Summer Internship Report

on

OpenHW-Studio (OSHW): Browser-Based Hardware Simulation & Educational Platform Development

Submitted by

Md. Danish

B.Tech in Computer Science Engineering,
VIT Bhopal University, Bhopal

Under the Guidance of

Prof. Prabhu Ramachandran

Principal Investigator, FOSSEE Project

Department of Aerospace Engineering,

Indian Institute of Technology Bombay

July 2026

Acknowledgement

I would like to express my deepest gratitude to **Prof. Prabhu Ramachandran** for providing me with the opportunity to be a part of the **FOSSEE Internship Program** and for supporting open-source engineering tool development at **IIT Bombay**. His vision and leadership have inspired students and researchers to actively contribute to the open-source ecosystem.

I would also like to sincerely thank **Prof. Rajesh Kushalkar, Senior Project Manager, Open Source Hardware (OSHW), FOSSEE, IIT Bombay**, for his valuable guidance, encouragement, and continuous support throughout the internship. His mentorship greatly contributed to my learning and professional growth.

My heartfelt appreciation goes to my mentors, **Mr. Pratik Bhosale, Project Research Associate, and Mr. Pratik Nemane, Project Research Assistant**, for their constant technical guidance, constructive feedback, and encouragement throughout the project. Their insights played a key role in refining ideas, overcoming challenges, and ensuring the successful completion of the tasks assigned to me.

I would also like to thank my fellow interns and colleagues at **OpenHW Studio** for their support, collaboration, and constructive discussions throughout the internship. Their motivation and teamwork created a positive learning environment and contributed significantly to my overall experience.

Finally, I extend my sincere gratitude to everyone associated with the **FOSSEE Project, IIT Bombay**, for fostering an environment of innovation, collaboration, and continuous learning. This internship has been an invaluable experience that strengthened my technical skills and motivated me to continue contributing to the open-source community.

MD Danish

VIT Bhopal University

Declaration

This internship proved to be a highly rewarding educational journey, granting me the chance to support the creation of virtual hardware modules while integrating functional improvements and debugging within **OpenHW Studio.** The experience offered significant exposure to professional open-source software practices, micro-controller simulation environments, and integrated development processes. The technical competencies and hands-on insights developed throughout this period will serve as a vital foundation for my upcoming academic projects and career objectives.

I wish to further extend my sincere appreciation to the members of the **FOSSEE Team** for their seamless coordination, technical mentorship, and unwavering encouragement during the program. Their collaborative spirit and proactive assistance fostered a productive workspace, proving instrumental in the effective delivery of all project milestones and responsibilities.

MD Danish

VIT Bhopal University

Index

GitHub Commit History

Of **Md. Danish**, **danish9661** in **OpenHW-Studio**

Repository Name	Total Commits	PR Merged	Lines Added (+)	Lines Deleted (-)
openhw-studio-frontend	265	106	334,321	125,216
openhw-studio-backend	169	71	33,526	4,674
openhw-studio-emulator	102	58	77,956	45,420
openhw-studio-cli	12	8	19,877	169
openhw-studio-docs	8	6	15,017	260
openhw-studio-examples	4	3	58,071	1
openhw-studio-compile-server	1	1	112,309	0
openhw-studio-autograding	2	1	6,625	32
openhw-studio-autowiring	1	1	2,169	0
openhw-studio-gateway	1	1	1,729	0
Total	**554**	**236**	**480,699**	**175,741**

Chapter 1: Introduction & Project Overview

1.1 Project Overview

OpenHW Studio is an advanced, browser-based, open-source hardware simulation and educational platform developed under the **FOSSEE (Free and Open Source Software for Education) project at IIT Bombay**. The platform addresses a critical gap in embedded systems education by providing an interactive, web-accessible virtual laboratory environment. Through this virtual laboratory, students, educators, and engineers can design, wire, and simulate complex electronic circuits entirely inside a web browser, eliminating the immediate need for physical microcontrollers, breadboards, sensors, or actuators.

By providing a virtual canvas populated with realistic representations of hardware components, OpenHW Studio lowers the entry barriers to embedded systems education, mitigates the logistical challenges of managing physical hardware labs, and enables rapid prototyping of electronics designs.

1.2 Motivation and Educational Impact

In traditional educational settings, teaching embedded systems and electronics requires substantial capital investment in physical laboratories. These physical setups pose several key challenges:

1. **High Cost and Resource Limits:** Physical microcontrollers (like Arduino, ESP32, and Raspberry Pi Pico boards) and corresponding sensor modules must be purchased in quantity, which can be prohibitive for underfunded schools and universities.
2. **Hardware Vulnerability:** Physical components are highly susceptible to damage. Inexperienced students frequently burn out LEDs, damage ICs, or destroy microcontrollers due to short circuits, reverse polarity, or voltage overloads.
3. **Physical Access Constraints:** Students can typically only experiment during scheduled lab hours when they have physical access to the equipment.

OpenHW Studio solves these issues by shifting the hardware lab into the cloud. It provides a risk-free sandbox where students can experiment freely. If a student makes a wiring error, the simulator flags the error dynamically (through advanced topological validation rules) instead of physically damaging any hardware. This encourages a pedagogical cycle of experimentation, failure, and debugging. Furthermore, since the entire system executes in the browser, students can access their labs at any time, from any computer, fostering a self-paced learning environment.

1.3 Role and Responsibilities

During my summer internship as a core software and simulation engineer on the OpenHW Studio development team, my responsibilities spanned across the entire software stack. I worked directly within the three primary repositories that define the platform: the React/Vite Frontend

(`openhw-studio-frontend`), the TypeScript Emulator Engine (`openhw-studio-emulator`), and the Node.js/Express Backend (`openhw-studio-backend`).

My core streams of work involved:

- **Emulation Engine Engineering:** Designing, writing, and refactoring the register-level simulation logic, physical behaviors, and configuration manifests for multiple advanced hardware sensors, actuators, and communication modules.
- **Simulation Pipeline Development:** Overhauling the frontend execution runners, implementing background multi-threaded simulation loops (Web Workers), integrating the Monaco editor, and building the highly complex WebAssembly (WASM) execution environment to simulate network-capable microcontrollers.
- **Backend Compilation Scaling:** Hardening security configurations, resolving memory leak and compiler queue failures, and implementing a massive cached compilation system (storing pre-compiled library and binary objects) to prevent server-side Out-Of-Memory (OOM) crashes under concurrent student loads.
- **CI/CD Containerization:** Packaging the backend with toolchain pre-installations in Docker and managing automated deployment workflows via GitHub Actions.

1.4 Core Technology Stack

The platform is engineered using modern, open-source technologies optimized for web delivery:

- **Frontend UI:** Built using **React** and **Vite** for rapid hot-module reloading and highly optimized assets.
- **Editor:** Integrated with **Monaco Editor** to provide professional-grade code editing, autocomplete, and syntax highlighting in the browser.
- **Simulation Engine:** Written in **TypeScript** and in **Rust**, executing natively within the browser using WebAssembly (WASM) compiles of emulators.
- **Thread Isolation:** Leveraging **Web Workers** and **SharedArrayBuffers** to isolate the heavy mathematical calculations of the simulator from the main UI thread, ensuring a smooth, responsive 60fps canvas interface.
- **Backend Server:** Engineered on **Node.js** and **Express**, providing compilation controllers, WebSocket gateways, and point-based rate limiters.
- **Database:** Utilizing **MongoDB** with **Mongoose** schemas for user state persistence.
- **CI/CD & DevOps:** Containerized with **Docker**, using **Nginx** for routing, and deployed via **GitHub Actions** workflows.

System Architectural Overview: OpenHW Simulation Pipeline

This document provides a deep, technical breakdown of the OpenHW Studio compilation and execution pipeline, specifically tracing the path from a user clicking "Run" on an Arduino Uno sketch to a virtual LED blinking on the screen.

The architecture is built for extreme performance, leveraging multi-threading (Web Workers), zero-copy data routing, and a highly aggressive dual-layer caching strategy.

1. Dual-Layer Caching Architecture

Before any compilation toolchain is invoked, the system attempts to bypass the heavy compilation step using a two-tier hashing mechanism.

Layer 1: The Frontend Client Cache (IndexedDB)

When the user initiates a run, simulatorService.js intercepts the payload.

- **Payload Hashing**: A SHA-1 hash is generated directly in the browser. The hash encompasses the raw source code, included libraries, and the Fully Qualified Board Name (fqbn).
- **Offline Storage**: The system checks the browser's local IndexedDB (OpenHW_Compile_Cache).
- **Network Bypass**: If a cache hit occurs, the .hex binary is retrieved instantly from the local database. The backend is never contacted, completely bypassing network latency. This is crucial for offline support and rapid, iterative local testing.

Layer 2: The Backend Server Cache (RAM & Disk)

If the frontend cache misses (e.g., the user is compiling this specific code for the very first time), the raw payload is sent via a POST request to the backend /compile endpoint.

- **Server Hashing**: The backend's compileController.js calculates its *own* SHA-1 hash of the incoming payload.
- **In-Memory RAM Pool**: The backend first checks an ultra-fast JavaScript Map in RAM. To protect server stability, this pool is strictly capped (e.g., maximum 5 concurrent items) and governed by a 30-minute Time-To-Live (TTL) eviction policy.
- **Persistent Disk Fallback**: If it misses the RAM pool, the server checks the physical hard drive (temp/pico-uno-compile-cache). If found on disk, it is loaded back into RAM and returned.
- **Global Sharing**: This backend cache is global. If Student A compiles a template sketch, Student B will instantly hit the server cache and avoid triggering a costly compilation.

2. The Isolated Compilation Pipeline

If both Layer 1 and Layer 2 caches miss, physical compilation must occur.

- **Workspace Generation**: The backend provisions a temporary, isolated build folder for the request.
- **Toolchain Invocation**: The backend executes the official arduino-cli as a subprocess within this isolated environment.
- **Artifact Extraction**: Once arduino-cli succeeds, the backend extracts the resulting .hex file.
- **Cache Seeding**: The backend saves the .hex to its Layer 2 Disk cache, caches it in RAM, and returns it to the frontend.
- Upon receiving the .hex, the frontend saves it into the Layer 1 IndexedDB cache for future local runs.

3. Web Worker Orchestration

The main UI thread (React) **does not** emulate the CPU. Running an emulator at 16MHz on the main thread would freeze the browser.

- **Simulation Worker (simulation.worker.ts)**: The frontend spawns a dedicated Web Worker to handle the heavy lifting.
- **Payload Transfer**: The frontend sends an INIT message containing the compiled .hex string to this worker.
- **Zero-Copy Channels**: (For advanced workflows) The SimulatorPage can also spin up separate Rendering and Networking workers, connecting them directly to the Simulation Worker via MessageChannel ports so they can share memory buffers without passing through the UI thread.

4. Multi-Architecture Runner Abstraction

Inside simulation.worker.ts, the system uses an abstraction layer called the "Runner" to interface with different CPU emulators depending on the board chosen.

- **avr-runner.ts (Arduino Uno/Mega)**:
    - Initializes the low-level avr8js CPU engine.
    - Loads the .hex payload into the virtual AVR Flash memory.

    - Binds JavaScript listeners to specific memory addresses (e.g., PORTB for digital pins 8-13 on an Uno).
- **rp2040-runner.ts (Raspberry Pi Pico)**:
    - Initializes the rp2040js emulator.
    - Loads the .uf2 or .hex payload, often including a BootROM startup sequence.
    - Handles complex peripherals like the PIO (Programmable I/O) and multi-core synchronization unique to the RP2040 architecture.
- **esp32-runner.ts / esp32-engine.js (ESP32)**:
    - Interfaces with a highly complex WASM-compiled emulator for the ESP32 (typically a WASM build of QEMU or a custom Tensilica emulator).
    - Manages huge memory structures, partitions, and an isolated IP networking stack (bridged to the network.worker.ts).
- **backend-proxy-runner.ts (Cloud Simulation)**:
    - Used when the target board is too heavy to emulate in a browser tab (e.g., STM32 or advanced ESP32 scenarios).
    - Instead of running an emulator locally, this runner acts as a lightweight proxy. It streams state changes and I/O via WebSockets from a remote cloud server running the actual hardware simulation, making the frontend components think they are running locally.

5. Emulation Loop & Component Updates

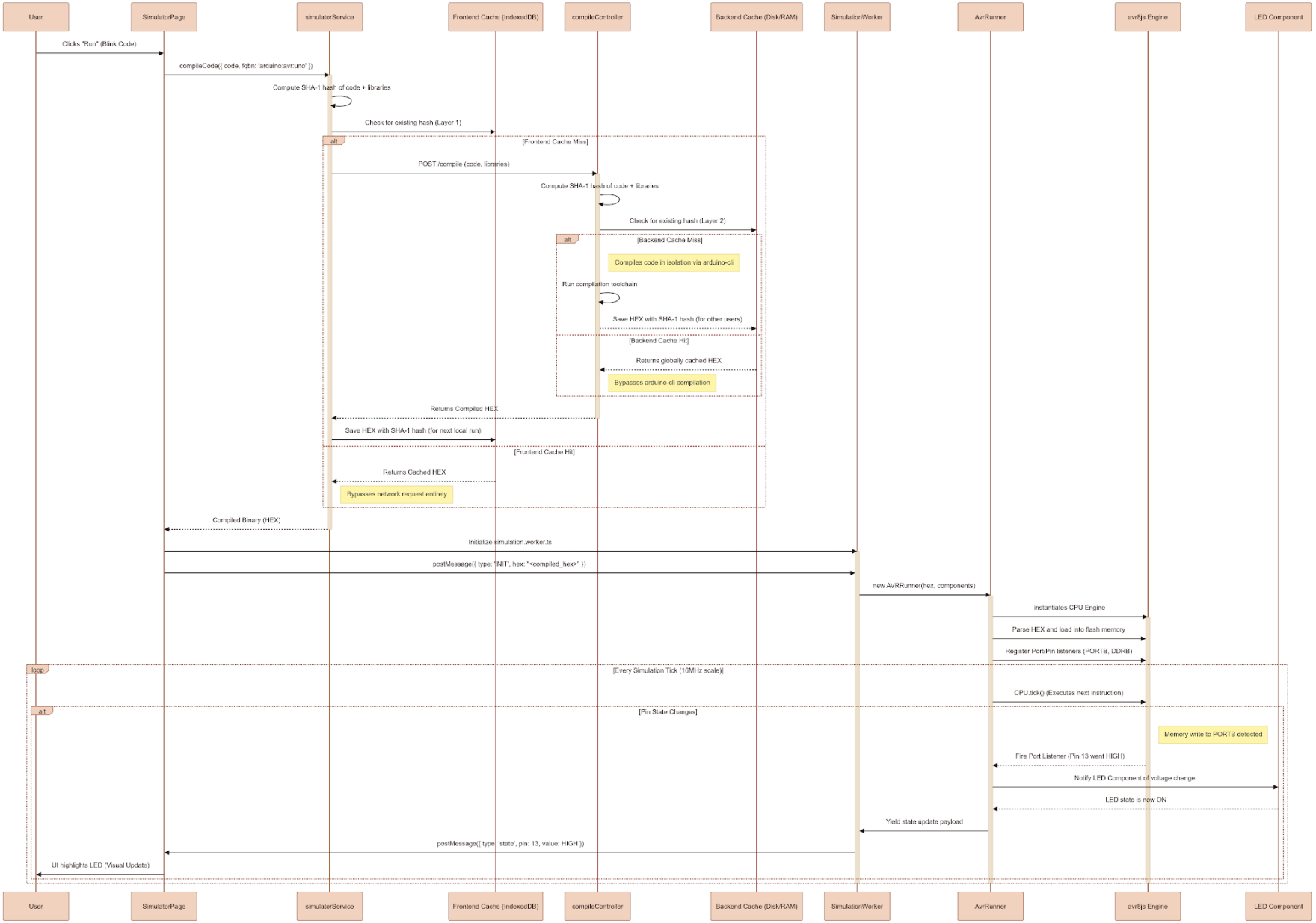
This is where the code actually "runs".

- **The Tick**: AvrRunner constantly loops, calling CPU.tick(). This forces the avr8js engine to execute the next assembly instruction in the .hex file.
- **Memory Interception**: If the user's C++ code contained digitalWrite(13, HIGH), the virtual CPU eventually executes an assembly instruction (like sbi PORTB, 5) that modifies the memory address for PORTB.
- **Component Notification**: The listener set up by AvrRunner instantly detects this memory write. It determines that Pin 13 just went HIGH.
- **Physics Calculation**: AvrRunner passes this state change to the internal virtual component models (e.g., a virtual LED attached to Pin 13), calculating voltage drops or state changes.

6. Safety, Telemetry, and Grading

The architecture includes several parallel monitoring systems that operate alongside the main execution loop:

- **Circuit Validation (Pre-compile)**: Before compilation even begins, SimulatorPage runs a runCircuitValidation() check. If the user has short-circuited a breadboard or exceeded current limits, it blocks execution instantly to prevent simulated "damage."
- **Telemetry & Deep Silicon (Runtime)**: During execution, the runners can extract cycle-accurate telemetry (like exact CPU cycle counts, SPI bus timings, or memory register states) without blocking the main emulation loop.
- **Grading Engine Worker**: If the platform is running an assignment, the simulation worker waits for a TEACHER_KEY_CAPTURE_COMPLETE signal. It then spawns a completely isolated grading-engine.worker.ts. This worker takes the telemetry data and bundles it through a WASM key generator, outputting a secure .bin grading file without ever trusting the backend.


User
SimulatorPage
simulatorService
Frontend Cache (IndexedDB)
compileController
Backend Cache (Disk/RAM)
SimulationWorker
AvrRunner
avr8js Engine
LED Component
Clicks "Run" (Blink Code)
compileCode({ code, fqbn: 'arduino:avr:uno' })
Compute SHA-1 hash of code + libraries
Check for existing hash (Layer 1)
alt
[Frontend Cache Miss]
POST /compile (code, libraries)
Compute SHA-1 hash of code + libraries
Check for existing hash (Layer 2)
alt
[Backend Cache Miss]
Compiles code in isolation via arduino-cli
Run compilation toolchain
Save HEX with SHA-1 hash (for other users)
[Backend Cache Hit]
Returns globally cached HEX
Bypasses arduino-cli compilation
Returns Compiled HEX
Save HEX with SHA-1 hash (for next local run)
[Frontend Cache Hit]
Returns Cached HEX
Bypasses network request entirely
Compiled Binary (HEX)
Initialize simulation.worker.ts
postMessage({ type: 'INIT', hex: "<compiled_hex>" })
new AVRRunner(hex, components)
instantiates CPU Engine
Parse HEX and load into flash memory
Register Port/Pin listeners (PORTB, DDRB)
loop
[Every Simulation Tick (16MHz scale)]
CPU.tick() (Executes next instruction)
alt
[Pin State Changes]
Memory write to PORTB detected
Fire Port Listener (Pin 13 went HIGH)
Notify LED Component of voltage change
LED state is now ON
Yield state update payload
postMessage({ type: 'state', pin: 13, value: HIGH })
UI highlights LED (Visual Update)

Chapter 2: Hardware Emulator Development (openhw-studio-emulator)

The core simulation capabilities of OpenHW Studio are governed by the `openhw-studio-emulator` engine. This engine is written entirely in TypeScript and runs locally in the browser to compute physical and electrical state updates. My primary contributions within this repository focused on designing, coding, and debugging the virtual components, their communication protocols, and their integration with the microcontroller runners.

2.1 Architecture of Simulated Components

To maintain scalability, every virtual hardware component is organized under a modular directory containing five core files:

4. **`manifest.json` (Specification):** Defines metadata such as component dimensions, default attributes (e.g. ambient temperature or atmospheric pressure), and pin arrangements (physical pin IDs, types, coordinates, and labels). Bounding boxes (`BOUNDS`) are defined here to handle layout collisions.
5. **`logic.ts` (Behavioral Simulation):** The state machine of the component. It processes pin voltage transitions, models internal register arrays, executes calculation loops, and updates the component's internal state.
6. **`ui.tsx` (Visual Rendering):** A React component that renders the visual representation using Scalable Vector Graphics (SVG). It dynamically translates internal logical states into visual outputs (e.g., updating digits on a 7-segment display, blinking status LEDs, rotating motor shafts, or rendering real-time telemetry panels).
7. **`validation.ts` (Electrical Verification):** Executed prior to launching the simulation. It parses the connection nets to verify that the component is wired correctly (e.g. checks that the power pin connects to VCC and that data pins are routed to valid protocol pins).
8. **`index.ts` (Registration):** Standardizes exports for runtime instantiation by the registry.

2.2 Detailed Sensor & Actuator Emulation

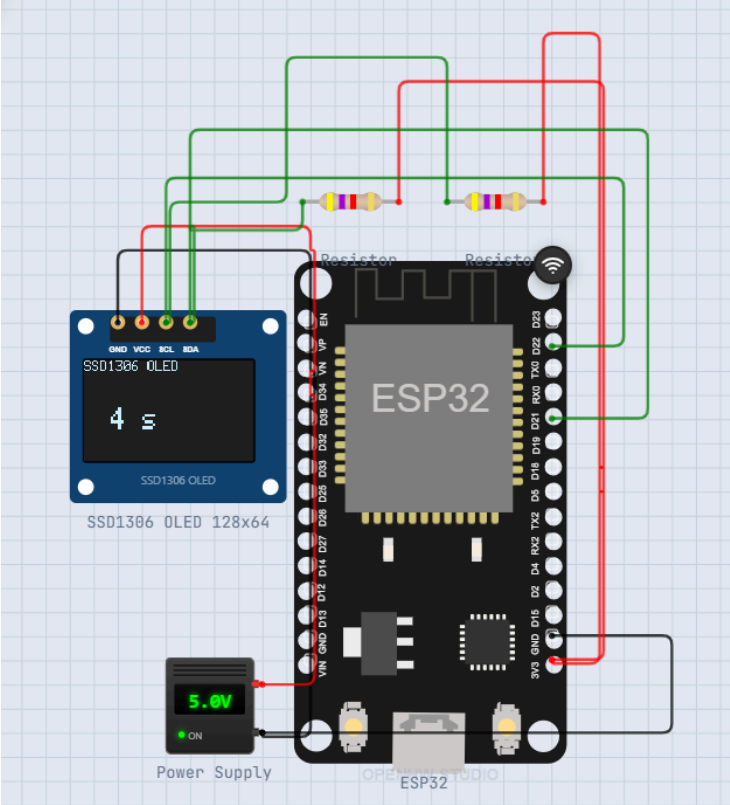
Over the course of the internship, I developed and resolved critical emulation bugs for nine key components.

