned by the TP5100 2S lithium-ion charging module (J2), dual N-channel MOSFET battery protection (U2, U3), and AO3401A high-side power switch (Q1), producing a regulated V_{SYS} rail. Dedicated 3.3 V (J4) and 5 V (J5) switching regulators supply the ESP32-WROOM-32 (U1) and peripheral subsystems respectively. The 4.7 kΩ I^2C pull-up resistors R20 and R21 tie SDA and SCL to 3.3 V and route to the POGO_IN (J10) and POGO_OUT (J12) inter-block pogo-pin connectors. Peripheral interfaces include the TFT touchscreen display (J6), WS2812B LED strip (J8), 4×4 LED matrix (J11), LM386 audio amplifier module (J9), and speaker (J14).

IV. Software & Firmware Design

A. Firmware Architecture

The Brain block firmware is organized around several FreeRTOS tasks that separate scanning, execution, and communication. First, the `block_cfg_scan` task continuously scans the shared I^2C bus for connected child blocks and updates the current physical configuration. Within that scan path, the configuration manager compares each new scan against the previous state, detects topology changes, and generates a JSON description of the block chain. The Brain also runs an executor task, which steps through the detected block sequence as a simple program using a program counter, delay states, and loop handling. In parallel, a Wi-Fi/TCP client task sends configuration updates to the companion app and receives validation messages, heartbeat traffic, and execution-related commands. This task separation improves timing predictability in the local firmware path because scanning, execution, and communication follow defined control paths instead of being combined in one large loop.

Child blocks use a shared register-and-command protocol so that the Brain can interact with each one in a consistent way. During scanning, the Brain reads `REG_WHOAMI` from each responding device to determine its logical type. During runtime, child blocks handle commands such as `CMD_SET_LED`, `CMD_GET_DATA`, `CMD_RESET`, and `CMD_EXECUTE`. This allows the Brain to use one common interface across different block types while still allowing each child block to manage its own local peripherals and behavior.

Invalid configurations are handled conservatively on the firmware side. When the physical topology changes, the Brain clears the previous validation state and requires the companion app to validate the new configuration before execution can begin. In addition, blocks that appear missing or return an unknown type are flagged during scanning and increase the configuration error count. If a configuration changes while execution is already in progress, the executor is stopped so the system does not continue running an outdated block arrangement. Therefore, the firmware helps maintain predictable and safe system behavior while the companion app performs the main structural validation of the block sequence.

B. Block Behavior Logic

The Brain executor interprets each block by logical role. Control-flow blocks (IF/END_IF, LOOP/END_LOOP, DELAY) act as structural instructions: conditionals gate execution on button input, loops repeat using a managed loop stack, and delay blocks suspend the executor for a timed interval. Input blocks (button block) provide runtime events that drive conditional branching. Output blocks (LED Color Flash, Note, Music Sequence) cause the Brain to issue I^2C configuration commands followed by CMD_EXECUTE, triggering local peripheral actions on the child block. The result is a complete programming loop: the Brain interprets structure, inputs provide decisions, and outputs deliver the system response.

C. Implementation – Key Functions

Listing 1 shows the Brain's FreeRTOS task creation. Three parallel tasks maintain timing independence between event polling, network communication, and program execution.

Listing 1. Brain block FreeRTOS task structure (app_main).

```
// I2C master init and initial child-block scan
ESP_ERROR_CHECK(i2c_master_init());
i2c_safe_scan();
tft_ui_start();          // LVGL display task

// Poll child blocks for DATA_READY events (priority 5)
xTaskCreatePinnedToCore(block_event_poll_task,
    "block_evt", 4096, NULL, 5, NULL, 0);

start_network_client(); // Wi-Fi/TCP client task

// Program execution engine (priority 3)
xTaskCreatePinnedToCore(brain_executor_task,
    "brain_exec", 6144, NULL, 3, NULL, 0);
```

Listing 2 shows the child block command dispatcher. Every child block exposes this uniform interface so the Brain can configure and trigger any block type through a common set of I^2C commands.