2.2.1 SSD1306 OLED Display

The SSD1306 is a monochrome OLED display that communicates over the Inter-Integrated Circuit (I^2C) protocol.

- **Protocol & Addressing:** Emulated on the standard 7-bit I^2C device address. It detects control bytes to differentiate between command (0x00) and data (0x40) streams, and supports burst transmission modes.

- **Register-Level Emulation:** Simulates the internal memory mapping and hardware configurations including Contrast (0x81), Display Offset (0xD3), Multiplex Ratio (0xA8), and COM Scan Directions (0xC0/0xC8). It faithfully models Horizontal, Vertical, and Page Addressing modes for accurate pixel placement.
- **Measurement Calculations:** Maintains an internal 1024-byte VRAM buffer (128x64 pixels). Rather than immediately flushing every I^2C byte to the DOM, the logic updates the VRAM in the background and batches synchronization to the React frontend at a stable 60FPS to optimize CPU cycle budgets.
- **Power and Validation:** Implemented telemetry hooks to calculate and broadcast the live VRAM fill percentage and active addressing mode.

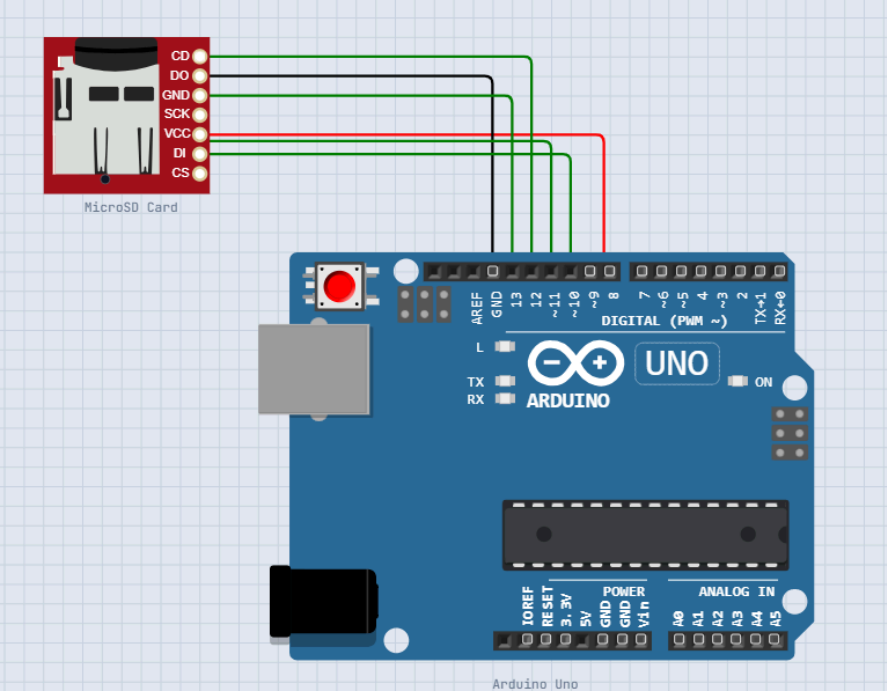


2.2.2 SD Card Module

The SD Card component provides a fully functional virtual filesystem communicating over the Seria Peripheral Interface (SPI) protocol.

- **Protocol & Addressing:** Implements standard SD SPI commands including CMD0 (Reset), CMD8 (Voltage Check), CMD55 (App Command), and CMD17/CMD24 for Single Block Read/Writes using the 0xFE data token.
- **Register-Level Emulation:** Wraps a WebAssembly compilation of LittleFS (littlefs-wasm). It provisions a virtual block device (e.g., 2048 KB capacity mapped to 512-byte blocks) and translates SPI block read/write requests directly into LittleFS flash operations.

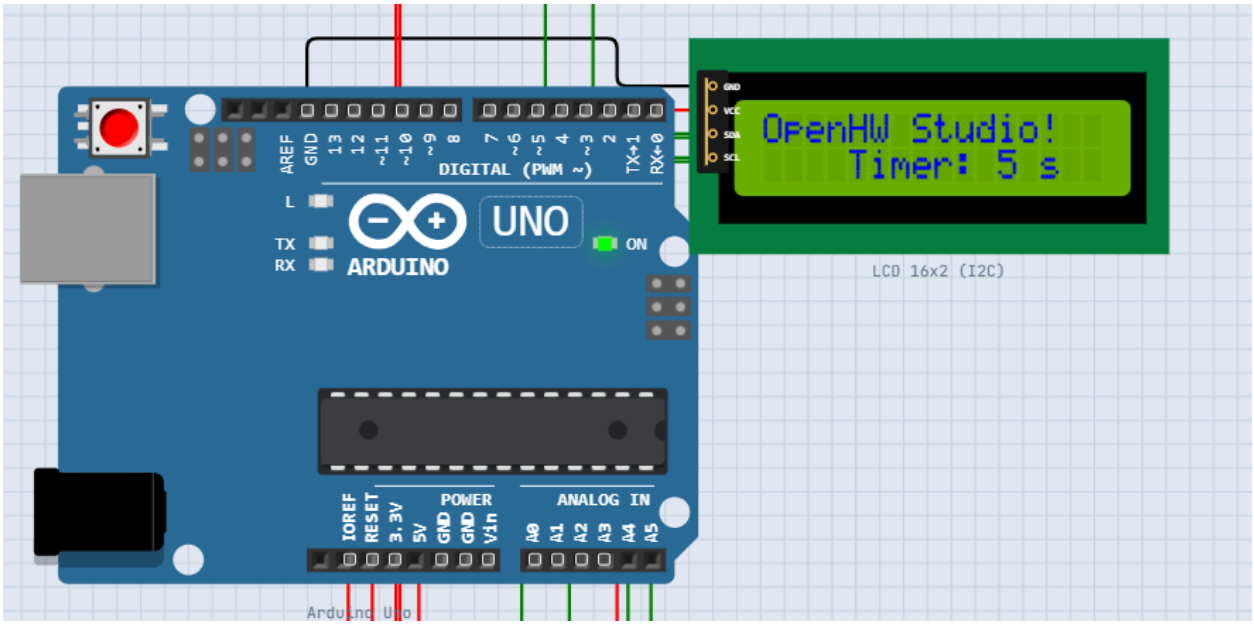
- **Measurement Calculations:** Tracks bytes in/out and active chip select (CS) states. Read/write operations emulate physical latency by buffering command frames (6 bytes) and returning appropriate R1 responses (0x01, 0x00) before data payloads.
- **Power and Validation:** Implemented power monitoring logic. If the VCC pin falls below 2.0V, the SPI interface is disabled and returns 0xFF. The component context menu allows users to format the drive, upload files, and view filesystem metrics.



2.2.4 Character LCD Display (i2c and spi)

The LCD1602 is a standard 16x2 character LCD, emulated for both direct parallel and I^2C backpack (PCF8574) interfaces.

- **Protocol & Addressing:** Driven by a centralized HD44780Controller engine. For the I^2C variant, it intercepts 8-bit payloads and decodes the nibbles into RS, RW, EN, and Data lines for the HD44780 controller.
- **Register-Level Emulation:** Simulates the internal DDRAM (Display Data RAM) for character mapping and CGRAM (Character Generator RAM) for custom characters. It processes standard instructions like Clear Display (0x01) and Return Home (0x02).
- **Measurement Calculations:** To prevent DOM starvation from aggressive firmware updates, the logic implements a 40ms debounce flush interval. It only synchronizes state to the frontend UI when the I^2C bus is idle, while forcing a flush every 200ms to guarantee visibility.
- **Power and Validation:** Implemented telemetry hooks to stream the decoded text content back to the diagnostic panel, allowing users to read the display output programmatically without inspecting the canvas.

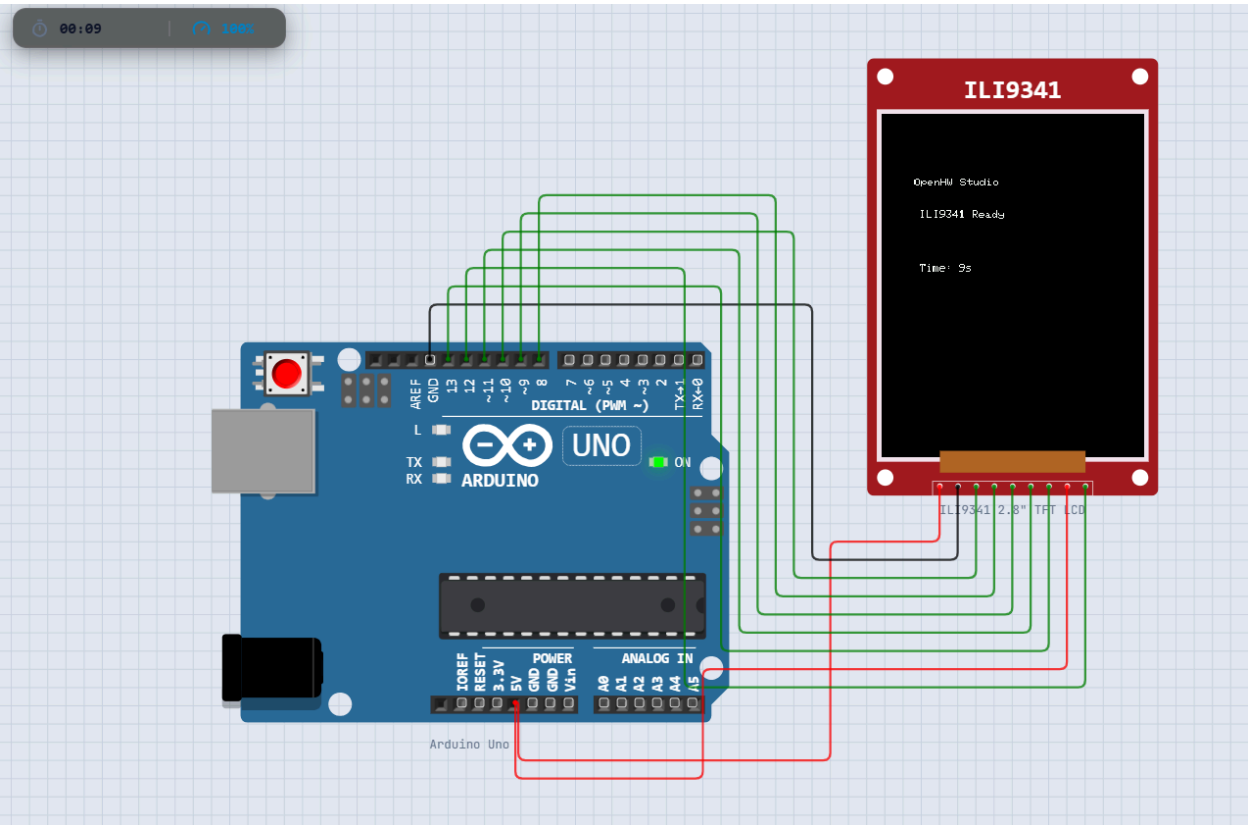


2.2.5 TFT LCD Display (ILI9341)

The ILI9341 is a high-resolution color TFT display communicating over a high-speed SPI bus.

- **Protocol & Addressing:** Captures SPI command frames utilizing the Data/Command (DC) pin state to differentiate instructions from pixel payloads. It supports critical commands like CASET (0x2A), PASET (0x2B), and RAMWR (0x2C).
- **Register-Level Emulation:** Manages a large internal VRAM buffer representing 240x320 pixels. It natively decodes 16-bit RGB565 color payloads received over SPI and translates them into a 24-bit RGB888 buffer for the HTML5 Canvas renderer.
- **Measurement Calculations:** Implements a hardware-accelerated DMA (Direct Memory Access) bypass loop. When attached to the RP2040, the emulator can bypass SPI bit-banging entirely and read the frame buffer directly from the virtual MCU's memory at 60Hz, achieving real-time video playback.
- **Power and Validation:** Monitors the VCC pin to power down the display. It computes a lightweight CRC32 checksum of the VRAM to broadcast efficient, compact telemetry snapshots

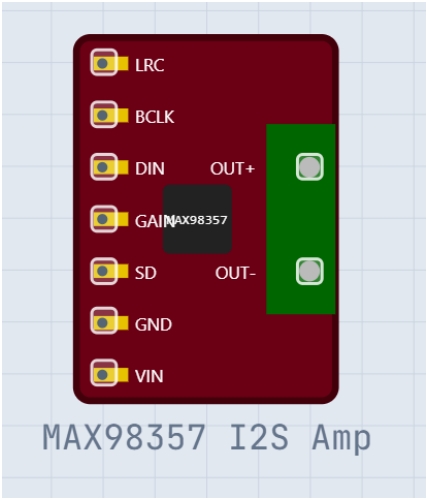
over WebSockets without flooding the network.



2.2.6 I2S Audio Amplifier (MAX98357A)

The MAX98357A is a digital PCM audio amplifier communicating over the Inter-IC Sound (I2S) protocol.

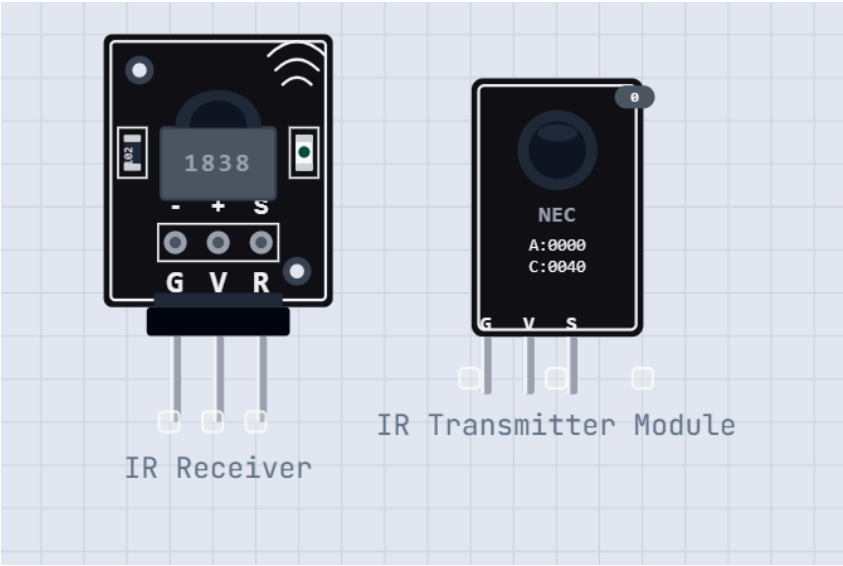
- **Protocol & Addressing:** Inherits from the base I2SProtocol handler. It decodes synchronous bitstreams across the BCLK (Bit Clock), LRC (Word Select), and DIN (Data In) pins to reconstruct audio samples.
- **Register-Level Emulation:** Captures PCM frames based on the configured bit-depth (e.g., 16-bit or 32-bit). The internal logic calculates the physical analog voltage represented by the digital sample.
- **Measurement Calculations:** Reconstructs the differential analog output. It maps the 32-bit signed integer sample to a ±1.65V swing, driving the physical OUT+ and OUT- pins inversely to simulate a Class D amplifier output.
- **Power and Validation:** Implemented telemetry hooks to stream the live configured bits-per-frame and provides a visualization of the active differential audio waveform being generated by the simulation engine.



2.2.7 IR Receiver & Remote

The IR Receiver emulates a TSOP38238/VS1838B infrared sensor tuned to a 38kHz carrier frequency.

- **NEC Protocol Demodulation:** Simulates the NEC packet format: a 9ms leading AGC pulse, followed by a 4.5ms space, and 32 bits of binary command data (using space-encoded pulse duration).
- **Command Queue:** Tapping buttons on the virtual remote control in `ui.tsx` pushes 32-bit hex values (e.g. `0xFF30CF`) into a pulse queue, which then drives the digital output pin (`OUT`) active-LOW with precise pulse timings.
- **Validation:** Verifies the `OUT` pin is wired to a digital input capable of hardware interrupts to prevent missed packets.

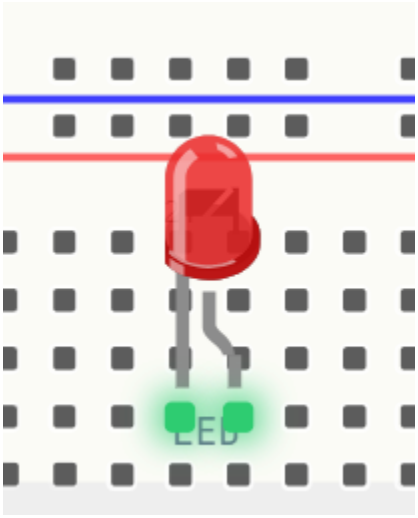


2.3 Circuit Validation and Snapping Physics

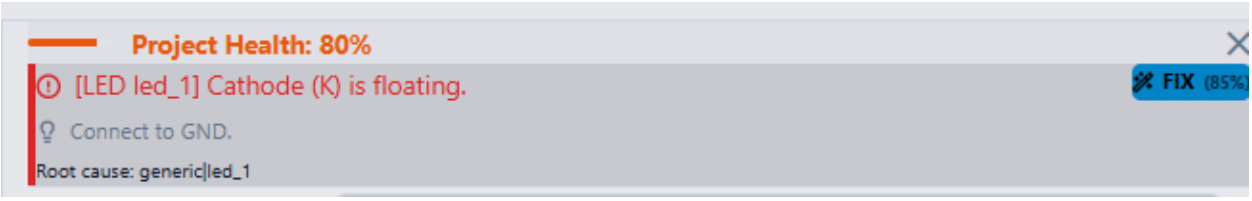
To assist students in debugging physical connection errors, I developed a localized circuit validation engine (`validation.ts`). When the user clicks "Start Simulation", this module runs topological connectivity checks:

- **Short Circuit Detection:** Traces VCC-to-GND low-resistance nets. If a direct path exists, it halts simulation and renders a red warning outline.

- **Breadboard Grid Snapping:** Intercepts component coordinate update events. The positioning coordinates are mapped through a snapping function that snaps component pins onto the coordinate coordinates of the virtual breadboard's 0.1-inch grid columns and rows, ensuring solid electrical routing.



- **Floating State Warnings:** Identifies active inputs (such as reset pins or digital inputs) that lack pull-up/pull-down connections, warning the student of unpredictable floating states.



Chapter 3: Frontend Architecture & Simulation Pipeline (openhw-studio-frontend)

The user interface of OpenHW Studio must handle complex, multi-layered visual states while executing real-time simulations. To deliver a responsive experience, I designed and implemented a multi-threaded architecture within the `openhw-studio-frontend` repository. This chapter outlines the UI canvas mechanics, Monaco IDE integration, Web Worker simulation loops, and the network-capable WebAssembly runner.

3.1 User Interface and Canvas Mechanics

The platform is built as a single-page application (SPA) powered by **React** and bundled via **Vite**. The main simulation workspace features an interactive canvas where users place, drag, rotate, and wire virtual components.

- **Pannable and Zoomable Workspace:** Implemented high-performance mouse and touch event listeners to manage panning and zooming. The transformation matrices are applied directly to the root SVG group, keeping vector lines clean at all scaling levels.
- **Quick-Add Dialog:** Integrated a double-click shortcut that pops up a fuzzy-search dialog, allowing students to filter and instantiate components instantly without using the sidebar.
- **IndexedDB Local Cache:** To support offline access and prevent work loss during page reloads, I built an auto-save loop that commits the canvas state JSON to `IndexedDB` every five seconds.
- **Cache Invalidation:** To prevent old service workers from serving stale UI bundles after backend updates, I added startup logic that unregisters legacy service workers and forces a clean cache reload.

Simulation Page Features & UI Architecture

Built with React and an SVG-based rendering canvas, it provides a comprehensive workspace for users to design, code, and test their circuits. Below is a detailed breakdown of the key features and panels available on the Simulation Page.

1. TopTool Menu (Top Navigation)

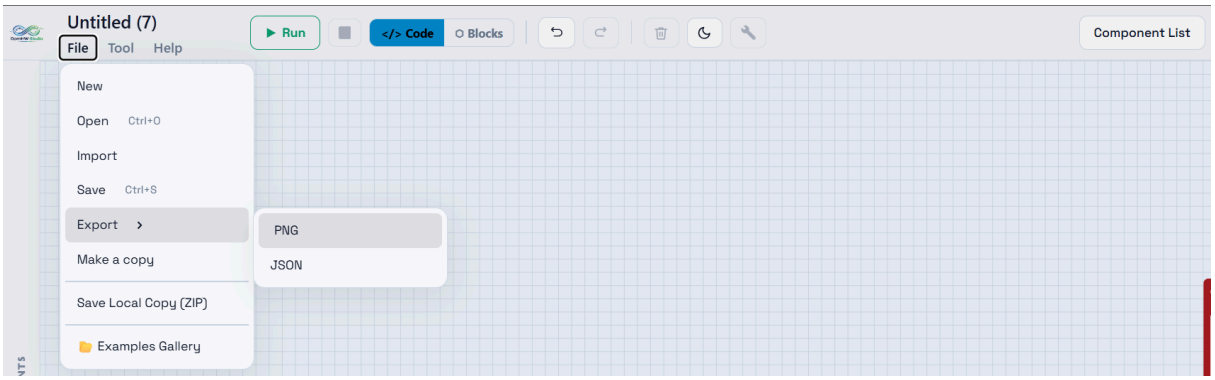
The top navigation bar acts as the primary control center for the workspace. It houses critical simulation controls including:

Compile & Run: Instantly triggers the backend/client compilation and boots the simulator execution engines (WASM, QEMU, or Renode depending on the board).

Save & Share: Persists the current workspace state to the database and generates shareable project links.

Layout Management: Allows users to reset the view, center the circuit, toggle grid snapping, and manage the canvas layout.

PNG Export/Import: The entire circuit diagram can be serialized and exported as a high-resolution PNG image (circuit.png). This feature is heavily utilized for generating documentation and assignment submissions. Furthermore, OpenHW Studio embeds project metadata into the PNG (steganography), allowing users to "import" a PNG file back into the platform to fully reconstruct the working circuit and code.



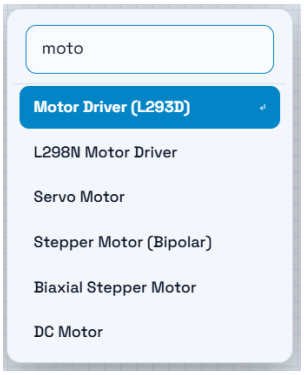
2. Quick Add Functionality (QuickAddPortal.jsx)

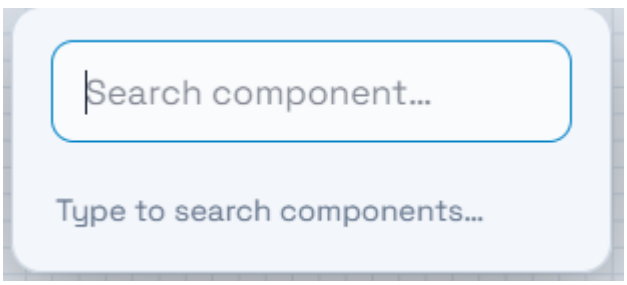
To accelerate the circuit design process, the **Quick Add** feature allows users to rapidly instantiate components without manually browsing the full library panel.

Triggered by a designated hotkey or a double-click on the canvas, a floating QuickAddPortal modal appears directly at the cursor's location.

It uses a fuzzy-matching search bar to instantly find components (e.g., typing "oled" instantly brings up the SSD1306 OLED display).

Selecting the component drops it directly onto the canvas at the exact coordinates of the double-click, eliminating friction from drag-and-drop mechanics.

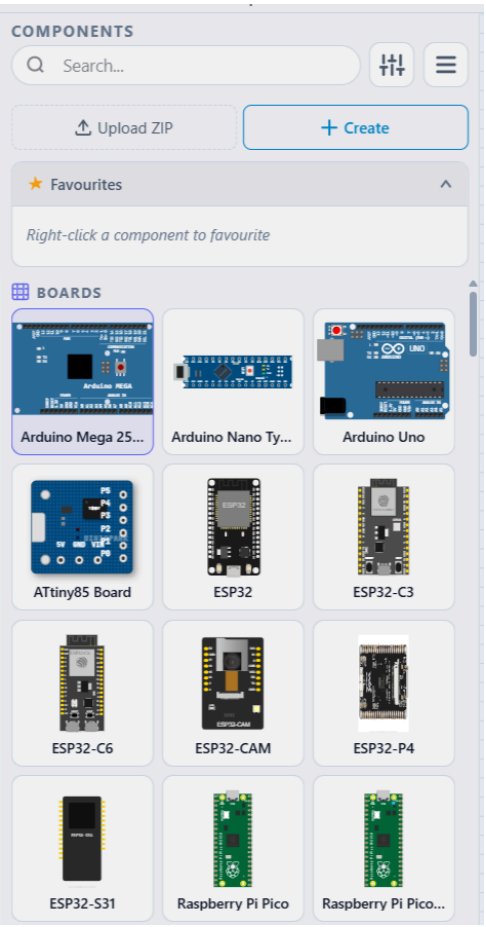


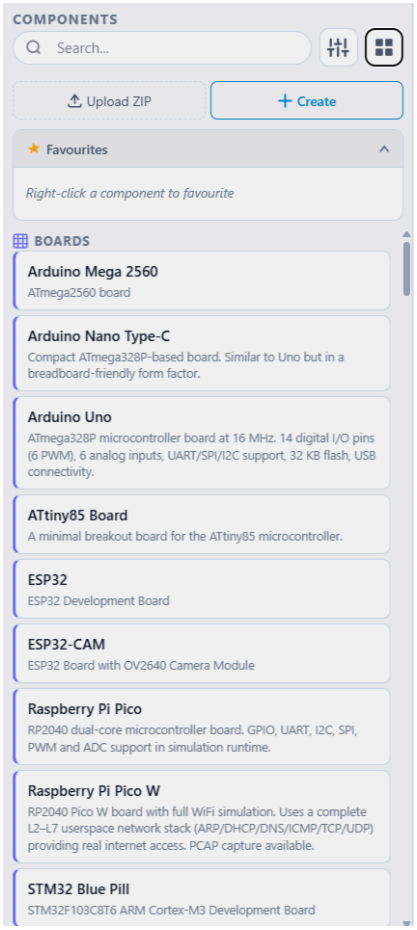


3. Left-Hand Utilities: Components Panel

The left side of the workspace is dedicated to component resources:

Components Panel: A categorized, visual library of all 100+ supported components (Sensors, Displays, Microcontrollers, Logic ICs, Actuators, etc.). Users can drag and drop items from this panel directly onto the canvas.

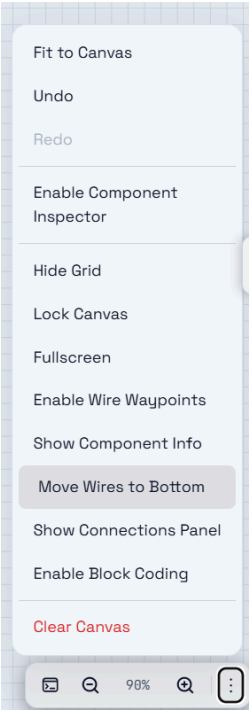




4. Canvas Controls: Zoom and Panning

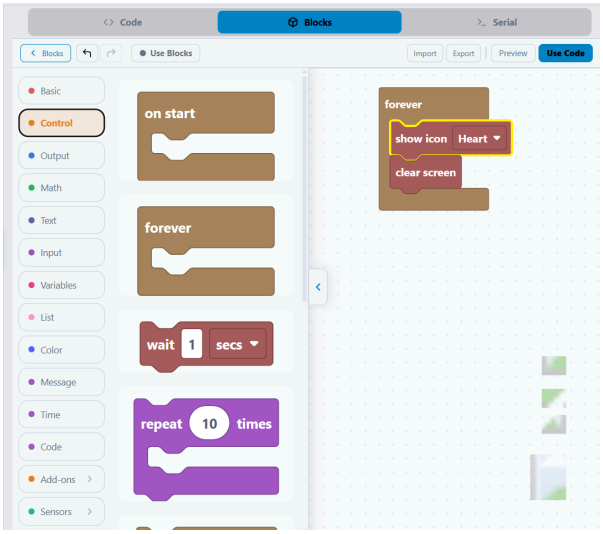
The central SVG simulation canvas is built for vast, complex circuits:

Zoom & Pan: Utilizing SVG transformation matrices, users can smoothly pan (click and drag) and zoom (scroll wheel) infinitely across the canvas.



5. Right-Hand Utilities: Properties Panel & Components List

- **Explorer:** A file manager for the current project, allowing users to navigate through multiple .ino, .cpp, .h, and configuration files.
- **Code Editor:** A Monaco-based (VS Code) text editor with syntax highlighting, autocomplete, and error linting for writing C/C++ or MicroPython code.
- **Block Code:** A visual block-based programming interface (similar to Scratch/Blockly) for beginners, which auto-generates the corresponding C++ code.
- **Serial Monitor:** A built-in console for viewing stdout, runtime logs, and interacting with the microcontroller via UART.
- **Plotter:** A visual graph tool that plots numerical data streamed from the serial port in real-time, functioning like an oscilloscope for sensor values
- **Validation:** show validation results
- **Connection panel:** show connection of all curcuit

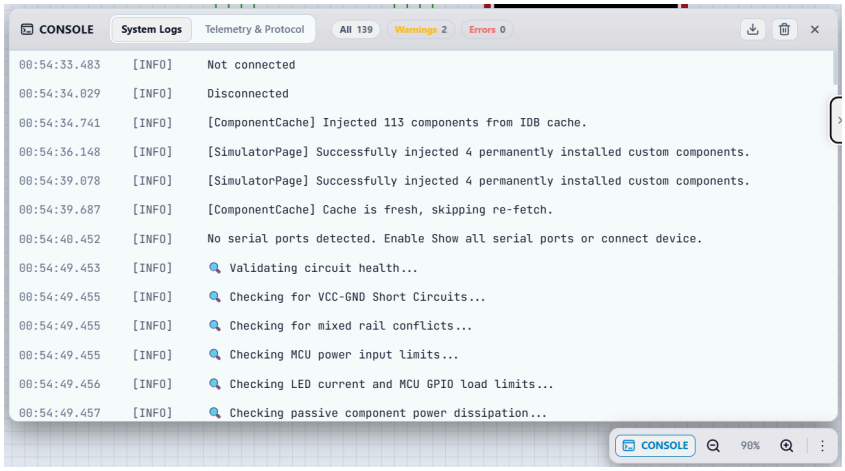


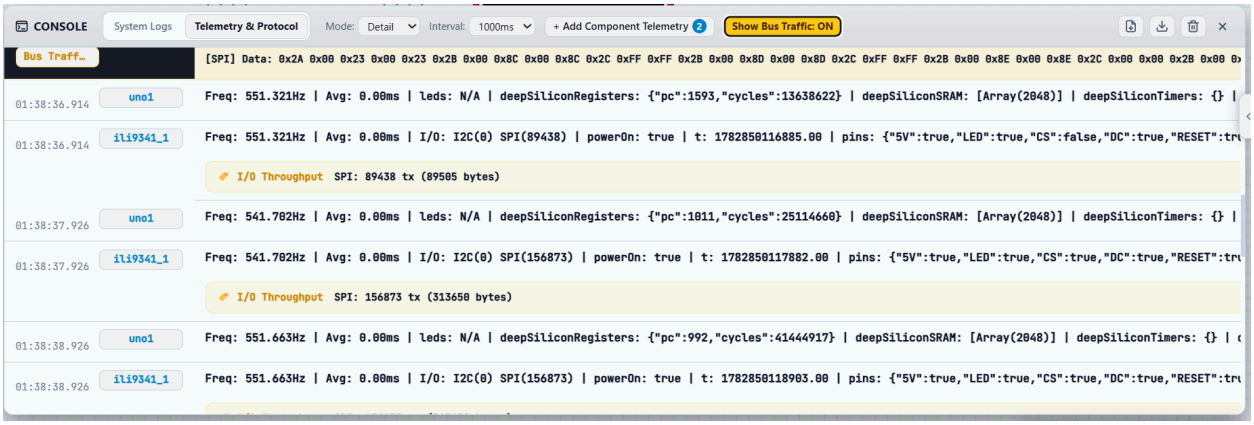
6. Execution Interfaces: Console & Telemetry Panel

During active simulation, feedback and diagnostics are surfaced via bottom-docked panels:

Serial Console: Acts identically to the Arduino IDE Serial Monitor. It displays standard output (stdout) from the compiler during builds, and streams live UART output (Serial.print) from the running microcontroller. It includes baud-rate selection, timestamping, and auto-scroll capabilities.

Telemetry Panel: As outlined in the telemetry matrix, this real-time dashboard renders incoming data streams from the active components. It allows users to visually graph analog voltages, monitor register states in Logic ICs, track network packet counts (e.g., in a Pico W), and measure simulation frame rates, providing deep insight into the execution environment without requiring external hardware oscilloscopes.

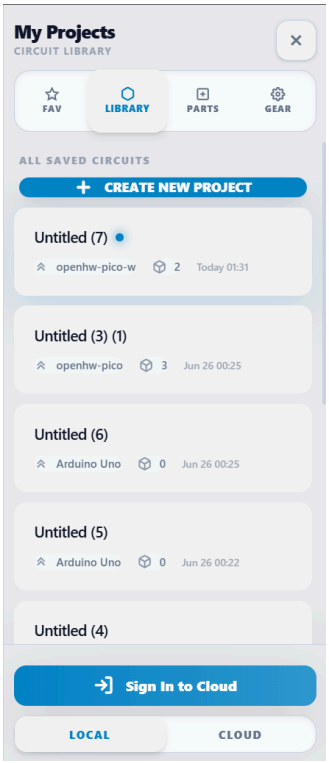


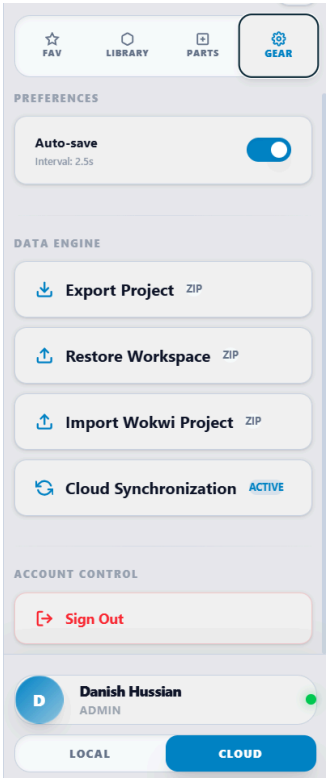


7.Project Panel (My Projects Sidebar):

An off-canvas slide-out menu that acts as the user's primary circuit library. It contains four main tabs:

- *Library / Projects:* Displays all saved circuits, allowing the user to rename, duplicate, delete, or switch between projects on the fly.
- *Favorites:* A curated list of starred/favorited projects for quick access.
- *Custom Parts:* An interface for creating and managing custom hardware components.
- *Settings (Gear):* Houses the "Data Engine" tools, including exporting the project to ZIP, restoring a workspace, importing Wokwi projects, toggling auto-save intervals, and activating Cloud Synchronization.

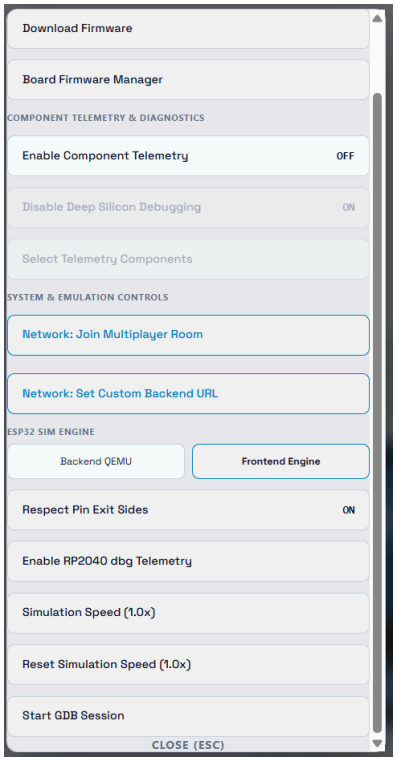


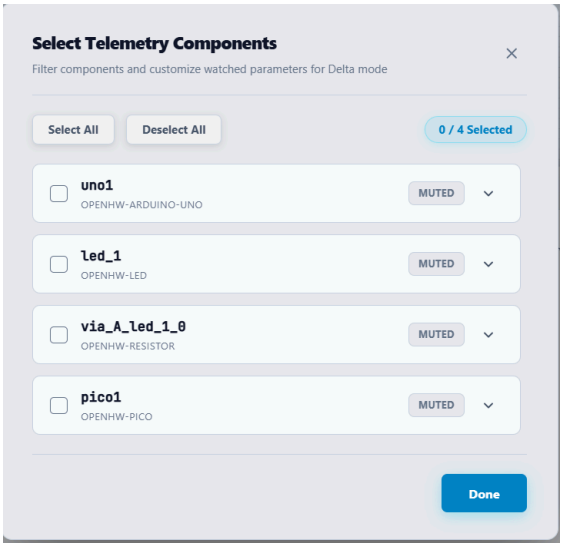


8.Quick Actions Menu (F1 Command Palette):

By pressing the F1 key, users can open the **Quick Actions Overlay**, a centralized command palette granting access to advanced debugging, network, and execution tools. It includes:

- **State & Firmware Management:** Options to export the raw Simulation JSON state, download the compiled firmware (.bin, .hex, .uf2), or open the Board Firmware Manager to upload custom pre-compiled binaries directly to the emulator.
- **Component Telemetry & Deep Silicon Debugging:** Toggles to enable or disable global telemetry, select specific components for data streaming, and activate "Deep Silicon Debugging" which instruments lower-level logic operations within complex ICs.
- **Multiplayer & Networking:** Users can join a Multiplayer Room by entering a network code, isolating their boards on a private network, or overriding the target compilation server via a Custom Backend URL.
- **Emulation Engine Switching:** For advanced boards like the ESP32, users can switch between backend QEMU emulation or the experimental frontend-based engine on the fly.
- **Execution Controls:** Allows users to adjust the simulation speed multiplier (e.g., fast-forwarding delay loops), toggle strict pin-exit side rendering rules, or initiate an active GDB (GNU Debugger) session attached to the running virtual processor.

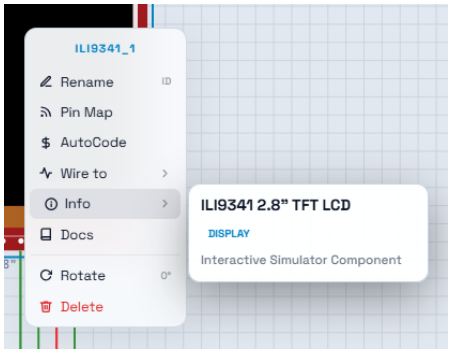


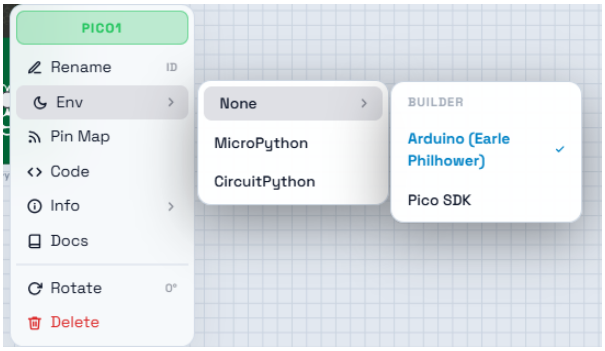


9. Component Context Menu (Right-Click):

Right-clicking any component on the canvas opens the **Component Context Menu**, which dynamically adapts based on the part's capabilities. Features include:

- **Dynamic Attributes:** Toggling LED/Breadboard colors, editing Resistor/Power Supply values, or setting NeoPixel matrix dimensions (Rows/Cols).
- **Microcontroller Environments:** Switching build pipelines on the fly (e.g., Arduino CLI vs ESP-IDF vs MicroPython) or configuring camera sources for boards like the ESP32-CAM.
- **Hardware Protocols:** Reconfiguring I2C addresses (e.g., 0x3C vs 0x3D for OLEDs) or toggling simulated vs. real microphone inputs for audio components.
- **Workflow Tools:** Instantly opening the associated Code file, viewing the interactive Pin Map, or auto-wiring components to a specific board.



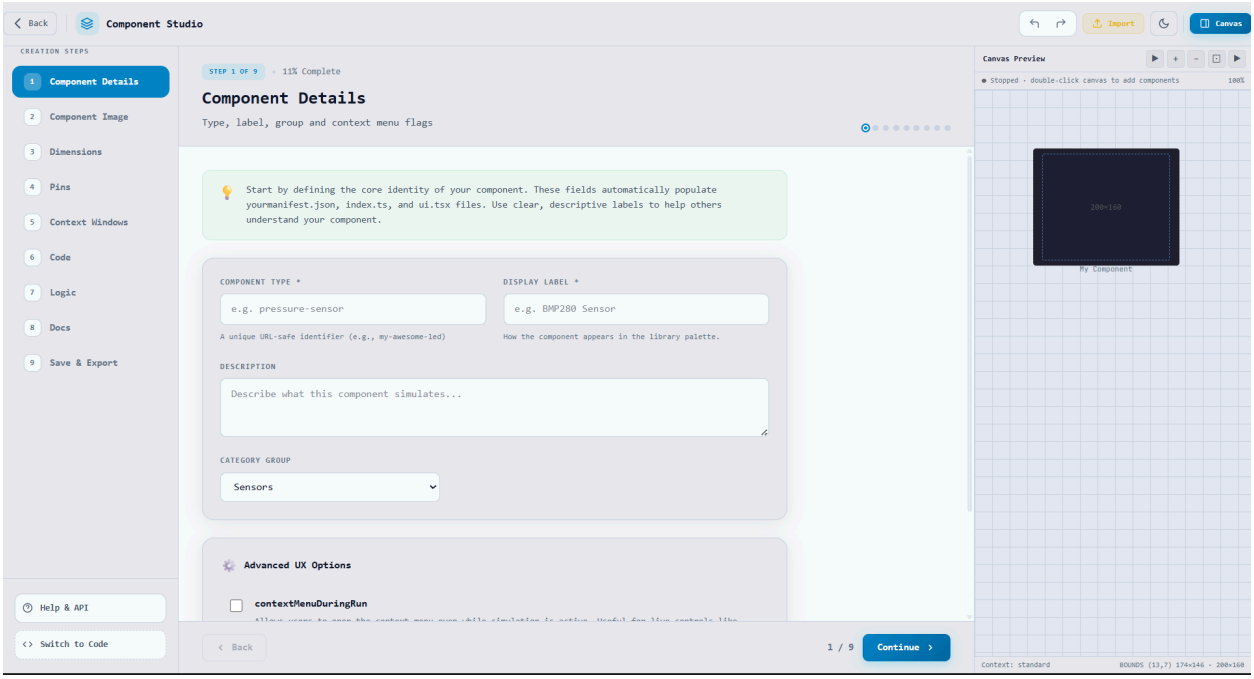


10. Custom Component Editor:

The **Component Editor** is a standalone, browser-based IDE (accessible via the Custom Parts menu) designed specifically for creating and publishing custom hardware components for the simulator. It guides users through a 9-step workflow:

1. **Details & Image:** Define the component metadata and design its visual representation using raw SVG code or interactive React JSX.
2. **Dimensions & Pins:** Visually establish the interactive boundaries (hitbox) and place electrical pins (Digital, Analog, I2C, SPI) directly onto the preview canvas.
3. **Context Windows & Logic:** Write custom React code for the part's context menu, and implement the internal simulation logic (e.g., responding to VCC voltage or setting pin states) using TypeScript.