Listing 2. Child block I2C command handler (LED Color Flash block).

```
void handle_command(uint8_t *buf, int len) {
    switch (buf[0]) {
    case CMD_SET_LED:   // configure color/pattern
        color_id = buf[1] % 10;
        break;
    case CMD_EXECUTE: { // trigger LED animation
        led_action_t act = {ACTION_EXECUTE, color_id};
        xQueueSend(s_action_queue, &act, 0);
        break;
    }
    case CMD_RESET:
        color_id = 0;
        current_status = STATUS_READY;
        break;
    }
}
```

D. Flutter App Architecture

A companion Flutter application [7] provides the primary user interface for visualizing, validating, and monitoring the Blocks o' Code (v3) system. The app connects to a lightweight TCP server that relays messages between the Brain block and the mobile or desktop device. All communication is expressed in terms of a JSON BlockConfiguration model, which encodes the ordered list of detected blocks, their logical types, and associated metadata such as loop counts or delay parameters.

Internally, the app is organized into three main layers: a networking layer that manages the TCP socket and heartbeat behavior, an application logic layer that performs grammar validation and state management, and a presentation layer that renders the live block list and 3D visualization. The networking layer implements robust reconnect behavior and periodic heartbeats to detect dropped connections and automatically restore communication without user intervention. Incoming JSON messages are parsed into strongly typed Dart models, which are then exposed to the rest of the app through a centralized state container.

The grammar engine enforces four core syntactic rules, including proper pairing of IF/END_IF and LOOP/END_LOOP blocks, valid placement of THEN blocks, and the requirement that executable sequences begin with a Brain block and end with an output-capable block. When a configuration violates any of these rules, the app surfaces explicit error messages that highlight the offending block and describe the issue in child-friendly language. The app also provides both a textual list and a real-time 3D visualization of the physical assembly, allowing learners to see how their tangible program maps to a logical structure. Telemetry, such as connection status and execution state, is displayed alongside this view to help instructors and students monitor system health.

Listing 3. Flutter grammar validation entry point.

```
static List validateAll(BlockConfiguration config) {
 final violations = [];

 // Check Brain Block rule
 violations.addAll(checkBrainBlockRule(config));

 // Always validate sequences too, so users can see all
      placement issues
 // in a single pass (Brain Block plus If/Loop/ordering
      problems).
 violations.addAll(checkIfBlockSequences(config));
 violations.addAll(checkLoopBlockSequences(config));
 violations.addAll(checkSequenceIsolation(config));

 return violations;
}
```

Listing 3 shows the Flutter grammar-validation entry point. All four structural rules are evaluated in a single pass and all violations are aggregated, allowing the app to report complete configuration feedback instead of stopping at the first error.

E. Implementation - Communication

Communication in the Blocks o' Code (v3) platform spans three layers: inter-block communication over I^2C, system-level control between the Brain and a TCP relay server, and application-level interaction with the Flutter companion app. As described in Section II, the four-pin pogo connectors carry the shared I^2C bus and common ground reference between blocks. The Brain block acts as the I^2C master, periodically scanning the bus to discover attached blocks, reading identification registers, and issuing commands such as configuration updates and `CMD_EXECUTE` to drive local peripherals on the child cubes.

Above this local bus, the Brain maintains a Wi-Fi connection to a lightweight TCP server on the local network. Whenever the physical configuration changes, the Brain converts the current block chain into a compact JSON `BlockConfiguration` structure and transmits it to the server. The server relays this message to the Flutter application, which parses the JSON, performs grammar validation, and updates its live block list and 3D visualization. Validation results and execution commands from the app follow the reverse path: the app sends a JSON message to the server, the server forwards it to the Brain, and the Brain routes the resulting actions back down to the child blocks over the I^2C bus.

To keep the entire path responsive and robust, the communication protocol uses short, line-delimited JSON messages with explicit `type` fields (for example, `CONFIG_UPDATE`, `VALIDATION_RESULT`, and `EXECUTE_COMMAND`) and periodic heartbeat traffic. A dedicated FreeRTOS communication task on the Brain manages the TCP socket, monitors heartbeat timeouts, and triggers automatic reconnect attempts when needed. This layered design allows button presses and other physical interactions on the child blocks to propagate through the Brain, relay server, and app, and then return as validated execution commands, while keeping latency and jitter low enough for the system to feel interactive in classroom use.

V. Mechanical Design

The mechanical enclosures were modeled in SolidWorks to create a durable cube-style housing for each block within the system. The enclosure design was developed to securely support the internal electronics while maintaining an intuitive and child-friendly form factor. Internal mounting features were integrated into the enclosure to precisely align the printed circuit boards, magnetic connectors, and pogo pin communication interfaces. These mounting points ensure consistent electrical contact between blocks while maintaining structural rigidity during repeated assembly and handling.