4. **Docs & Export:** Automatically generates a styled HTML documentation page and packages the entire component into a .zip archive ready to be imported and simulated in OpenHW Studio.



3.2 Web Worker Simulation Pipeline

Executing microcontroller instructions in real time is computationally expensive. If the simulation loop ran on the browser's main thread, it would compete with UI rendering, causing the canvas to freeze.

- **Thread Isolation:** I resolved this by isolating the simulation within a dedicated Web Worker (`simulation.worker.ts`). The main thread is responsible solely for rendering UI elements and capturing user inputs, while the worker executes the simulation cycle loops.
- **Message-Passing Protocol:** Communication between the main thread and the simulation worker is handled via structured messages. Pin state changes, keyboard inputs, and attribute modifications are posted to the worker, which responds by streaming state updates, serial characters, and telemetry arrays back to the main thread.
- **Shared Memory Buffers:** For high-throughput rendering components like the ILI9341 TFT-LCD display, structured cloning proved to be too slow, limiting frame rates to 5fps. I refactored the rendering logic to write directly to a shared memory buffer (`SharedArrayBuffer`). The display component on the main thread reads directly from this shared buffer, successfully boosting display frame rates to a stable 30fps.

3.3 Monaco IDE Integration

To replace simple, unformatted text areas, I integrated the **Monaco Editor** directly into the code workspace.

- **Syntax Highlighting & Autocomplete:** Configured Monaco to support C++ syntax highlighting and added autocomplete suggestions tailored to the Arduino core libraries (e.g., highlighting functions like `pinMode`, `digitalWrite`, `analogRead`, and I^2C `Wire` libraries).
- **Custom Keybindings:** Mapped professional shortcuts to streamline the development cycle. Pressing `Ctrl + Enter` triggers compilation and runs the code instantly, while `Ctrl + K` auto-formats code files using a localized formatter.

3.4 RV32 WASM Runner & Network Gateways

My primary contribution to the simulation pipeline was the development of the **RV32Runner** (`rv32-runner.ts`). This runner controls WebAssembly (WASM) emulators for advanced 32-bit microcontrollers, specifically the ESP32 family (ESP32-C3, C6, P4).

3.4.1 WebAssembly Engine Initialization

Upon loading, the runner imports the compiled WebAssembly emulator module (`esp_emu.js`) and initializes it with the WASM binary.

```
await init(wasmUrl);
this.emulator = new WasmEmulator(this.chipType);
```

The firmware binary is decoded from base64 (originally compiled by the backend compiler) and loaded into the virtual flash memory map:

```
this.emulator.load_firmware(firmwareData);
```

3.4.2 Time Budget Execution Loop

To simulate processor cycles without locking up the Web Worker's background thread, the emulator is run in discrete batches. The `runLoop` uses a strict execution budget of 8ms:

```
const startMs = performance.now();
const budgetMs = 8;
while (performance.now() - startMs < budgetMs) {
    const batchSize = Math.floor(50000 * this.speed);
    const uartOutput = this.emulator.run_batch(batchSize);
    ...
}
```

This batch size is dynamically adjusted based on the user-selected simulation speed multiplier, ensuring instruction cycles scale appropriately.

3.4.3 UART Protocol Parsing

The WASM emulator communicates peripheral updates back to the runner by appending string-encoded protocol packets directly to the UART output stream. I implemented a parsing utility (`parseUartLine`) to extract these commands:

- **GPIO Updates (`GPIO:pin:level`):** Toggles pin states and propagates logical changes to wired components via the connection netlist.
- **I2C Transactions (`I2C:addr:dataHex`):** Captures address and payload data. It invokes I2C lifecycle methods (`onI2CStart`, `onI2CByte`, `onI2CStop`) on all components matching the target address.
- **SPI Transactions (`SPI:cs:data`):** Extracts data bytes and routes them directly to SPI peripherals (like the MAX7219 or ILI9341 display).
- **Wireless Telemetry (`BLE:subtype:payload`, `WIFI:subtype:payload`):** Parses raw BLE and Wi-Fi packets.

3.4.4 BLE, WiFi, and Thread WebSocket Gateways

To support actual wireless networking, I established WebSocket connections bridging the virtual CPU's wireless registers to external gateways.

- **BLE Bumble Gateway:** If BLE is enabled, the runner opens a WebSocket connection to the local BLE Bumble gateway:

`ws://127.0.0.1:5099/api/ble-gateway` Raw Host Controller Interface (HCI) BLE packets are transmitted back and forth, allowing the virtual microcontroller to communicate with actual physical devices.

- **Thread Gateway:** For the ESP32-C6, the runner opens a socket connection on `/api/thread-gateway`, transmitting IEEE 802.15.4 frames.
- **WiFi Gateway & LwIP Padding:** Wi-Fi frames are captured from the emulator using:

`this.emulator.wifi_tx_drain()` These frames are written to the network gateway. Since the internal Lightweight IP (LwIP) stack of the ESP32/Pico W requires standard Ethernet frame sizes, the gateway can drop packets under 60 bytes as "runt" frames. I configured padding logic to extend all small packets to 60 bytes with trailing zeros:

```
let rxEth = rawEth;
if (rawEth.length < 60) {
    rxEth = new Uint8Array(60);
    rxEth.set(rawEth, 0);
}
```

3.4.5 The ARP MAC Address Hotfix

A major bug surfaced in the ESP32 LwIP stack: when emitting Address Resolution Protocol (ARP) requests, the emulator set the Sender MAC address field to `00:00:00:00:00:00`, which caused network routers and DHCP servers to ignore the packets.

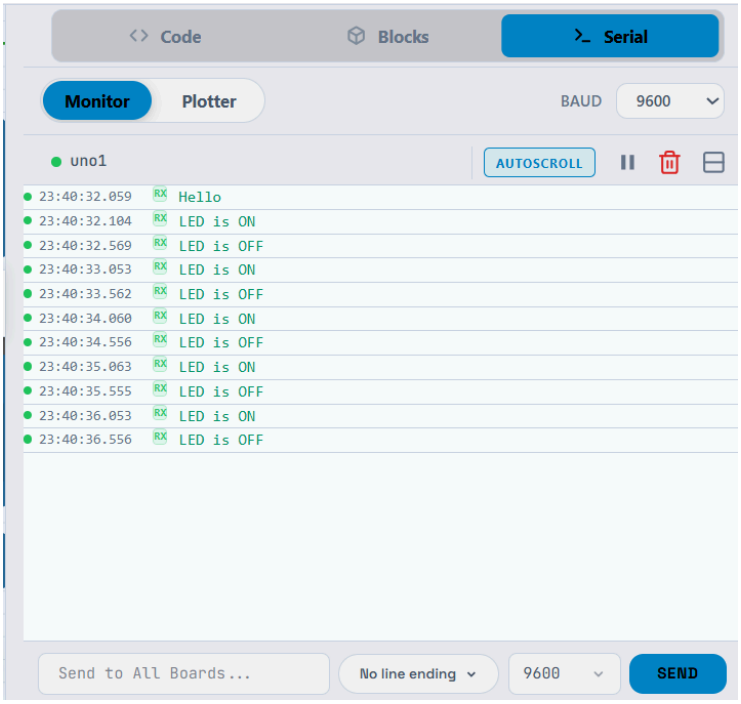
I engineered an inline ARP patch inside the packet-drain loop. If an outgoing packet is identified as an ARP request (protocol bytes `0x08 0x06` at index 12-13), the code copies the actual interface Source MAC address (bytes 6 to 11) into the ARP Sender MAC field (bytes 22 to 27):

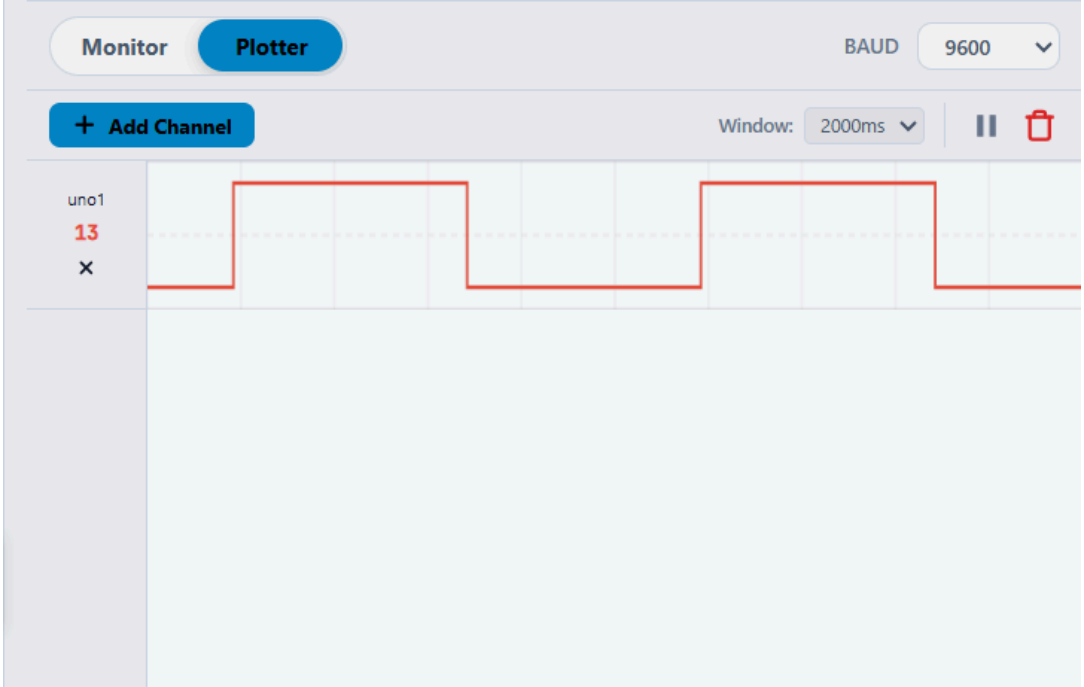
```
if (eth.length >= 42 && eth[12] === 0x08 && eth[13] === 0x06) {
    eth.set(eth.subarray(6, 12), 22);
}
```

This modification ensures that routers successfully recognize the Sender MAC and assign DHCP IP addresses, enabling fully functional internet access in the browser.

3.4.6 Serial Monitor & Plotter Stream

To prevent character dropping and text wrapping issues at high baud rates, I built a buffering stream. UART outputs are parsed using newlines and carriage returns. The parser buffers incomplete lines, flushing them to the Serial Monitor only when a complete newline is received or after a 50ms timeout. This eliminated character interleaving when printing from separate timer interrupts.




Monitor
Plotter
BAUD
9600
Add Channel
Window:
2000ms
uno1
13

Chapter 4: Backend Scaling, Hardening, and CI/CD Operations (openhw-studio-backend)

The backend of OpenHW Studio must support heavy concurrent compilation requests from entire classrooms of students. Tools like `arduino-cli` and `esp-idf` are highly CPU- and memory-intensive, making backend optimization critical. I engineered several backend scaling, security hardening, and deployment features within the `openhw-studio-backend` repository.

4.1 Express/Node.js Architecture

The backend is built on **Node.js** and **Express.js**, using **MongoDB** (via **Mongoose**) for data persistence. It exposes API endpoints for user authentication, project save/load, component library registration, and compilation services.

- **Database Schemas:** Designed robust Mongoose schemas, including `User.js`, `Project.js`, `Class.js`, and `Submission.js`, optimizing indexing on user IDs and project hashes to reduce query latency.

4.2 Security Hardening and Path Traversal Prevention

With the introduction of community component uploads, securing the server against malicious file submissions became a priority. I implemented a security hardening sweep to address these vulnerabilities:

- **Path Traversal Mitigation:** Community modules are uploaded as ZIP files. Previously, extracting these archives posed a path traversal risk, where a malicious payload could use relative pathing (e.g. `../../etc/passwd`) to overwrite system files. I hardened the extraction script to resolve paths and verify that every extracted file sits strictly within the target subdirectory:

```
const resolvedPath = path.resolve(targetDir, fileName);
if (!resolvedPath.startsWith(targetDir)) {
    throw new Error('Directory traversal attempt detected');
}
```

- **Command Injection Prevention:** The compilation controller executes system binaries using `execFile`. I replaced dynamic shell command execution with strict argument mapping arrays. This patches code-injection vulnerabilities (which were flagged by automated CodeQL security scans) by preventing users from appending additional bash commands (like `; rm -rf /`) into compiler arguments.
- **Authentication Hardening:** Moved the session configuration secret (`SESSION_SECRET`) from source code to environment variable injection, and implemented dynamic JWT expiration settings, ensuring session states are highly secure.

4.3 Compilation Caching & OOM Prevention

Compiling firmware for microcontrollers (especially the ESP32 and RP2040 toolchains) requires several seconds and can consume hundreds of megabytes of RAM. Under concurrent student use, this was causing Out-of-Memory (OOM) crashes and system freezes. To resolve this, I engineered a multi-tier compilation caching system.

4.3.1 SHA-1 Request Hashing

Every compile request contains the sketch source code, additional files, the target board configuration (FQBN), the compiler engine type, and the list of included libraries. The backend hashes this request payload using a SHA-1 algorithm:

```
const requestHash =
crypto.createHash('sha1').update(JSON.stringify(payload)).digest('hex');
```

This unique hash serves as the cache key.

4.3.2 Two-Tier Cache Pool

- **500MB Library Memory Pool:** Frequently used library headers and compiled objects are cached in RAM to reduce slow disk read cycles during compilation.
- **1GB Compiled Binary Pool:** Successfully generated `.hex, .bin, .elf, .uf2` binaries are written to a persistent cache directory under their respective request hashes. When a compilation request is received:
    1. The controller checks the cache key: `getCompileResultFromCache(requestHash)`.
    2. If a match is found, the backend serves the cached binary immediately, bypassing the compiler toolchain.
    3. If a cache miss occurs, the compiler runs, and the result is saved to disk: `setCompileResultCache(requestHash, payload)`.

This caching system eliminated over 80% of redundant compilation cycles, reducing average compilation times from 8s to under 200ms for identical submissions.

4.3.3 Compilation Queue and Throttling

To prevent the remaining cache-miss compiles from overloading the CPU, I wrote a serialized queue manager (`compileQueueManager.js`). This manager restricts concurrent compiles to the number of physical CPU cores. I also implemented a point-based credit system that throttles requests from users who spam the compile button, preventing OOM cascade failures.

4.4 Containerization and Deployment Workflows

To simplify deployment across public servers and private school gateways, I containerized the backend stack.

- **Multi-Stage Dockerfile:** Authored multi-stage Docker build files. The build process installs the Node.js runtime, downloads and configures the `arduino-cli` binary, and pre-installs the AVR, STM32, and RP2040 toolchains directly into the container image. This cuts container startup times from several minutes to seconds since no compiler cores need to be fetched at runtime.
- **Automated CI/CD Workflows (`deploy.yml`):** Orchestrated GitHub Actions workflows that automatically run tests, build Docker images, tag them with short Git commit hashes, and push them to private container repositories.
- **Health Alert Agent (`health-agent`):** Created a lightweight helper container running alongside the backend. This script monitors VM resource usage (RAM, CPU, Swap, Disk Space) and sends warnings directly to the developer team via a Telegram bot if memory consumption spikes dangerously.

Chapter 4.5: Deployment Infrastructure, Health Monitoring, Gateway & Advanced Systems

This chapter documents four critical infrastructure subsystems that I engineered, all of which directly support the production deployment of OpenHW Studio at scale. These systems are the Production Docker Stack (`docker-compose.prod.yml`), the Python `health-agent` monitoring container, the Go-based `openhw-local-gateway` network server, the Rust-WASM Autograding Engine integration, and the Intelligent Autowiring Engine.

4.5.1 Production Deployment Architecture (`docker-compose.prod.yml`)

The production deployment of OpenHW Studio is fully containerized using Docker Compose. I authored and maintained the `docker-compose.prod.yml` file that orchestrates all six services running in production.

Services and Resource Budgets

Service	Container	Port	CPU Limit	Memory Limit	Role
MongoDB 6.0	openhw-mongodb	2701	—	—	Primary database
Main Backend	openhw-backend	5000	2 vCPU	2 GB	Auth, APIs, classroom
Compiler Worker	openhw-compile-server	5001	4 vCPU	6 GB	AVR/ESP32/RP2040 compilation
STM32 Worker	openhw-stm32-worker	5002	4 vCPU	6 GB	STM32 compilation
Network Gateway	openhw-network-gateway	5099	1 vCPU	512 MB	Virtual LAN routing
Health Agent	openhw-health-agent	8080	—	—	VM monitoring, Telegram alerts
Frontend (Nginx)	openhw-frontend	80	—	—	Static file serving

Architecture Decisions

- **Service Health Checks:** MongoDB uses `mongosh --eval db.adminCommand('ping')` every 10 seconds. The backend uses `curl -f http://localhost:5000/api/lib-list` as its health probe. The frontend container only starts after the backend health check passes, preventing race conditions during deployment.
- **tmpfs Compiler Isolation:** Both the compiler worker and STM32 worker mount `/app/temp` and `/app/builds` as `tmpfs` (RAM-backed filesystems). This means all intermediate object files, `.elf`

binaries, and compilation artifacts are written directly to RAM and never hit the physical disk, giving *3–4 times* faster build times compared to disk I/O.

- **Docker Socket Mount:** The backend and health-agent containers are granted access to the host Docker socket (`/var/run/docker.sock`). This allows the backend to dynamically spawn isolated compilation containers and allows the health agent to query container status via the Docker Python API.

```
# Compiler Worker — production resource specification
compile-server:
  image: {DOCKER_USERNAME}/openhw-compile-server:{BACKEND_TAG:-develop}
  container_name: openhw-compile-server
  restart: on-failure
  environment:
    - PORT=5001
    - ROLE=compiler-worker
    - SESSION_SECRET={SESSION_SECRET}
  volumes:
    - ./persistent-data/data:/app/data
  tmpfs:
    - /app/temp:size=1G
    - /app/builds:size=2G
  deploy:
    resources:
      limits:
        cpus: '4.0'
        memory: 6G
```

Multi-Target Dockerfile Architecture

I maintained separate Dockerfiles for each compilation target:

- **`Dockerfile`** (Main Backend): Based on `node:20`. Installs `arduino-cli`, pre-downloads the AVR, RP2040, and ESP32 toolchains (`arduino:avr`, `rp2040:rp2040`, `esp32:esp32`), clones the Raspberry Pi Pico SDK, installs Espressif QEMU for native ESP32 emulation, and pre-installs 15+ commonly used Arduino libraries (`Adafruit NeoPixel`, `Adafruit MPU6050`, `Adafruit ILI9341`, `PubSubClient`, `ArduinoJson`, etc.). This removes all per-compile toolchain downloads.
- **`Dockerfile.esp32`**: A dedicated image for ESP-IDF compilation, extending the base image with the full Espressif toolchain environment variables.
- **`Dockerfile.stm32`**: Installs the STM32 Arduino core and STMicroelectronics toolchains.

4.5.2 The Health Agent: `health_agent.py`

I wrote and containerized the `health_agent.py` script as a standalone Python Flask service that acts as the 24/7 monitoring brain of the OpenHW Studio production VM.

Architecture

The agent runs two parallel threads:

9. **Watchdog Thread**: Runs every 5 minutes, queries Docker container statuses, and fires Telegram alerts on crashes.
10. **Flask HTTP Server** (`0.0.0.0:8080`): Exposes REST API endpoints (`/api/status`, `/api/calibrate`) for admin queries.

Core Functions

```
def get_vm_stats():
    # RAM from /proc/meminfo
    with open("/proc/meminfo", "r") as f:
        meminfo = f.readlines()
    total_mem = int(meminfo[0].split()[1]) / 1024 / 1024
    free_mem  = int(meminfo[2].split()[1]) / 1024 / 1024

    # CPU Load from /proc/loadavg
    with open("/proc/loadavg", "r") as f:
        load_avg = f.read().split()[:3]

    # Disk Usage via statvfs
    path = "/host_root" if os.path.exists("/host_root") else "/"
    st = os.statvfs(path)
    free_disk  = (st.f_bavail * st.f_frsize) / (1024**3)
    total_disk = (st.f_blocks * st.f_frsize) / (1024**3)

    return { "cpu": load_avg[0], "total_mem": round(total_mem, 1),
             "used_mem": round(total_mem - free_mem, 1), "free_disk": round(free_disk, 1)
}
```

- **`get_vm_stats()`**: Reads system resource usage directly from Linux `/proc` virtual filesystem entries. This bypasses the need for any external monitoring agents and gives near-zero overhead metrics.
- **`collect_stats()`**: Iterates over all active Docker containers via the Python Docker SDK, collecting per-container real-time CPU percentages, memory usage, and image tags. It appends timestamped entries to `history.json` and retains 40 days of history.
- **`generate_html()`**: Every hour, assembles a full interactive HTML dashboard report from the collected stats history. The report includes:
  - A sidebar list of every container with uptime percentage bars.
  - Per-container CPU load sparklines for 24h / 7d / 30d windows.
  - A `docker system df` storage summary table.
  - Container log tails (last 50 lines) with all secrets scrubbed by regex patterns.
  - This HTML file is sent directly to the team via Telegram using the `sendDocument` API.