Cutouts were incorporated into the enclosure to provide access to all external components and interfaces. These include openings for the power switch, USB-C charging port, TFT touchscreen display, LED matrix module, and pogo pin connection interfaces. The placement of these cutouts was carefully designed to ensure accessibility while protecting internal components from accidental contact or damage during normal use.

Thermal venting was also incorporated into the enclosure design to allow passive airflow through the system. Ventilation holes located on the side panels help dissipate heat generated by the internal electronics, including the microcontroller, display, and audio amplifier. This passive cooling approach eliminates the need for active cooling components while maintaining safe operating temperatures.

The cube geometry and hinged lid mechanism allow the enclosure to be easily opened for assembly, maintenance, and debugging during development. At the same time, the external design maintains smooth surfaces and rounded edges to ensure that the blocks remain comfortable and safe for children to handle. Overall, the enclosure design balances durability, thermal management, and ergonomics while supporting the modular architecture of the Blocks o' Code system.

VI. Testing and Results

A. Spec 1: LED Selection Latency

System responsiveness was verified by measuring the latency between an LED color selection on the companion app and the corresponding hardware response on the LED Color Flash block. This latency was measured using a logic analyzer by timestamping the I^2C transaction initiated by the user input event and the resulting LED update command issued to the child block. This metric captures the local firmware path from input handling to visible output, and the firmware processes the state change efficiently enough for the visual feedback to feel instantaneous to the user.

Across ten runs, the measured latency had a mean of 13.1406 ms, with a minimum of 13.128 ms and a maximum of 13.206 ms. The design target was 50 ms or less. Therefore, the measured result remained well within specification. Just as important, the variation between runs was extremely small. That low spread indicates that the measured firmware path exhibits highly repeatable, deterministic timing behavior. In practice, this means the preview appears both immediate and consistent to the user.

B. Spec 2: System Communication Latency

End-to-end communication latency was defined as the time from a physical configuration change in the block chain to the corresponding update that appears in the UI

Fig. 4. External SolidWorks model showing the TFT display, LED matrix, and pogo-pin interfaces.

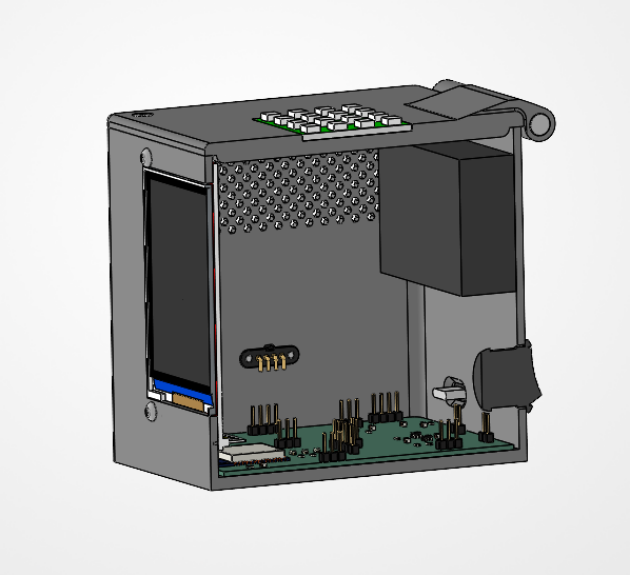

Fig. 5. Internal layout showing PCB mounting, pogo-pin connector, and speaker placement.

of the companion app. This path includes the Brain's I^2C rescan and JSON generation, Wi-Fi transmission to the TCP server, relay to the Flutter application, JSON parsing, grammar validation, and UI refresh. The design target for this metric was a maximum of 2000 ms to ensure that learners perceive the system as responsive and directly linked to their physical actions.

Latency was measured across ten independent trials. At the embedded side, the Brain toggled a GPIO line immediately upon detecting a new configuration; at the application side, a timestamp was logged the moment

TABLE I
LED SELECTION LATENCY (10 RUNS)

Run Number	Latency (ms)
1	13.206
2	13.131
3	13.142
4	13.133
5	13.133
6	13.129
7	13.133
8	13.138
9	13.133
10	13.128
Mean	13.1406
Std Dev	0.023
Min / Max	13.128 / 13.206

the updated configuration was rendered in the UI. Both endpoints were captured on video and measured frame-by-frame to derive each trial's end-to-end elapsed time. Table II summarizes the results. The observed latencies ranged from 33 ms to 70 ms, with a mean of 49.6 ms. All measurements were therefore more than an order of magnitude below the 2000 ms specification.