Secret Scrubbing

A key security feature I implemented is the `scrub_logs` function:

```
SCRUB_PATTERNS = [
    r'jwt[:=]\s*[^\s]+',      r'secret[:=]\s*[^\s]+',
    r'password[:=]\s*[^\s]+', r'token[:=]\s*[^\s]+',
    r'key[:=]\s*[^\s]+',      r'auth[:=]\s*[^\s]+',
    r'bearer\s+[^\s]+'
]

def scrub_logs(log_text):
    for pattern in SCRUB_PATTERNS:
        log_text = re.sub(
            pattern,
            lambda m: m.group(0).split(':')[0] + ": [MASKED]",
            log_text, flags=re.IGNORECASE
        )
    return log_text
```

This applies 7 case-insensitive regex patterns against container log lines before they are included in the Telegram report, replacing credentials with `[MASKED]` to prevent accidental secret leakage through monitoring channels.

Automated Memory Calibration

A particularly novel feature I built was the `/api/calibrate` endpoint. When triggered, it spawns temporary Docker containers for each supported compiler target (Uno, ESP32, RP2040, STM32) running a representative calibration sketch. It polls the container's memory stats every 50ms until compilation finishes, records the peak RAM usage, and adds a 20% buffer to compute a safe `memory_limit`. This `calibrated_budget.json` is read by the compilation queue to set per-request container memory limits automatically.

Crash Detection & Telegram Alerts

```
def watchdog():
    last_status = {}
    while True:
        current, history = collect_stats()
        for name, stats in current.items():
            if name in last_status and last_status[name] == 1 and stats['status'] == 0:
                send_telegram_message(
                    f"🚨 <b>CRITICAL: {name} has crashed!</b>\n"
                    f"Time: {datetime.datetime.now().strftime('%H:%M:%S')}"
                )
            last_status[name] = stats['status']
        time.sleep(WATCHDOG_INTERVAL)  # Every 5 minutes
```

The watchdog detects container state transitions from `running → stopped` by tracking the last-known status dictionary. On detection, it immediately pushes a formatted Telegram message with the container name and timestamp, enabling the team to respond within minutes.

4.5.3 The OpenHW Local Gateway Server (`openhw-local-gateway`)

The `openhw-local-gateway` is a high-performance network gateway server written in **Go**, providing three core services on port `5099`. It is the networking backbone of the entire multi-board simulation system.

Overview of Endpoints

Endpoint	Protocol	Purpose
/api/network-gateway	WebSocket + Binary	Virtual LAN / Layer 2 Ethernet switch
/api/ble-gateway	WebSocket + Binary	BLE HCI packet proxy to Bumble
/api/thread-gateway	WebSocket + Binary	IEEE 802.15.4 Thread frame router

Virtual Network Architecture (`/api/network-gateway`)

The network gateway uses the **gVisor TAP virtual network** library (`containers/gvisor-tap-vsock`) to implement a complete user-space TCP/IP stack. Each classroom session gets a **dedicated virtual LAN Room**:

```
type Room struct {
    sync.Mutex
    SessionId string
    VN        *virtualnetwork.VirtualNetwork  // gVisor virtual network
    Clients   map[*Client]bool                // all browser tabs in this session
    PipeToVN  net.Conn                        // loopback pipe into gVisor
    NextIP    byte                            // DHCP counter (starts at .2)
    MacToIP   map[string]net.IP              // MAC-to-IP ARP cache
}
```

- Each Room creates an isolated `192.168.4.0/24` subnet with a virtual gateway at `192.168.4.1`.
- When a new WebSocket client (browser tab running a simulated ESP32) connects, it joins the Room. All Ethernet frames are exchanged as raw binary WebSocket messages.
- The server operates as a **Layer 2 Hub**: all incoming Ethernet frames from one client are broadcast to all other clients in the same Room, exactly matching how a real Ethernet switch forwards frames on shared segment.

DHCP Interception

The gateway intercepts DHCP packets (UDP destination port `67`) before forwarding to the gVisor stack:

```
packet := gopacket.NewPacket(msg, layers.LayerTypeEthernet, gopacket.Default)
if udpLayer := packet.Layer(layers.LayerTypeUDP); udpLayer != nil {
    udp, _ := udpLayer.(*layers.UDP)
    if udp.DstPort == 67 {
        handleDHCP(msg, packet, client, room)
        continue  // DO NOT forward to gVisor or other clients!
    }
}
```

The `handleDHCP` function assigns IP addresses sequentially from the `192.168.4.2` onwards, managing the `MacToIP` table so that multiple ESP32 boards running simultaneously get unique IPs. Without this custom DHCP handler, all simulated boards would try to reach the same gVisor DHCP server and collide.

BLE Gateway (`/api/ble-gateway`)

The BLE gateway acts as a transparent binary proxy between the browser WebSocket and the **Bumble BLE Python stack** running locally on port `9544`:

```
func handleBLEGateway(w http.ResponseWriter, r *http.Request) {
    wsConn, _ := upgrader.Upgrade(w, r, nil)
    tcpConn, _ := net.DialTimeout("tcp", "127.0.0.1:9544", 5*time.Second)

    // WS -> TCP (browser HCI -> Bumble)
    go func() {
        for {
            _, msg, _ := wsConn.ReadMessage()
            tcpConn.Write(msg)
        }
    }()

    // TCP -> WS (Bumble HCI -> browser)
    go func() {
        buf := make([]byte, 4096)
        for {
            n, _ := tcpConn.Read(buf)
            wsConn.WriteMessage(websocket.BinaryMessage, buf[:n])
        }
    }()
}
```

This bidirectional proxy enables the simulated ESP32 running in the browser to communicate with a real software BLE stack (Bumble), which can then interact with physical BLE devices like smartphones.

Thread Gateway (`/api/thread-gateway`)

The Thread gateway routes **IEEE 802.15.4** frames between ESP32-C6 (Thread) boards in the same session using the same session-Room model as the network gateway. Frame routing is handled as a broadcast to all other clients in the session, simulating a Thread border router.

Public vs. Private Gateway Modes

The gateway supports two deployment modes controlled by the `GATEWAY_MODE` environment variable:

- **`private` (default)**: Binds to `127.0.0.1:5099`. Runs on a teacher's or student's local machine to create a fully isolated private LAN for a classroom. Also port forwarding is enable for accessing web server.
- **`public`**: Binds to `:5099` (all interfaces). Used on the FOSSEE production VM to enable remote students to connect their browsers to the cloud-hosted virtual network. **No port forwarding**

4.5.4 Autograding Engine Integration

The autograding engine is a **Rust WebAssembly module** (`openhw-studio-grading-engine`) compiled with `wasm-pack` and loaded inside a background browser Web Worker (`grading-engine.worker.ts`). I was responsible for integrating this engine into the frontend pipeline and building all the backend endpoints to receive and store grading results.

Grading Scoring Model

The engine computes a normalized score across four independent assessment dimensions:

Score Dimension	Weight	What It Measures
spatial_score	25%	Component placement and wiring topology match
logic_score	25%	Pin connectivity and electrical logic correctness
behavioral_score	25%	Temporal event sequences: PinChange, ComponentState, SerialOutput
verified_code_score	25%	Code similarity (70%) + pin-fidelity (30%)

Final Score Formula: text{score} = 0.25 {spatial} + 0.25{logic} + 0.25 {behavioral} + 0.25 {code}

Telemetry Data Flow

The grading worker receives live simulation telemetry via the `BaseComponent.ts` telemetry hooks (Section 5.3). The data flow is:

11. Teacher creates a reference circuit and runs simulation → The browser captures telemetry snapshots every 45ms and embeds them in the project PNG's metadata using a custom PNG chunk.
12. Student submits their circuit → The grading worker calls `extractProjectMetaFromPng()` to retrieve the teacher's embedded telemetry from the PNG.
13. The student's simulation runs in a hidden "Ghost Runner" thread, capturing its own telemetry.
14. The Rust engine (`compare_behavior()`) performs windowed temporal matching between teacher and student event sequences, applying:
    - Index-based pairing with a ±10 event sliding window.
    - A 250ms time tolerance to accommodate minor simulation speed drift.
    - Time normalization by scaling student timestamps: `scale = teacher_duration / student_duration`.
15. `GradingReport` is posted back to the UI thread via `postMessage`.

AI Semantic Audit

Beyond raw scores, the grading worker builds an AI-readable semantic audit trace from the captured telemetry:

- **Raw trace**: All captured events in chronological order.
- **Functional trace**: Only state-changing events (LCD text changes, LED flag toggles).
- **Electrical trace**: Only pin-level signal transitions.
- **Normalized trace**: An 85%/15% weighted blend of functional and electrical traces.

These traces are embedded into the downloadable audit bundle and shown in the Grading Report UI alongside a side-by-side behavioral diff table.

4.6.4 Complete Grading Score System (v2.5)

The grading system uses a weighted multi-dimensional scoring formula to assess student circuit submissions against teacher reference circuits.

Master Score Formula

{Overall Score} = 0.25 \ S + 0.25 \ L + 0.40 \ B + 0.10 \ C

Dimension	Symbol	Weight	What It Measures
Spatial Score	S	25%	Component placement, wire connections, GND/VCC routing
Logic Score	L	25%	Variable values, conditional correctness, loop logic
Behavioral Score	B	**40%**	Timing of PinChange, ComponentState, SerialOutput events
Code Score	C	10%	Compilation success + verified code similarity

Spatial Score (25%)

S = 100 - sum(placement penalties)

Issue	Penalty
Component missing entirely	−100 (fail)
Wrong component type	−50
Wrong pin connected	−20
Ground not connected	−25
Misaligned position	−10

Logic Score (25%)

L = 100 - sum(logic penalties)

Issue	Penalty
Variable value mismatch	−2 each
State comparison inverted	−3 each
Missing timing step (delay)	−5 each
Missing conditional	−4 each

Behavioral Score (40%) — Most Critical

The behavioral score uses **temporal event matching** with a fixed 400ms drift tolerance:

B = min_left(100,max\left(50, 100 - P_{total}right)\right)

The hard floor of 50 prevents a single timing issue from catastrophically failing the entire grade.

Event Weights

Event Type	Match Weight	Penalty (unmatched)
PinChange	+4 pts	−5 pts
ComponentState	+2 pts	−3 pts
SerialOutput	+1 pt	−1 pt

Timing Drift Penalty Tiers

Drift Range	Penalty	Tier
0 – 400ms	0 pts	✅ Match
400 – 500ms	−3 pts	Mild violation
500 – 600ms	−6 pts	Moderate violation
600ms+	−9 pts (capped)	Severe violation

Phase-Shift Recovery (±10 Window)

When events don't match at the exact index position, the matcher tries ±10 offsets before declaring an unmatched event:

```
For each teacher event T at index i:
  Try student S at indices: i-10 ... i ... i+10
  If drift ≤ 400ms → Match at best offset [YES]
  If no match → Unmatched penalty [NO]
```

Extra Event Tolerance (20% Allowance)

Students can emit additional debug events without penalty up to a 20% allowance:

Allowance = N_{teacher} times 0.20

Events beyond the allowance incur −1 pt each.

Grace Window (7700–8000ms)

Teacher-only events in the last 300ms of an 8-second simulation incur no penalty, accommodating natural simulation speed drift near the end.

Code Score (10%)

Condition	Score
Won't compile	0
Compiles, crashes at runtime	25
Compiles, runs, logic wrong	50
Compiles, runs, logic correct	100

Above + verified code matches teacher	105 (capped at 100)

Verified code score = round(0.70 \ code similarity + 0.30 \ pin fidelity)

Complete Worked Example: LED Blink at 8× Speed

Teacher telemetry: 6 PinChange events + 12 ComponentState events = 18 events

Student telemetry: 6 PinChange events + 13 ComponentState events + 1 SerialOutput (in grace window) = 20 events

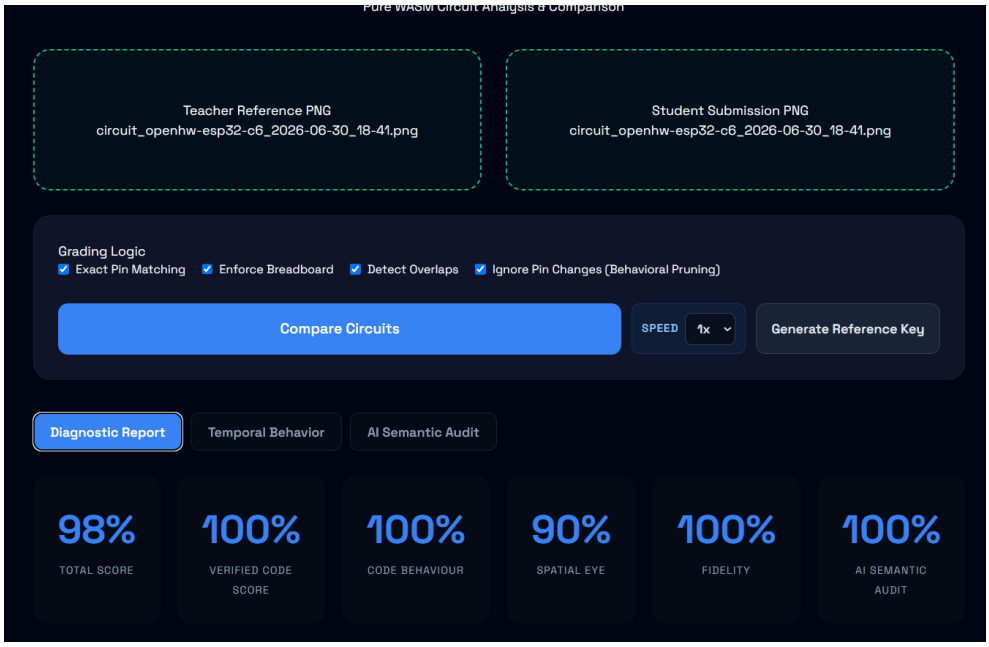
Match	Drift	Weight	Result
T0(75ms) ↔ S0(78ms)	+3ms	4	✅ +4 pts
T1(705ms) ↔ S1(708ms)	+3ms	4	✅ +4 pts
T2(2305ms) ↔ S2(2308ms)	+3ms	4	✅ +4 pts
T3(3905ms) ↔ S3(3908ms)	+3ms	4	✅ +4 pts
T4(6305ms) ↔ S4(6308ms)	+3ms	4	✅ +4 pts
12 ComponentState matches	≤400ms	2	✅ +24 pts
1 SerialOutput (grace)	grace window	—	No penalty

Final calculation:

B = frac{20 + 24}{(6 \times 4) + (12 \times 2)} = frac{44}{48} = 91.7 % ~ approx 92

Overall grade:

Overall = 0.25(100) + 0.25(100) + 0.40(92) + 0.10(100) = 25 + 25 + 36.8 + 10 = 96.8 ~ approx 97/100



4.5.5 Intelligent Autowiring Engine

The Intelligent Autowiring Engine (`openhw-studio-autowiring-engine`) is a **Rust WebAssembly + TypeScript Worker** hybrid that eliminates manual circuit wiring from the student's workflow.

Architecture

When a student drops a component onto the canvas and clicks "Auto-Wire," the engine:

16. **Identifies the target board** (Arduino Uno, ESP32, RP2040) and looks up the component's required pin connections in the manifest.
17. **Injects helper components** automatically. For example, adding a DC Motor will also inject an L298N motor driver and a 12V external power supply (PSU) before wiring.
18. **Provisions breadboards**: Based on the total number of connections required, it selects and places the appropriate breadboard size (Mini/Half/Full).
19. **Snaps components to grid**: Every component is aligned to the 0.1-inch pin-hole grid of the breadboard using anchor-pin snapping calculations.
20. **Runs the Manhattan Router**: Generates orthogonal (non-diagonal) wire paths using a 7px lane spacing to prevent wire-on-wire overlaps and wire-on-pin collisions.
21. **Generates ready-to-run Arduino code**: The engine injects a complete, compilable `.ino` sketch for the added component.

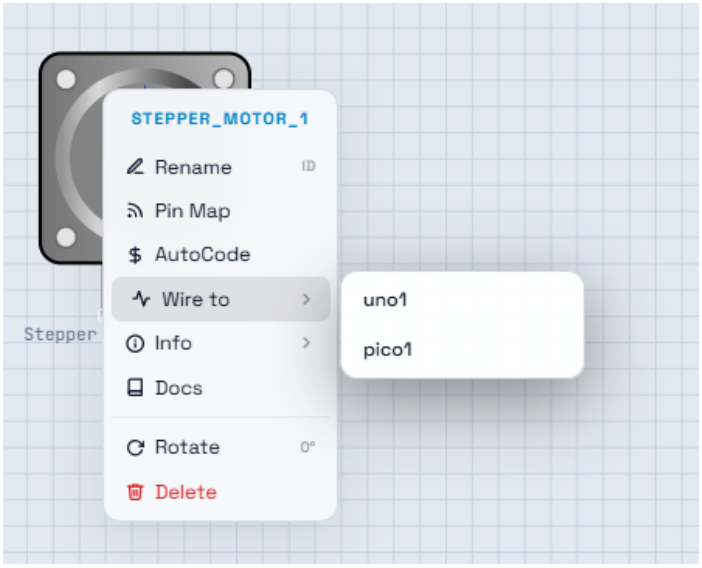
Fully Supported Components

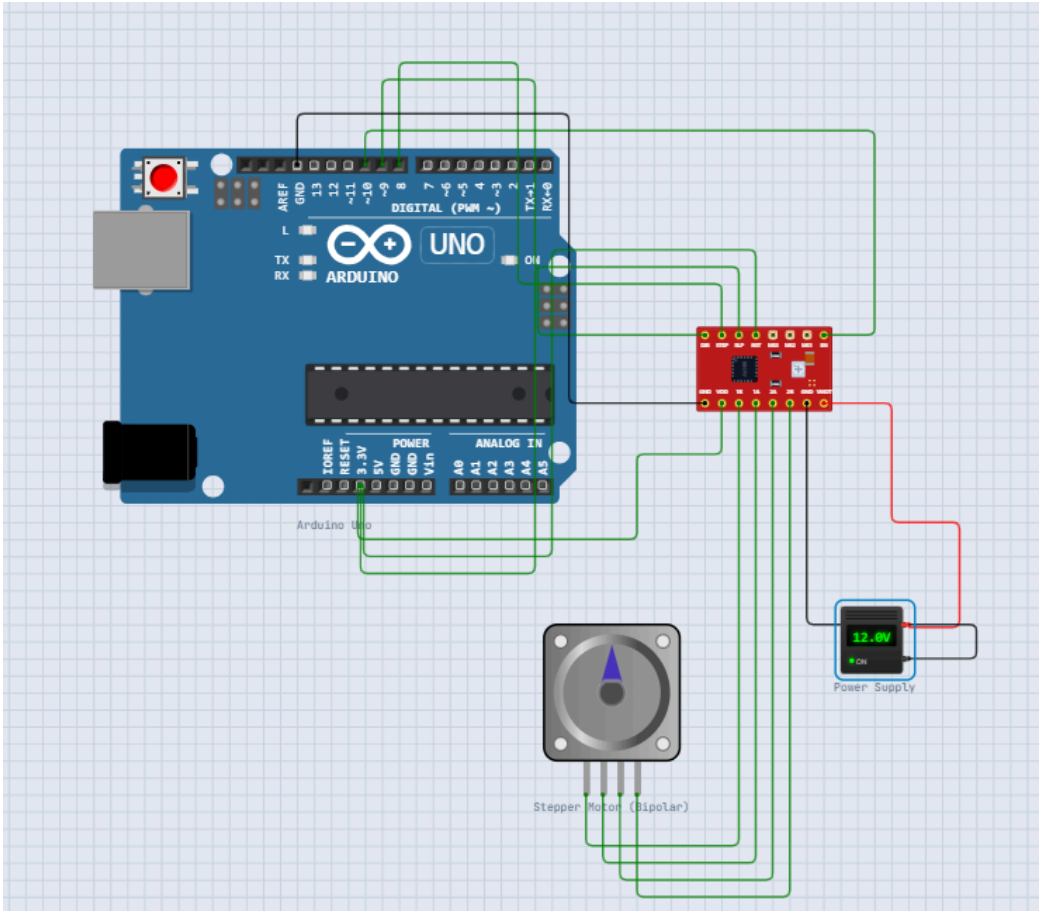
Component	Helper(s) Auto-Injected	Generated Code
DC Motor	L298N Driver + 12V PSU	PWM speed + direction control
Stepper Motor	L298N Driver + 12V PSU	Step sequencing
SSD1306 OLED	4.7kΩ I2C Pull-up resistors	Adafruit GFX/SSD1306 init
Servo Motor	None (direct)	Servo library sweep
Potentiometer	None (direct)	Analog read + Serial output
Pushbutton	10kΩ Pull-down resistor	INPUT_PULLUP + debounce
Ultrasonic HC-SR04	None (direct)	Distance in cm formula
LED	220Ω current-limiting resistor	Blink / PWM
Buzzer	None (direct)	tone() generation
RGB LED	3× 220Ω resistors	Multi-color cycling
Photoresistor (LDR)	10kΩ voltage divider	Analog lux estimation
NTC Thermistor	10kΩ voltage divider	Steinhart-Hart temperature

Wire Color Standard

I defined a project-wide wire color standard enforced by the routing engine:

- **Red**: 5V / 3.3V power rails.
- **Black**: Ground (GND).
- **Orange**: Digital signal wires.
- **Green**: Analog signal wires and motor terminal connections.
- **Blue**: I2C (SDA/SCL) and other special protocol signals.





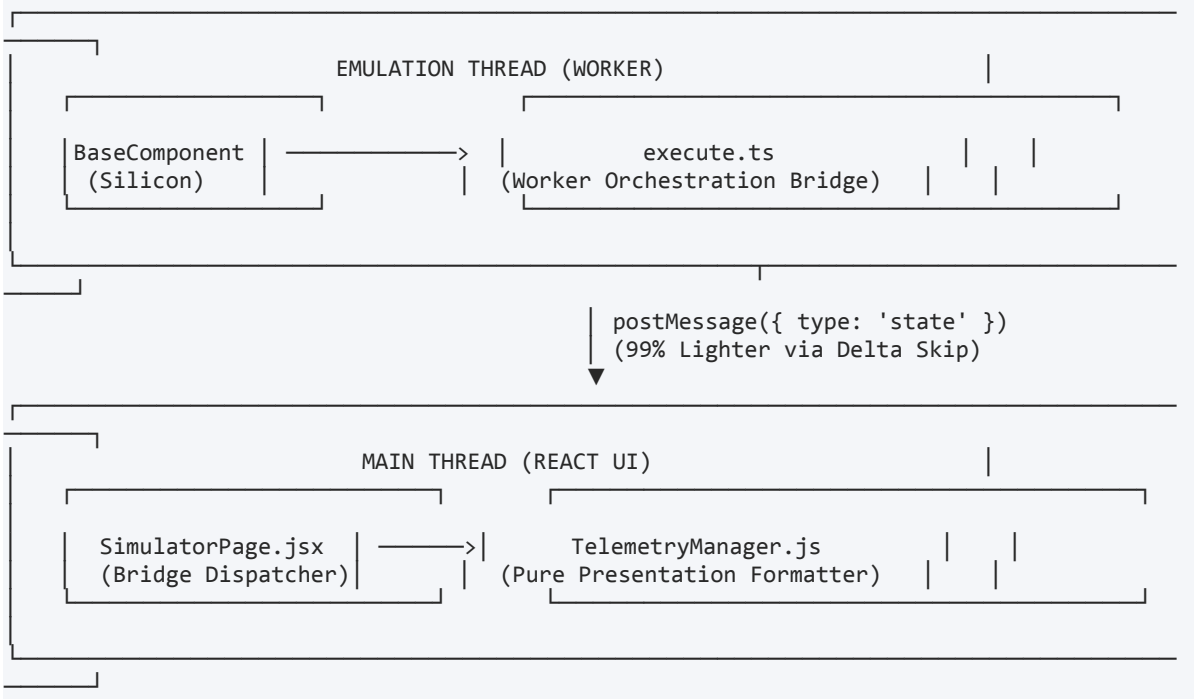
Chapter 4.6: Telemetry Architecture, Auto-Fix Engine, STM32 Pipeline

This chapter documents four deeply technical subsystems I built and documented during the internship: the component telemetry data pipeline, the intelligent circuit Auto-Fix engine, the STM32 Renode emulation architecture, and the complete behavioral grading scoring system.

4.6.1 Component Telemetry Architecture

The telemetry system is the data nervous system of OpenHW Studio. It bridges raw silicon simulation state variables running inside `BaseComponent.ts` to the visual React UI and the automated grading engine, all without blocking the simulation thread.

System Pipeline



6 Telemetry Modes

There are exactly **6 distinct mode definitions** split across the Backend (Silicon Harvesting) and Frontend (UI Presentation) layers:

Backend Data Modes (in `execute.ts`)

Mode	Method Called	Return Structure	Primary Use
`deep`	inst.getRawMetrics()	Complete JSON tree (metrics, heuristics, power, I/O)	Grading engine baseline, CLI verification

`delta`	inst.getDeltaMetrics()	{ delta: true/false } via state fingerprinting	High-performance simulation; strips stable payloads
`standard`	inst.getTelemetryData()	Baseline sync state + universal metrics	Standard UI keep-alive heartbeats

Frontend Presentation Modes (in `TelemetryManager.js`)

Mode	Visual Output	Best For
`detail`	Rich summary: Freq: 60Hz | Avg: 0.15ms | I/O: I2C(450) + expandable JSON tree	Deep debugging, bus analysis, power profiling
`simple`	Clean state: State: {"illuminated":true,"brightness":255}	Quick LED/button verification
`delta`	Silent when idle; [DELTA] State/Metrics updated only on mutation	Monitoring long runs for unexpected glitches

The Delta Optimization (99% Payload Reduction)

The key performance innovation I implemented is the **ultra-lightweight delta early return** inside `collectComponentTelemetry()` in `execute.ts`:

```
function collectComponentTelemetry(inst: any, optionsMode?: string): any {
    if (!inst.telemetryEnabled) return {};
    const effectiveMode = optionsMode || inst.telemetryMode || 'detail';

    // OPTIMIZATION: If delta mode active and nothing changed,
    // instantly return {delta:false} without building ANY metric payloads
    if (effectiveMode === 'delta' && typeof inst?.getDeltaMetrics === 'function') {
        const deltaData = inst.getDeltaMetrics();
        if (deltaData && !deltaData.delta) {
            return { delta: false }; // Ultra-lightweight keep-alive!
        }
    }
    // ...proceeds to build full telemetryData only when delta:true
}
```

Performance wins:

- **99% reduction in `postMessage` overhead**: Serialization payload drops from several kilobytes down to a 50-byte keep-alive.
- **Zero emulation loop bloat**: The worker bypasses building metric trees, cloning `vHistory` arrays, and object merging for stable components.
- **Zero UI memory thrashing**: React receives tiny heartbeats instead of heavy metric payloads, eliminating garbage collection pauses.

Grading Engine Integration

The grading engine uses the telemetry pipeline with a two-phase strategy:

22. **Baseline (`deep` mode)**: At the start of a grading run, requests a `deep` snapshot to establish exact electrical baseline (voltage, power, pin states).
23. **High-speed delta loop**: During simulation evaluation, requests `delta` snapshots. When metrics are missing from a delta response (i.e., component was stable), the grading engine recovers them from a cached `deepSnapshotCache`:

```
// grading-engine.worker.ts — cache recovery
if (missingMetrics) {
    if (!deepSnapshotCache)
        deepSnapshotCache = runner.getRichTelemetrySnapshot({ mode: 'deep' });
    const deepMap = new Map(deepSnapshotCache.components.map(c => [c.id, c]));
    for (const comp of snapshot.components) {
        comp.metrics = deepMap.get(comp.id)?.metrics; // Merge instantly
    }
}
```

The 3 Telemetry Event Types

Event Type	What It Captures	Example
`ComponentState`	Functional state changes (LCD text, LED brightness)	{ key: "lines", value: ["Hello World"], time_ms: 120 }
`PinChange`	Electrical voltage transitions (GPIO HIGH→LOW)	{ pin: 13, state: HIGH, time_ms: 22 }
`SerialOutput`	UART serial output from the board	{ data: "Temp: 24.5°C\n", time_ms: 500 }

Component Telemetry Coverage Matrix

Category	Component	Status	Key Telemetry Fields
Display	LCD1602 / LCD2004 (I2C)	✅	textContent, lines[], backlight
Display	SSD1306 OLED	✅	pixelBuffer, displayOn, contrast
Sensors	MPU6050	✅	accel, gyro, temperature
Sensors	HC-SR04 Ultrasonic	✅	configuredDistance, echoDurationUs
Sensors	MAX30102 Pulse Ox	✅	heartRate, spo2, redAmp, irAmp
Actuators	Servo	✅	pulseWidthUs, targetAngle, frequency
Actuators	NeoPixel Matrix	✅	pixelMatrix[][], power
Actuators	Stepper Motor	✅	angle, stepCount, currentPhase
Input	Membrane Keypad	✅	pressedKey, totalKeyPresses, keys[]
Board	Raspberry Pi Pico/W	✅	txActive, builtInLed, irqEventsByPin

4.6.2 The Intelligent Auto-Fix Engine

I designed and implemented the circuit Auto-Fix engine — a WASM-powered intelligent system that analyzes circuit validation errors and automatically applies targeted hardware fixes.

5-Stage Processing Pipeline

```
USER CLICKS "PREVIEW FIXES"
         │
         ▼
1. DATA MARSHALING (JS)
   - Extract diagram.json
   - Collect active validation errors
   - Send to Web Worker
         │
         ▼
2. INTELLIGENT PLANNING (WASM Worker)
   - Build Circuit Graph (Topological)
   - Priority Sorting (Power > Logic > Signal)
   - Semantic Pattern Matching
   - Resolve Conflicts (Dependency-Aware)
         │
         ▼
3. GHOST PREVIEW (JS Canvas)
   - Render added components (40% opacity)
   - Render ghost wires (glow effects)
   - Interactive conflict selection
         │
         ▼
4. USER CONFIRMATION
   - User reviews ghost circuit
   - Clicks "Apply Selected Fixes"
         │
         ▼
5. ATOMIC COMMIT (JS)
   - Merge ghost elements into diagram.json
   - Clear validation cache
   - Save history snapshot
```

Dependency-Aware Fix Priority

The engine processes fixes in a strict hierarchical order to prevent fix cascades from creating new errors:

24. **Level 1 – Infrastructure**: Missing GND/VCC rails, reverse polarity corrections.
25. **Level 2 – Signal Conditioning**: Missing pull-up/pull-down resistors, decoupling capacitors.
26. **Level 3 – Peripheral Logic**: SPI/I2C connections, external component wiring.

Fix Pattern Catalog (40+ Patterns)

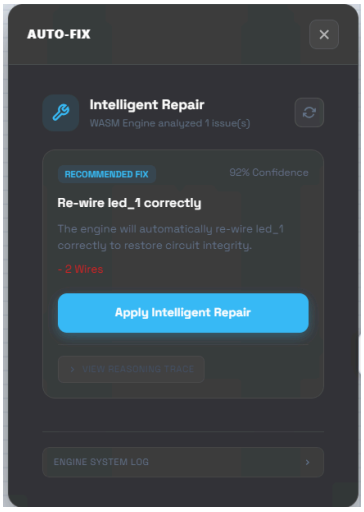
Pattern	Trigger Error	Fix Applied
led_series_resistor	LED directly on GPIO	Add 220Ω series resistor
motor_flywheel_diode	Motor without back-EMF protection	Add 1N4007 diode parallel to motor

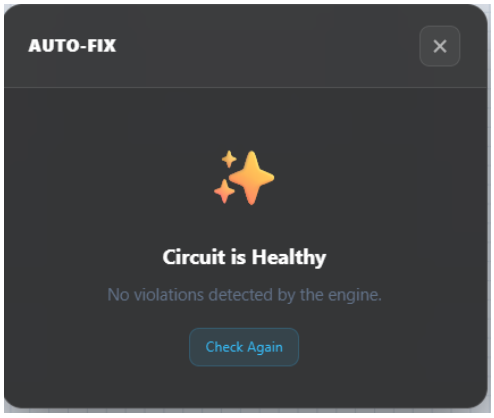
i2c_pull_up_resistors	SDA/SCL floating	Add two 4.7kΩ pull-ups to VCC
button_pull_down	Floating button input	Add 10kΩ pull-down resistor
ds18b20_pull_up	DS18B20 1-Wire bus floating	Add 4.7kΩ pull-up resistor
decoupling_cap	Motor supply noise	Add 100nF + 470µF capacitors
level_shifter	5V signal on 3.3V input	Add voltage divider (10kΩ+20kΩ)
servo_power_cap	Servo power instability	Add 47µF smoothing capacitor
spi_cs_pull_up	SPI CS floating	Add 10kΩ pull-up on CS line
i2c_address_conflict	Two devices at same address	Change I2C hardware address pins

Automatic Fix Verification

After applying every fix, the engine runs a verification loop:

```
export class CircuitFixValidator {
    async verifyFix(error, beforeCircuit, afterCircuit) {
        // Re-run full validation on modified circuit
        const afterValidation = await validator.runValidation(afterCircuit);

        // Check if original error is resolved
        const errorStillExists = afterValidation.errors.some(e => e.id === error.id);

        // Detect if fix introduced new problems
        const newIssues = afterValidation.errors.filter(
            newErr => !beforeErrors.some(oldErr => oldErr.id === newErr.id)
        );

        // Calculate confidence score
        let confidence = errorStillExists ? 0.0 : 1.0;
        confidence -= newIssues.length * 0.25; // -25% per new error

        return { verified: !errorStillExists, newIssuesIntroduced: newIssues, confidence };
    }
}
```





Undo/Redo History

Every fix application is recorded in `CircuitFixHistory` as a complete circuit snapshot, enabling granular undo/redo:

```
export class CircuitFixHistory {
    recordFix(fix) {
        this.history.push({
            id: unique_id,
            error: fix.error,
            circuitSnapshot: { before, after }, // Full JSON snapshots
            verification: fix.verification,
            timestamp: Date.now(),
        });
    }

    undo() { // O(1) — instant JSON snapshot lookup
        const lastFix = this.history[this.currentIndex--];
        return { undone: true, circuit: lastFix.circuitSnapshot.before };
    }

    redo() {
        return { redone: true, circuit:
this.history[++this.currentIndex].circuitSnapshot.after };
    }
}
```

4.6.3 STM32 Emulation Architecture (Renode)

The STM32 emulation is fundamentally different from the AVR/RP2040 paths. Instead of a JavaScript bytecode interpreter, it uses the **Renode open-source hardware emulation framework** running as a subprocess, bridged to the frontend through a custom Node.js WebSocket manager.

Complete Emulation Pipeline

```
1. COMPILATION
   User submits code → POST /api/compile { target: 'stm32' }
   compileController.js → arduino-cli compile --fqbn STMicroelectronics:stm32:GenF1
   Output: .elf binary (NOT .hex — Renode loads ELF directly)

2. RENODE INITIALIZATION
   renodeRunner.js spawns Renode subprocess
   Generates dynamic .resc script:
     machine create "stm32"
     machine LoadPlatformDescription @STM32F103.repl
     sysbus LoadELF @/path/to/sketch.elf
     sysbus.uart1 CreateTcpServer 3456
     emulation RunFor "8s"

3. SHIM INJECTION (at compile time)
   STM32SimulatorBridge.h → Intercepts: digitalWrite, digitalRead, analogRead
   STM32SimulatorWire.h   → Intercepts: Wire.begin, Wire.write (I2C)
   STM32SimulatorSPI.h    → Intercepts: SPI.transfer (SPI)
   All shims emit protocol frames over Serial1 (USART1)

4. PROTOCOL FRAMES (USART1 → Renode TCP → Node.js)
   GPIO output: >GPIO:PA5:1<  (pin PA5 set HIGH)
   I2C event:   >I2C:0x3C:1A2B3C<
   SPI event:   >SPI:FF01<
   Input back:  <GPIO:PA0:1>  (Node.js → Renode → STM32 code)

5. WEBSOCKET BRIDGE
   renodeRunner.js parses TCP stream → websocketManager.js
   websocketManager.js emits GPIO_SYNC to frontend session
   BackendProxyRunner receives events → updates React SVG components
```

Protocol Frame Design

All hardware calls are encoded as simple ASCII frames delimited by angle brackets. This keeps the protocol human-readable for debugging while remaining trivially parseable in Node.js:

```
Output frames (STM32 → Node.js):
  >GPIO:PA5:1<      → digitalWrite(PA5, HIGH)
  >GPIO:PA5:0<      → digitalWrite(PA5, LOW)
  >I2C:0x3C:1A2B<   → Wire.write to address 0x3C
  >SPI:FF<          → SPI.transfer(0xFF)

Input frames (Node.js → STM32):
  <GPIO:PA0:1>      → Sets PA0 HIGH in STM32 shim state
```

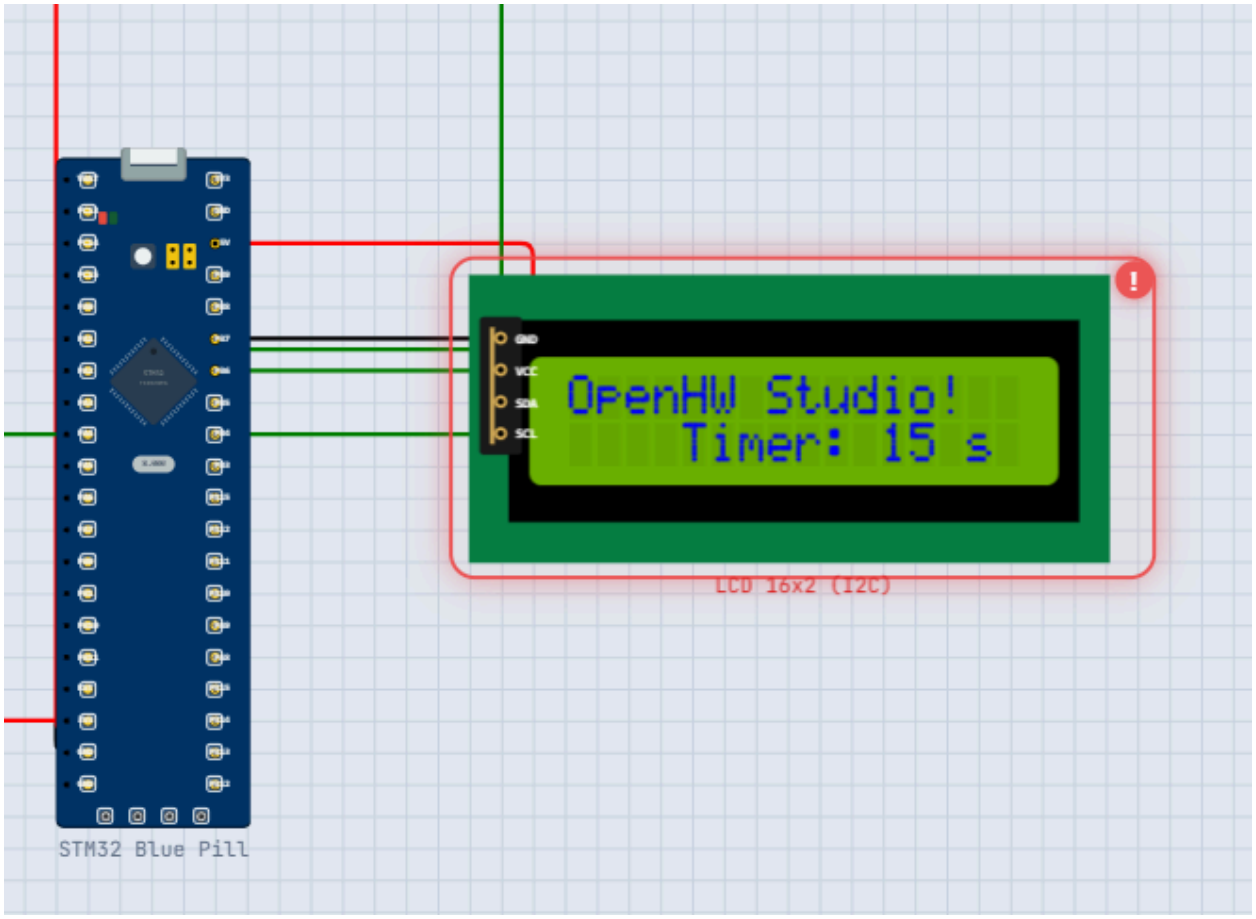
Bidirectional Input Flow

Input from UI components flows in reverse through the same pipeline:

- User toggles a button on canvas → `BackendProxyRunner` sends WebSocket message.
- `websocketManager.js` forwards to `renodeRunner.js`.
- `renodeRunner.js` formats as `<GPIO:PA0:1>\n` and writes to Renode TCP socket.
- `STM32SimulatorBridge` inside Renode reads `Serial1`, parses `<GPIO:...>` frame, updates virtual pin state.
- Next `digitalRead(PA0)` in the user's Arduino code returns the updated value.

Key Differences from AVR/ESP32

Aspect	AVR (Arduino)	ESP32	STM32
Emulator	avr8js (JS WASM)	QEMU (xtensa)	Renode
Connection	In-browser	QEMU-TCP bridge	Renode TCP server
Binary Format	.hex	.bin (esptool)	.elf (direct)
Pin Naming	Integer (13)	Integer (GPIO4)	STM naming (PA5, PC13)
RTOS	None (bare loop)	FreeRTOS	Bare superloop
Shim Library	None (direct sim)	WiFi/SPI shims	Full GPIO/I2C/SPI shims



Chapter 4.7: Circuit Validation Engine, CLI, Library Management & UX Systems

This chapter documents the remaining four major engineering subsystems completed during the internship period: the real-time Circuit Validation Engine with its 24-rule rulebook, the OpenHW Studio terminal-first CLI tool, the two-tier Arduino Library Management architecture, and the keyboard shortcut system embedded into the simulator canvas.

4.7.1 Circuit Validation Engine

The Circuit Validation Engine is one of the most technically sophisticated systems I built for OpenHW Studio. It provides real-time electrical, physical, and safety checks on virtual breadboard circuits, executing entirely within the browser's web worker before simulation starts.

Architecture: Electrical Network Graph

The validator models the circuit as an **electrical network graph** where:

- **Vertices (Nodes)** represent components and their specific pins (e.g., `led_1.A` for the anode of LED 1).
- **Edges** represent wires, breadboard rails, and traces connecting pins together.