The relatively narrow spread between the minimum and maximum values indicates that the communication path is not only fast but also consistent under typical operating conditions. Users experience this as an almost immediate change in the on-screen visualization after rearranging blocks, which reinforces the cause-and-effect relationship between physical programming and system behavior.

TABLE II
SYSTEM COMMUNICATION LATENCY (10 RUNS)

Run Number	Latency (ms)
1	45
2	60
3	39
4	46
5	37
6	62
7	63
8	70
9	33
10	41
Mean	49.6
Std Dev	13.0
Min / Max	33 / 70

C. Spec 3: Electrical & I^2C Testing

Rise time measurements were used to evaluate I^2C performance under standard operating conditions. The measurement setup included:

- Oscilloscope probe connected to SDA/SCL test points (see Fig. 6)
- I^2C bus operating at 100 kHz
- Standard pull-up configuration (4.7 kΩ)

Rise time was defined as the interval between the 30% and 70% levels on the rising edge. Ten measurements were recorded, with rise times ranging from 270 ns to 300 ns and an average of 289 ns. These results are well below the 1000 ns maximum rise time specified for standard-mode I^2C operation. This confirms reliable communication across multiple connected blocks.

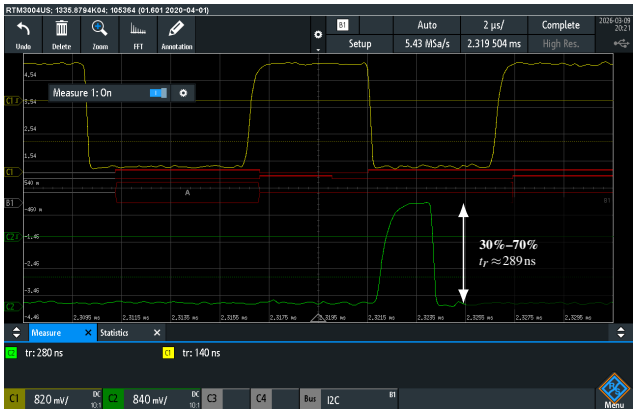

Fig. 6. I^2C SDA rise time at 100 kHz. The inset double-sided arrow marks the 30%–70% rise-time interval used for measurement; average across ten measurements was 289 ns, well below the 1000 ns standard-mode limit.

Additional Electrical Validation: Additional electrical validation included:

- Verification of regulator output stability at 3.3 V and 5 V
- Functional testing under battery operation
- Basic runtime validation of the battery system under typical load conditions

Overall, the electrical system met all functional requirements, with stable power delivery and reliable I^2C communication across the modular architecture.

D. Audio & Mechanical Testing

Mechanical and audio subsystems were tested to verify reliability under repeated use. Mechanical testing included repeated assembly cycles to evaluate enclosure durability, hinge performance, and magnetic connector alignment. The connectors maintained consistent electrical contact through the pogo-pin interface, and no structural degradation was observed.

TABLE III
I^2C RISE TIME (10 RUNS)

Run Number	Rise Time (ns)
1	280
2	300
3	270
4	270
5	300
6	290
7	300
8	290
9	300
10	290
Mean	289
Std Dev	12.0
Min / Max	270 / 300

Thermal performance was evaluated during extended operation. Passive ventilation provided sufficient heat dissipation, and internal temperatures remained within safe operating limits without active cooling.

Audio testing confirmed that the LM386 amplifier reliably drove the 8-Ω speaker with stable output and minimal distortion. Sound playback remained clear at maximum volume, and the bass boost configuration improved overall audio quality.

Overall, both subsystems met durability and performance requirements for classroom use.

VII. DISCUSSION

Blocks o' Code (v3) successfully operates as a cohesive interactive system. From the user's perspective, the workflow is straightforward: blocks are physically connected to form a program, the Brain block scans the arrangement, the companion app displays the detected logic and reports whether the sequence is valid, and valid programs can then be executed with immediate physical feedback. Output blocks such as the LED Color Flash and Music Sequence blocks allow the user to see and hear the result of the program, reinforcing the connection between physical arrangement and system behavior.

These results show that the platform already supports the core experience it was designed to provide. Rather than functioning as isolated hardware subsystems, the blocks, firmware, and app work together as a single educational system. The current prototype can detect valid and invalid configurations, prevent incorrect execution, and deliver responsive visual and audio feedback during operation. Future iterations will focus on expanding the available block library, improving connector robustness, and refining

the overall user experience, but the prototype already demonstrates the central value of the platform.