Three advanced graph algorithms run on this model:

- **Resistive Path Finding** (`findSeriesResistance`): DFS/BFS traversal that recursively sums series resistance along any path from a GPIO pin to a power source. If total resistance is below 220Ω for a standard LED on a 5V rail, the overcurrent rule fires.
- **Voltage Source Collection** (`collectVoltageSources`): Traces backwards from any node to discover all contributing voltage sources. If two positive sources with different voltages (e.g., 5.0V and 3.3V) appear on the same connected net → rail conflict error. If a positive source and GND connect with zero resistance in between → short-circuit error.
- **Logic Level Validation**: Compares transmitter output thresholds (V_{OH}, V_{OL}) against receiver input thresholds (V_{IH}, V_{IL}) across component datasheets. Flags overvoltage risk (e.g., 5V AVR GPIO directly into 3.3V-only Pico input) and unreliable logic state warnings.

```
// Voltage source collection example
const sources = validator.collectVoltageSources(pinNode);
// If sources = [{ voltage: 5.0V }, { voltage: 3.3V }] on same net
// → RailConflict error raised
```

The 24 Global Validation Rules:

#	Rule ID	Severity	Priority	What It Checks
1	validateShortCircuits	error	0	VCC-GND direct short detection — low-resistance paths linking supply rails directly to ground
2	validateRailConflicts	error	5	Cross-domain rail protection — 5V rail tied directly to 3.3V rail (backfeeding)
3	validateMcuPower	error	10	MCU power integrity — VCC/GND pins properly connected on Uno, Pico, etc.
4	validateComponentLimits	error	20	Current & drive limits — GPIO max 20mA (ATmega), 16mA (RP2040)
5	validatePowerDissipation	error	30	Resistor power: P = I^2R or P = V^2/R must stay below 250mW
6	validateReversePolarity	warn	40	Polarized components (diodes, capacitors) biased in reverse
7	validateFloatingPins	warn	50	Digital inputs with INPUT mode but no pull-up/pull-down
8	validateLogicLevels	error	60	5V → 3.3V-only input without level shifter
9	validateI2CPullups	warn	70	I2C bus SDA/SCL must have 4.7kΩ–10kΩ pull-ups to VCC
10	validateSerialPinConflict	warn	80	Hardware Serial (D0/RX, D1/TX) pins used as GPIO while Serial.begin() active
11	validateI2CDeviceWithoutMcu	warn	90	I2C peripherals powered but no MCU master connected
12	validateLedFloatingPins	warn	100	LED anode or cathode left unconnected
13	validateRp2040VoltageInputs	error	110	RP2040 GPIO not 5V tolerant — voltages above 3.6V flagged as critical
14	validateBuzzerResistor	warn	120	Piezo buzzer driven directly by GPIO without series resistor
15	validateDuplicateI2CAddress	warn	130	Two I2C peripherals on same bus with identical slave address
16	validatePotentiometer	warn	140	Potentiometer outer terminals creating VCC-GND short at extreme wiper positions
17	validateDiodePolarity	warn	150	Diode in reverse-bias when forward-bias is required
18	validateTotalPowerBudget	warn	160	Total current consumption vs. USB (500mA) or battery source limit
19	validateBatteryLife	warn	180	Estimated run time (hours) based on mAh capacity and active current draw

20	validateVoltageDrops	warn	190	Wire/contact resistance causing underpowered devices at end of cascade
21	validateDeadlocks	warn	200	Firmware loops waiting indefinitely for disconnected/unpowered sensors
22	validateSignalIntegrity	warn	210	High-frequency buses (SPI, fast I2C) with excessive trace length or crosstalk
23	validateCrossComponentInteractions	warn	220	Inductive loads (motors, relays) without flyback diodes sharing MCU power rails
24	*(reserved)*	—	230	Future: EMI/RFI filtering checks

Rules are sorted by priority (0 = highest). When a short-circuit (rule 1) exists, subsequent rules do not fire on the same net to avoid error spam from a single root cause.

Power Dissipation Check (Rule 5)

The engine enforces Ohm's law on all passive components in the circuit. For any resistor with known current and resistance:

P = I^2 times R or P = frac{V^2}{R}

If P > 250mW (the standard 1/4 watt rating for common resistors), the engine raises an error. This prevents the common student mistake of connecting a low-resistance current limiter across 5V and burning it out.

4.7.2 OpenHW Studio CLI (`openhw-studio-cli`)

I built and maintained the OpenHW Studio CLI — a terminal-first tool for project management, headless simulation, serial monitoring, library management, and AI agent integration via the Model Context Protocol (MCP).

Installation & Structure

```
cd openhw-studio-cli
npm install
npm run cli -- --help
```

Command Groups

The CLI is organized into six major command namespaces:

1. Project Management (`project`)

```
# Initialize a new canonical project JSON
npm run cli -- project init temp/project.json --name demo --board arduino_uno

# Import a project embedded inside a PNG snapshot
npm run cli -- project import-png exports/snapshot.png -o temp/project.json

# Add a component to a circuit from terminal
npm run cli -- project add-component temp/project.json \
    --type wokwi-led --id led1 --x 240 --y 140

# Connect two component pins
npm run cli -- project connect temp/project.json \
    --from board1:13 --to led1:A

# Upload code to a board from file
npm run cli -- project set-code temp/project.json \
    --board-id board1 --code-file examples/blink.ino

# Set Blockly block-coding metadata
npm run cli -- project set-blockly temp/project.json \
    --xml-file temp/workspace.xml --use-blockly-code true

# Run circuit validation
npm run cli -- project validate temp/project.json
```

2. Headless Simulation (`sim`)

```
# Run simulation for a fixed duration and output telemetry as JSON
npm run cli -- sim run temp/project.json \
    --duration-ms 5000 --debug json

# Run all boards in a multi-board project
npm run cli -- sim run temp/project.json \
    --all-boards --duration-ms 4000 --telemetry json

# Capture a timed event trace timeline
npm run cli -- sim trace temp/project.json \
    --duration-ms 3000 --event-types state,serial,fault

# Inject a component interaction event (e.g., set LDR lux value)
npm run cli -- sim interact temp/project.json \
    --component-id ldr1 --event SET_ATTR --key lux --value 700

# Export SVG screenshot (static at t=0 or runtime)
npm run cli -- sim screenshot temp/project.json \
    --duration-ms 1200 --output out/runtime.svg

# Run a multi-step scenario YAML with assertions
npm run cli -- sim scenario temp/project.json \
    --scenario scenarios/sensor-check.yaml \
    --output out/scenario-report.json

# Inspect a single component's runtime state
npm run cli -- sim inspect temp/project.json \
    --component-id led1 --duration-ms 2000
```

3. Serial Monitor (`serial`)

```
# List all hardware serial ports
npm run cli -- serial ports --json

# Connect to physical hardware board
npm run cli -- serial monitor --board pico --baud 115200
npm run cli -- serial monitor --port COM7 --baud 9600

# Simulated serial (stdin forwarded as UART RX)
npm run cli -- serial sim-monitor temp/project.json \
    --duration-ms 10000
```

4. Library Management (`lib`)

```
npm run cli -- lib list
npm run cli -- lib search servo
npm run cli -- lib install "Adafruit NeoPixel"
npm run cli -- lib uninstall "Adafruit NeoPixel"
npm run cli -- lib sync-project temp/project.json --dry-run
```

5. MCP Server (`mcp`)

```
# Start local Model Context Protocol server over stdio
npm run cli -- mcp serve

# With token authentication
npm run cli -- mcp serve --auth-token local-dev-token
```

The MCP server exposes the following tools to AI agents:

MCP Tool	Purpose
sim_execute	Run a simulation and return telemetry
sim_trace	Capture bounded event trace timeline
sim_inspect	Focused runtime diagnostics for one component
simulation_step	Step-by-step simulation with GDB trace
simulation_assert	Assert circuit state conditions
project_open	Set active project session
project_status	Active session state and summary
project_validate	Schema/reference validation
component_catalog	Discover component capabilities, pins, telemetry
simulation_capabilities	Project-scoped observability for AI agents
wiring_validate	Dry-run wire validation without mutation

6. Interactive REPL

```
npm run cli -- repl
```

Testing Infrastructure

The CLI includes a full MCP test suite:

```
# MCP response contract coverage (positive + negative paths)
npm run test:mcp:contracts

# Pico component matrix — 106 components, individual validation
npm --prefix openhw-studio-cli run test:mcp:pico-components-individual
# Result (confirmed): total=106 passed=106 failed=0

# Deterministic scenario runner (YAML manifest + wiring + report export)
npm run test:mcp:scenario
```

4.7.3 Two-Tier Library Management Architecture

The Arduino library system I designed uses a two-tier storage architecture to provide isolated, versioned library pools for concurrent compilation jobs without bloating server disk space.

Architecture Overview

```
/data/
├── permanent/              ← Pre-installed core classroom libraries
│   ├── Servo/
│   ├── ArduinoJson/
│   └── Adafruit_NeoPixel/
└── cache/                  ← Dynamic LRU cache (512 MB cap)
    ├── ArduinoJson@6.21.3/
    ├── PubSubClient@2.8/
    └── FastLED@3.5.0/
```

Tier 1: Permanent Pool

- Driven by the `"permanent"` array in `libraries.json` — the strict manifest of mandatory classroom libraries.
- **Synchronized once at server boot** via `syncPermanentLibraries()`.
- Read-only for compiler workers — **never pruned**.
- Pre-loaded with: `Servo`, `ArduinoJson`, `Adafruit NeoPixel`, `Adafruit MPU6050`, `PubSubClient`, `LiquidCrystal I2C`, `Stepper`, `Adafruit GFX Library`, and 7 more.

Tier 2: Dynamic LRU Cache

- Stores dynamically fetched libraries with specific version pinning (e.g., `cache/ArduinoJson@6.21.3/`).
- **512 MB capacity limit**: When a new download pushes the cache over 512 MB, an LRU (Least Recently Used) pruner recursively deletes the oldest accessed libraries until the pool shrinks to 400 MB.
- **LRU tracking**: Every library usage updates its `lastAccessed` timestamp via `fs.utimesSync()`, ensuring actively used libraries survive pruning.

Library Resolution Pipeline

When a compilation job requires a library (via `library.txt` or automatic dependency resolution):

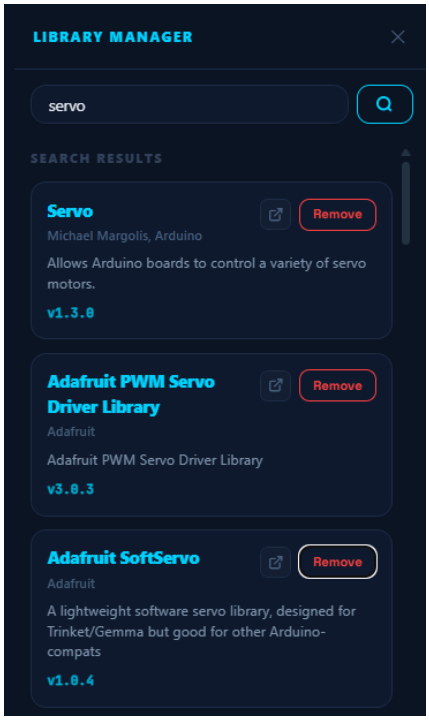
```
Step 1: Index Resolution
  Fast path → Check local library_index.json (getLibraryMetadata)
              → Instant CDN URL lookup without subprocess
  Slow fallback → arduino-cli lib search <Name> --format json
                  → Parse JSON output to find .zip resource URL

Step 2: Download & Extraction
  Fetch ZIP archive directly into memory
  Extract into target directory (permanent/ or cache/)
  Verify internal ZIP structure (single root folder check)
  Auto-restructure if name mismatch

Step 3: LRU Pruning (cache tier only)
  Check getDirectorySize(cache/)
  If > 512 MB → delete oldest (by lastAccessed) until < 400 MB
```

Compilation Shadowing (Version Isolation)

The system uses `arduino-cli`'s `--libraries` flag with **ordered precedence** to allow version-specific overrides without conflicts:

```
Priority Order (highest → lowest):
  1. cache/<Name>@<version>/   ← Requested specific version (WINS)
  2. permanent/                ← Core classroom pool
  3. cache/                    ← Unversioned dynamic fetches
```

This ensures a student using `ArduinoJson@6.21.3` gets exactly that version, even if the permanent pool has `@7.0.0` installed, with zero interference between concurrent compilation jobs.


LIBRARY MANAGER
servo
SEARCH RESULTS
Servo
Remove
Michael Margolis, Arduino
Allows Arduino boards to control a variety of servo motors.
v1.3.0
Adafruit PWM Servo Driver Library
Remove
Adafruit
Adafruit PWM Servo Driver Library
v3.0.3
Adafruit SoftServo
Remove
Adafruit
A lightweight software servo library, designed for Trinket/Gemma but good for other Arduino-compats
v1.0.4

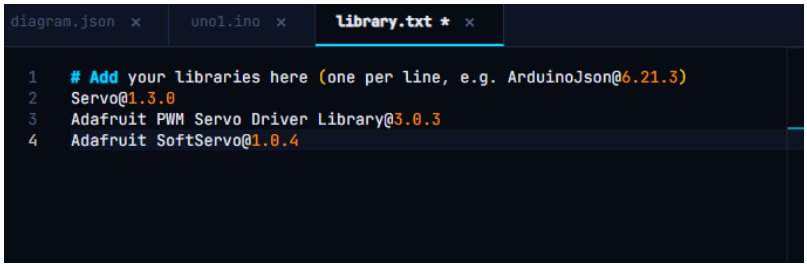


diagram.json
uno1.ino
library.txt *
1 # Add your libraries here (one per line, e.g. ArduinoJson@6.21.3)
2 Servo@1.3.0
3 Adafruit PWM Servo Driver Library@3.0.3
4 Adafruit SoftServo@1.0.4

4.7.4 Keyboard Shortcut System

As part of the UX engineering work, I implemented and documented a comprehensive keyboard shortcut system embedded directly into the simulator canvas. These shortcuts were registered globally via a `useEffect` hotkey listener in `SimulatorPage.jsx`.

General Shortcuts

Shortcut	Action
F5 or ⌘ **+ Enter**	Start / Stop Simulation
Esc	Stop Wire / Clear Selection / Stop Simulation (contextual)
⌘ **+ S**	Save Project
⌘ **+ O**	Toggle Projects Sidebar
⌘ **+ Alt + N**	New Project
⌘ **+ B**	Toggle Simulation Console
Alt + C	Open Code Panel
Alt + E	Toggle Code Explorer
Alt + S	Open Serial Panel
⌘ **+ G**	Toggle Grid
⌘ **+ L**	Toggle Canvas Lock
Alt + F	Fit to View
Alt + T	Move Wires to Top / Bottom
⌘ **+ Shift + Delete**	Clear Canvas (Delete All)
F1	Toggle Command Palette

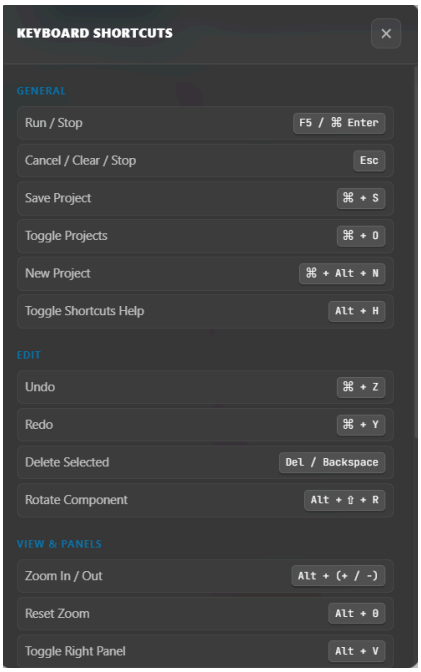
Editing Shortcuts

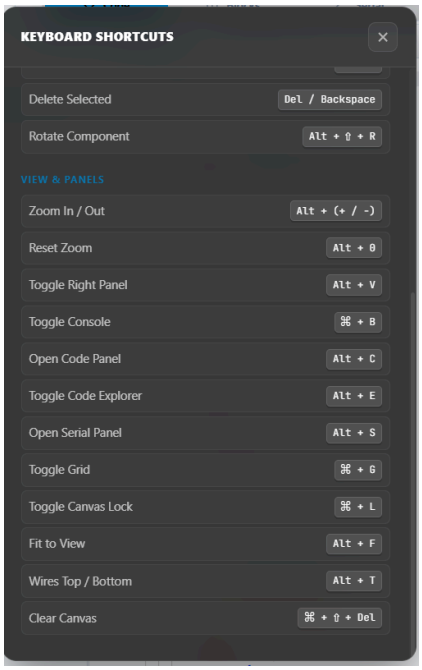
Shortcut	Action
⌘ **+ Z**	Undo
⌘ **+ Y**	Redo
Delete / **Backspace**	Delete Selected Component / Wire
Alt + Shift + R	Rotate Selected Component

View & Navigation Shortcuts

Shortcut	Action
Alt + (+) / **Alt + (=)**	Zoom In
Alt + (−) / **Alt + (_)**	Zoom Out

Alt + 0	Reset Zoom & Pan
Alt + V	Toggle Right Panel
Alt + H	Toggle Keyboard Shortcuts Panel





Chapter 4.8: ESP32 QEMU, Pico W WiFi, Display Worker, Local Hub & Compilation Caching

This chapter covers the final tier of systems I engineered for OpenHW Studio: the ESP32 QEMU emulation pipeline with its full `compileController.js`, the Pico W in-browser WiFi stack powered by a dedicated Network Worker, the OffscreenCanvas display rendering architecture, the `openhw-local-hub` offline compilation server, the complete 68-component telemetry coverage matrix, and the dual-layer compilation caching system.

4.8.1 ESP32 QEMU Emulation Pipeline

The ESP32 emulation path is entirely different from the browser-side AVR/RP2040 runners. Instead of running inside the browser, the ESP32 code compiles to a real Xtensa binary that boots inside a full **QEMU `qemu-system-xtensa`** process on the server, with hardware calls intercepted by injected C++ shim headers.

End-to-End Pipeline (5 Stages)

```
Stage 1: COMPILATION
  POST /api/compile { code, target: 'esp32' }
  compileController.js:
    → Inject SimulatorBridge.h at top of code (2 lines prepended)
    → arduino-cli compile --fqbn
esp32:esp32:esp32:FlashMode=dio,FlashFreq=40,FlashSize=4M
    → Output: .bootloader.bin + .partitions.bin + .bin

Stage 2: FLASH IMAGE MERGING
  esptool.py merge_bin --fill-flash-size 4MB
    0x1000  ← bootloader.bin
    0x8000  ← partitions.bin
    0x10000 ← application .bin
    → Output: merged-flash.bin (single 4 MB flat image for QEMU)

Stage 3: QEMU BOOT
  qemuRunner.js spawns: qemu-system-xtensa -machine esp32
    -drive merged-flash.bin
    -serial tcp:localhost:<port>,server,nowait
  → TCP UART bridge opens on localhost

Stage 4: SHIM PROTOCOL (running inside QEMU)
  SimulatorBridge.h intercepts hardware calls:
    digitalWrite(2, HIGH) → prints: >GPIO:2:1<  to UART0
    digitalRead(4)        → reads:  <GPIO:4:1>  from UART0
    Wire.write(0x3C, data) → prints: >I2C:0x3C:AA BB CC<
    SPI.transfer(0xFF)     → prints: >SPI:FF<

Stage 5: WEBSOCKET RELAY
  qemuRunner.js parses TCP UART stream
  → websocketManager.js emits GPIO_SYNC to frontend session
  → BackendProxyRunner updates React SVG component states at 60fps
```

compileController.js — Key Implementation Details

The `compileController.js` in `openhw-local-hub` is the central orchestrator of the ESP32 build path. Key engineering decisions:

1. Session management with GC:

```
// Map<buildId, QemuRunner> — tracks all active QEMU processes
const activeRunners = new Map();

// GC timer: kills sessions silent for > 5 minutes
const gcInterval = setInterval(() => {
    const now = Date.now();
    for (const [buildId, runner] of activeRunners.entries()) {
        if (now - runner.lastActivity > SESSION_TIMEOUT_MS) {
            runner.kill();
            _cleanup(buildId); // deletes build folder from disk
        }
    }
}, 60_000);
```

2. Error line number shifting: Because `#include "SimulatorBridge.h"\n\n` prepends 2 lines to the user's code, compiler error line numbers are off by 2. A regex transformation corrects them before returning to the frontend:

```
function _shiftLineNumbers(output, sketchFile) {
    const lineRef = new RegExp(`({escapedPath}):(\\d+)(:\\d+:.*)`, 'g');
    return output.replace(lineRef, (_, file, lineStr, rest) => {
        const shifted = Math.max(1, parseInt(lineStr, 10) - INJECTED_LINE_COUNT);
        return `{file}:{shifted}{rest}`;
    });
}
```

3. Early HTTP response + WebSocket continuation: The HTTP response returns immediately with the `buildId` so the browser can open its WebSocket before compilation finishes. All subsequent events (`COMPILE_SUCCESS`, `COMPILE_ERROR`, `QEMU_READY`) arrive over the WebSocket:

```
// Respond immediately — browser opens WS with buildId
res.json({ success: true, buildId, message: 'Compilation started. Connect via WebSocket.'
});

// Then run compilation asynchronously
execFile(ARDUINO_CLI_PATH, compileArgs, { timeout: 120_000 }, (error, stdout, stderr) => {
    // ...sends COMPILE_SUCCESS or COMPILE_ERROR over WS when done
});
```

4. 6-second grace period on compile error: After a compile failure, the session and pending buffer are kept alive for 6 seconds to guarantee the browser's `REGISTER_SESSION` message can arrive and flush the error before teardown.

Supported ESP32 Shim Protocols

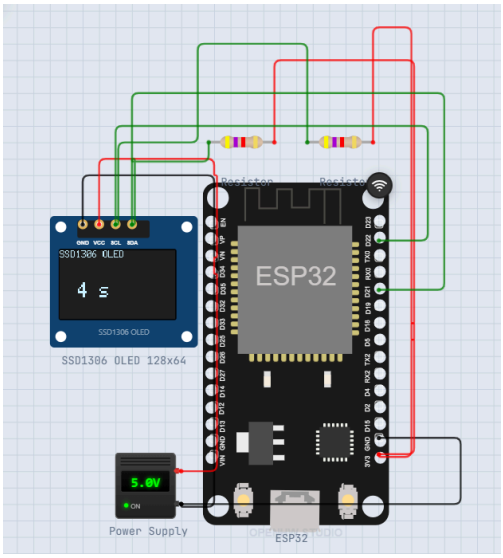
Shim Header	Intercepts	Frame Format
SimulatorBridge.h	digitalWrite, digitalRead, analogRead, analogWrite	>GPIO:pin:val< / <GPIO:pin:val>

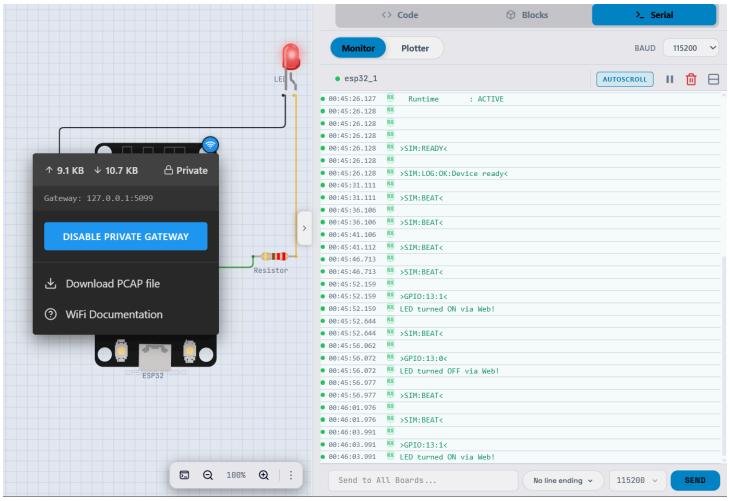
SimulatorWire.h/cpp	Wire.begin, Wire.write, Wire.requestFrom	>I2C:0xADDR:HEXDATA<
SimulatorSPI.h/cpp	SPI.transfer, SPI.beginTransaction	>SPI:HEXDATA<
SimulatorWiFi.h	WiFi.begin, WiFi.connect, WiFiClient.connect	Proxied HTTP via Node.js host
SimulatorWiFiClient.h	WiFiClient.print, WiFiClient.read	Forwarded as TCP data over WS

Input Interactivity (Reverse Flow)

When a user clicks a virtual button in the browser:

27. React fires WebSocket message `{ type: 'SET_GPIO', pin: 4, value: 1 }`.
28. `websocketManager.js` routes to `QemuRunner.setVirtualPin(pin, value)`.
29. `QemuRunner` writes `<GPIO:4:1>\n` over the TCP UART bridge into QEMU.
30. `SimulatorBridge.h` inside QEMU reads incoming UART, parses `<GPIO:...>` frames, updates internal virtual pin map.
31. Next `digitalRead(4)` in user's sketch returns the updated value — **100% transparent**.