VIII. Conclusion

Blocks o' Code (v3) demonstrates the successful implementation of a modular embedded system [5] tailored for interactive programming education. By combining ESP32 control, I^2C communication, tactile hardware, and a responsive companion application, the project transforms abstract coding concepts into an intuitive, hands-on learning experience for beginners.

The system integrates touchscreen displays, LED matrices, programmable lighting, audio feedback, and magnetic pogo-pin connectivity into one unified platform. The modular cube architecture allows users to physically construct program sequences while receiving immediate visual and auditory feedback, reinforcing the relationship between program structure and system behavior. The standardized hardware design ensures compatibility between blocks while simplifying manufacturing and system integration. The physical blocks provide the structure of the program, the firmware detects and interprets that structure, and the companion app validates and visualizes it for the user, creating a tangible programming workflow that bridges the gap between abstract logic and physical interaction.

Testing results demonstrate that the mechanical, electrical, and communication subsystems operate reliably under repeated use conditions. The enclosure design provides durability and adequate thermal management, while the communication architecture enables stable data transfer between blocks and the central controller. Together, these components form a cohesive system capable of supporting interactive educational applications.

Future iterations will focus on expanding the block library, hardening connector reliability, and conducting structured classroom evaluations to measure learning outcomes. Overall, Blocks o' Code (v3) provides a strong foundation for continued development and presents a practical, scalable model for hands-on beginner programming education through tangible interaction, real-time feedback, and modular embedded design.

Acknowledgment

The authors would like to sincerely thank Dr. Mike Borowczak, Dr. Arthur Weeks, and Dr. Sreeram Sundaresh for their invaluable guidance, mentorship, and continued support throughout the development of the Blocks o' Code project. Their technical insight and feedback were instrumental in shaping the system architecture, hardware design, and overall direction of the project. The authors also appreciate their encouragement during the design, testing, and refinement stages of the system.

The team would also like to acknowledge the support provided by the University of Central Florida's Electrical and Computer Engineering program, which provided access to laboratory resources, development tools, and facilities necessary for building and testing the prototype system. These resources played an important role in enabling the successful implementation of the Blocks o' Code platform.

Biography

Jordan Merkel is a 21-year-old senior graduating as a Computer Engineering student at the University of Central Florida. She plans to start full-time at Advanced Micro Devices (AMD) after graduation as an AI-Graphics RTL Design Engineer, and plans to pursue graduate education, with a goal to come back and teach.

Destiny Ellenwood is a 23-year-old senior graduating as a Computer Engineering student at the University of Central Florida. She plans to start full-time as a Test Electrical Engineer at Raytheon after graduation.

Annesley Kolb is a 23-year-old senior graduating as Electrical Engineering student at the University of Central Florida with industry experience at Turbine Technology Services, focusing on electrical schematics, control systems, and panel design. She aims to start a career where she can continue to gain hands-on skills.

Camilla Torres is a senior Electrical Engineering student at the University of Central Florida, graduating in May 2026. She conducts research at the College of Nursing and plans to pursue a master's degree, with the goal of developing assistive technologies, while also aspiring to start her own bakery.

References

[1] Espressif Systems, "ESP32 Series Datasheet," Espressif Systems, 2024.

[2] NXP Semiconductors, "I^2C-bus specification and user manual," NXP, 2021.

[3] M. Resnick et al., "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[4] H. Ishii and B. Ullmer, "Tangible bits: Towards seamless interfaces between people, bits and atoms," in *Proc. SIGCHI Conf. Human Factors in Computing Systems*, 1997, pp. 234–241.

[5] F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*. New York, NY, USA: Wiley, 2002.

[6] Amazon Web Services, "FreeRTOS Real-Time Operating System," [Online]. Available: https://www.freertos.org, 2024.

[7] Google LLC, "Flutter – Build apps for any screen," [Online]. Available: https://flutter.dev, 2024.

[8] Top Power ASIC Corp., "TP5100 1A/2A Switch Mode Lithium Battery Charger for 1–2 Cells in Series," Datasheet, 2019.

[9] Texas Instruments, “LM386 Low Voltage Audio Power Amplifier,” Datasheet, rev. D, 2023.
[10] KiCad Developers Team, “KiCad EDA,” [Online]. Available: https://www.kicad.org. Accessed: 2026.