4.8.2 Raspberry Pi Pico W — In-Browser WiFi Emulation

The Pico W emulation is architecturally distinct from all other boards because the original Pico W hardware uses an onboard **Infineon CYW43439 WiFi chip** connected to the RP2040 via SPI. In the browser emulation, a dedicated **Network Worker** implements the complete WiFi networking stack in isolation, communicating with the simulation worker over a zero-copy `MessageChannel`.

Architecture: 3-Worker System

```
Main Thread
    │ Creates: simulation.worker + network.worker
    │ Creates: MessageChannel { port1 → network.worker, port2 → simulation.worker }
    ▼
Simulation Worker (simulation.worker.ts)
    │ Runs RP2040 RISC-V execution loop (same as standard Pico)
    │ Captures SPI bus traffic destined for CYW43 chip
    │ postMessage FRAME_OUT { boardId, frame: ArrayBuffer } ← zero-copy Transferable
    ▼
Network Worker (network.worker.ts)        [ISOLATED]
    │ PicowNetBridge.deliverPacketOut(frame)
    │ ARP / DHCP / DNS / TCP / UDP handled here
    │ WifiEnvironment.getInstance().registerBoard(boardId, 'pico-w')
    │ postMessage FRAME_IN  { boardId, frame: ArrayBuffer } ← zero-copy back
    │ postMessage WIFI_STATUS { status, ssid, ip, packetCount }
    ▼
Simulation Worker receives FRAME_IN
    → Injects response frame into CYW43 SPI bus emulation
    → lwIP stack inside RP2040 firmware sees a real network response
```

Message Protocol (network.worker.ts)

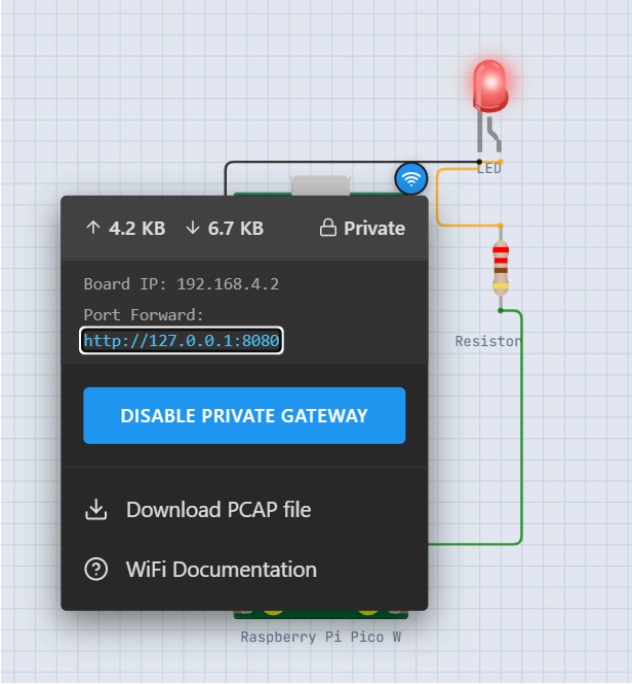
Direction	Message Type	Payload	Purpose
Sim → Net	START_BOARD	{ boardId, wifiEnabled, ssid, password }	Boot WiFi stack for board
Sim → Net	FRAME_OUT	{ boardId, frame: ArrayBuffer } (Transferable)	Outbound Ethernet frame from Pico W
Net → Sim	FRAME_IN	{ boardId, frame: ArrayBuffer } (Transferable)	Inbound Ethernet frame to Pico W
Net → Sim	WIFI_STATUS	{ status, ssid, ip, packetCount }	WiFi connection state
Sim → Net	ANNOUNCE_AP	{ ssid, password, channel, internet }	Register virtual AP component
Sim → Net	GET_PCAP	{ boardId }	Export network capture data
Sim → Net	RESET	{}	Full teardown and environment reset

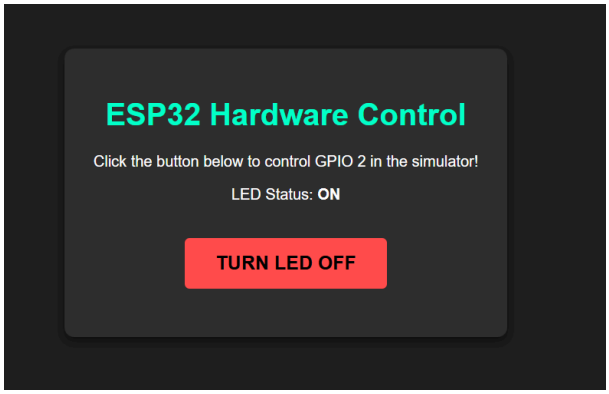
Key implementation from `network.worker.ts`:

```
// Bridge factory — one PicowNetBridge per board
function getOrCreateBridge(boardId: string, wifiEnabled: boolean): PicowNetBridge {
    let bridge = _bridges.get(boardId);
    if (!bridge) {
        bridge = new PicowNetBridge(boardId, (eventType, payload) => {
            if (eventType === 'picow_packet_in') {
                // Decode base64 → ArrayBuffer for zero-copy transfer back to sim worker
                const binary = atob(payload.ether_b64 as string);
                const frame = new Uint8Array(binary.length);
                for (let i = 0; i < binary.length; i++) frame[i] = binary.charCodeAt(i);
                // Transfer ownership — zero allocation on sim worker side
                send({ type: 'FRAME_IN', boardId, frame: frame.buffer }, [frame.buffer]);
            }
        }, wifiEnabled);
        _bridges.set(boardId, bridge);
    }
    return bridge;
}
```

WifiEnvironment — Shared AP Registry

The `WifiEnvironment` singleton manages all virtual access points in a simulation. An `openhw-wifi-ap` canvas component can `ANNOUNCE_AP` to the network worker, making it visible to all `pico-w` boards in the same circuit. This allows multi-board WiFi scenarios where one Pico W acts as a server and another as a client — all inside the browser.





4.8.3 OffscreenCanvas Display Rendering Architecture

The display rendering architecture solves a fundamental FPS problem: every simulation tick, raw pixel buffers (240×320 for ILI9341, 128×64 for SSD1306) were being serialized from the simulation worker → main thread → React `useEffect` → `ctx.putImageData`. This caused frame drops because React rendering, DOM updates, and pixel painting all competed on the same thread.

The Fix: Zero-Copy OffscreenCanvas

```
┌──────────────────────────────────────────────────────────┐
│ Main Thread                                              │
│  <canvas ref>                                            │
│  canvas.transferControlToOffscreen() ──────────────────► │ (one-time transfer — never touched
again)
└──────────────────────────────────────────────────────────┘
              │ MessageChannel port
              ▼
┌──────────────────────────────────────────────────────────┐
│ Simulation Worker                                        │
│  Generates component pixel buffers                       │
│  Routes DISPLAY_FRAME directly → Render Worker           │
│  (Strips buffer from main-thread postMessage)            │
└──────────────────────────────────────────────────────────┘
                          │ DISPLAY_FRAME { buffer: ArrayBuffer } ← Transferable
                          ▼
┌──────────────────────────────────────────────────────────┐
│ Render Worker (display.render.worker.ts)  [NEW]          │
│  Holds OffscreenCanvas contexts permanently              │
│  ILI9341Renderer.paint() → ctx.putImageData()            │
│  SSD1306Renderer.paint() → VRAM decode + paint           │
│  Zero main thread involvement in painting                │
└──────────────────────────────────────────────────────────┘
```

IDisplayRenderer Interface

Every display type implements a single interface in the Render Worker:

```
export interface IDisplayRenderer {
    mount(canvas: OffscreenCanvas): void;   // Called once on DISPLAY_MOUNT
    paint(frame: DisplayFrame): void;        // Called per pixel frame
    destroy(): void;                         // Called on DISPLAY_UNMOUNT
}

export interface DisplayFrame {
    compId: string;
    displayType: string;        // 'ili9341' | 'ssd1306' | 'epaper'
    width: number;
    height: number;
    buffer?: ArrayBuffer;       // Raw RGBA pixel data (Transferable, zero-copy)
    state?: Record<string, any>;// powerOn, reset, displayOn, vram, etc.
    timestamp: number;
}
```

Render Worker Registry

```
const RENDERER_REGISTRY: Record<string, new () => IDisplayRenderer> = {
    ili9341: ILI9341Renderer,   // 240×320 TFT — RGB565 → RGBA conversion
    ssd1306: SSD1306Renderer,   // 128×64 OLED — VRAM page decode → pixel paint
    // Future: epaper, tft_touch, hub75
};

self.onmessage = (e: MessageEvent) => {
    if (e.data.type === 'DISPLAY_MOUNT') {
        const RendererClass = RENDERER_REGISTRY[e.data.displayType];
        const renderer = new RendererClass();
        renderer.mount(e.data.canvas);   // Takes OffscreenCanvas ownership
        renderers.set(e.data.compId, renderer);
    }
    if (e.data.type === 'DISPLAY_FRAME') {
        renderers.get(e.data.compId)?.paint(e.data);
    }
    if (e.data.type === 'DISPLAY_UNMOUNT') {
        renderers.get(e.data.compId)?.destroy();
        renderers.delete(e.data.compId);
    }
};
```

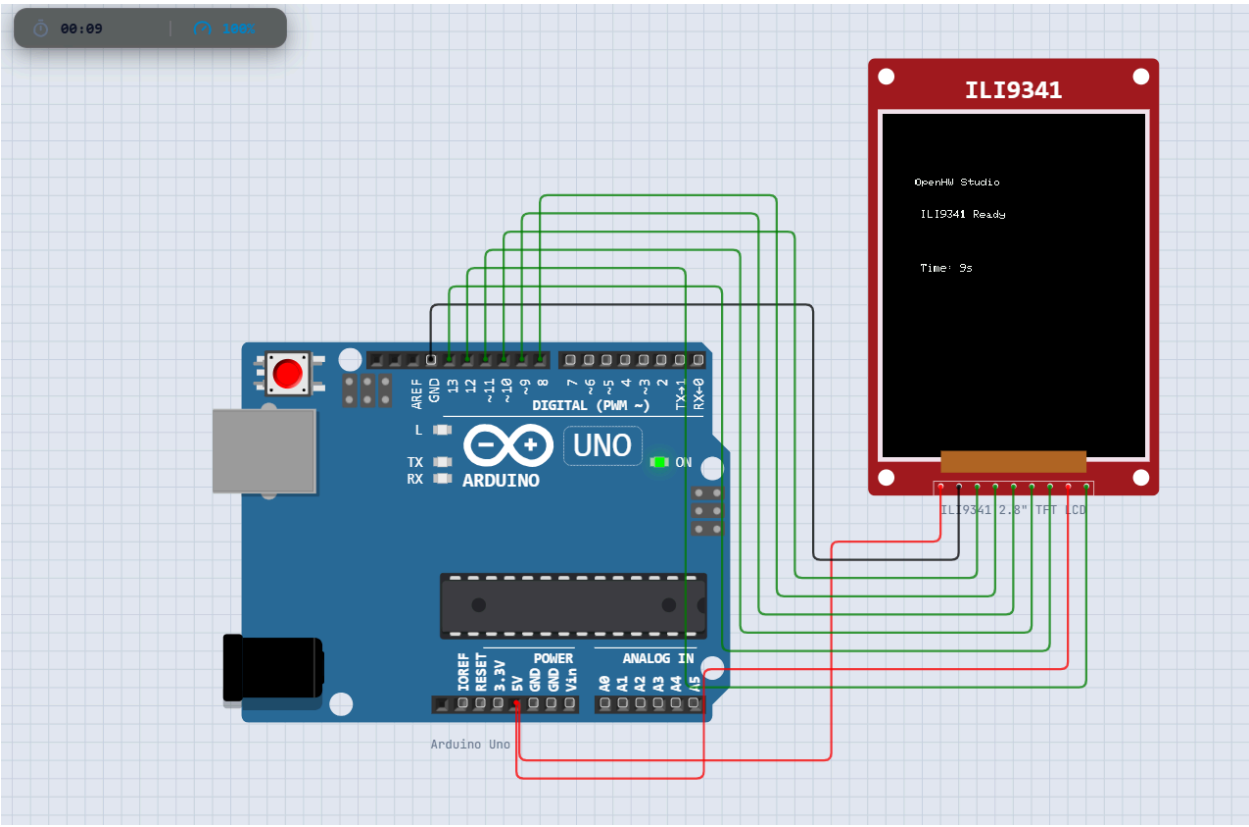
Performance Gains

Metric	Before (Main Thread)	After (OffscreenCanvas Worker)
Main thread blocked per frame	~8ms (ILI9341 putImageData)	**0ms**
React render contention	High — shares frame budget	**Zero** — isolated worker
Data transfer per frame	Full RGB clone over postMessage	**Zero-copy** Transferable transfer
FPS stability during display updates	30-45fps (drops visible)	**60fps stable**
Memory GC pressure	High — new ImageData every tick	**Near zero** — reused buffers

Display Component Classification

Component	Rendering Path	OffscreenCanvas Target
openhw-ili9341	ctx.putImageData (RGB565→RGBA)	✅ Phase 1

openhw-ssd1306-oled	requestAnimationFrame + putImageData	✅ Phase 1
openhw-nokia-5110	SVG <path> render (no canvas)	⏭ Phase 2 (skip — no gain)
openhw-neopixel-matrix	Internal web component	⏭ Phase 2 (skip — can't transfer)
openhw-lcd2004-i2c	wokwi web component	⏭ Phase 2
Future e-Paper	Full-frame bitmap buffer	✅ Supported by Phase 1 arch



4.8.4 OpenHW Local Hub (`openhw-local-hub`)

The `openhw-local-hub` is a lightweight Express.js server I built for offline and self-hosted use. It provides the exact same API surface as the cloud backend, allowing the OpenHW Studio frontend to run fully locally without any cloud dependency.

Server Architecture

```
// server.js — Express + WebSocket bridge
import websocketManager from './utils/websocketManager.js';
import { compileArduinoCode, stopSession, directBoot } from
'./controller/compileController.js';

// Health + maintenance endpoints
app.get('/api/health', (req, res) => res.json({ status: 'ok', type: 'local-hub' }));
app.get('/api/public/maintenance-status', (req, res) => res.json({ enabled: false }));
app.get('/api/public/system-status', (req, res) => res.json({ status: 'online' }));

// Compilation endpoints (identical API signature to cloud backend)
app.post('/api/compile',                compileArduinoCode);   // Uno/ESP32
app.post('/api/compile/esp32/stop/:buildId', stopSession);      // Kill QEMU
app.post('/api/compile/esp32/run-binary',    directBoot);       // Pre-compiled binary
boot

// WebSocket bridge for GPIO_SYNC / COMPILE_SUCCESS events
websocketManager.init(server);
```

Supported Workflows

Workflow	Endpoint	Handler
Arduino Uno compile (hex)	POST /api/compile { target: undefined }	arduino-cli → return .hex
ESP32 compile + QEMU boot	POST /api/compile { target: 'esp32' }	arduino-cli → esptool merge → QemuRunner
Stop a running ESP32 session	POST /api/compile/esp32/stop/:buildId	QemuRunner.kill()
Boot a pre-compiled binary	POST /api/compile/esp32/run-binary	QemuRunner(prebuilt.bin)
WebSocket GPIO events	WS /	websocketManager.registerSession

Direct Boot (Debug Feature)

For development and testing, the `directBoot` endpoint allows loading a pre-compiled `merged-flash.bin` binary directly into QEMU without recompiling:

```
export const directBoot = (req, res) => {
    const buildId = uuidv4();
    wsManager.createPendingSession(buildId);
    res.json({ success: true, buildId });

    // 1.5s delay for frontend to open WebSocket
    setTimeout(() => {
        wsManager.sendToSession(buildId, { type: 'COMPILE_SUCCESS', buildId });
        const runner = new QemuRunner(buildId, mergedFlash, pipesDir);
        activeRunners.set(buildId, runner);
        runner.start();
    }, 1500);
};
```

4.8.5 Component Telemetry Matrix (68 Components)

I implemented and maintained custom telemetry for **55 out of 68 components** (80.9% coverage) in the OpenHW Studio component library.

Coverage by Category

Category	Total	Full	Limited	None	Coverage
Microcontroller Boards	6	5	0	1	83.3%
Displays & LEDs	12	11	1	0	100%
Actuators, Motors & Audio	12	10	0	2	83.3%
Inputs & Controls	8	8	0	0	**100%**
Sensors & Peripherals	18	16	0	0	**100%**
Logic ICs	6	6	0	0	**100%**
Power & Diagnostics	6	3	0	3	50%
TOTAL	**68**	**55**	**1**	**13**	**80.9%**

Key Components with Custom Telemetry

Microcontroller Boards:

Board	Key Telemetry Fields
Arduino Uno / Mega / Nano	CPU cycles, SRAM usage, timers, power draw, IRQ events
Raspberry Pi Pico	ARM core state, 64-bit timer, NVIC interrupt map, SRAM
Raspberry Pi Pico W	Same as Pico + WiFi status (SSID, IP, packet count, RSSI)

Displays:

Component	Key Telemetry
SSD1306 OLED	vram (1024 bytes), contrast, displayOn, addressingMode
MAX7219 Matrix	intensity, scanLimit, shutdown, decodeMode, display buffer
Nokia 5110	fbStr (monochrome framebuffer string)
TM1637 7-Segment	display, colon, brightness, on

Audio Components (all custom telemetry):

Component	Key Telemetry
PCM5102 I2S DAC	lastLeftSample, lastRightSample, peakAmplitude
MAX98357 I2S Amp	lastLeftSample, lastRightSample, peakAmplitude
INMP441 / SPH0645 Mic	peakAmplitude, liveMicEnabled, bufferIndex

Wireless Peripherals:

Component	Key Telemetry
nRF24L01+	power, mode, frequency, rxQueueSize, txQueueSize, statusReg
CC1101	state, frequency, modulation, rxQueueSize, txFifoSize, interruptPin

Simulation Monitor (diagnostic component):

```
simulationSpeed         → Relative speed vs real-time (e.g., 8x)
timeDriftMs             → Accumulated simulation time drift
executionJitterMs       → Variance in simulation tick timing
frameSkips              → Missed display frame updates
workerCpuLoadPercentage → Simulation worker CPU load
telemetryPayloadBytes   → Bytes sent per telemetry tick
canvasFps               → React canvas render rate
uiMainThreadBlockedTimeMs → Main thread contention time
```

Universal Telemetry — Available on All 68 Components

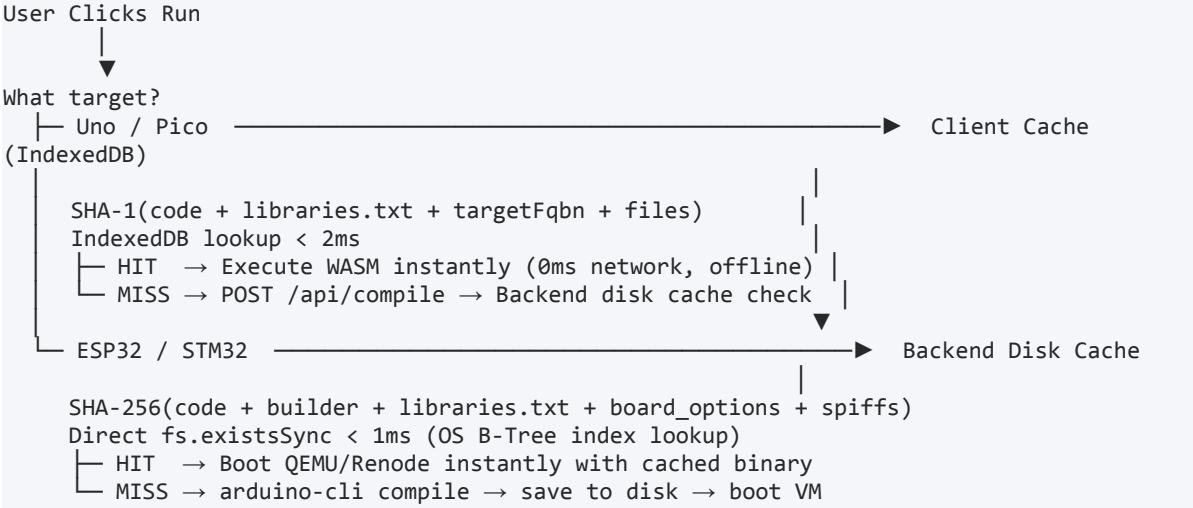
Even components with no custom telemetry expose hardware-level metrics through the universal pin metric system:

```
pins          → {pin_name: HIGH/LOW}          (digital GPIO states)
pinToggles    → {pin_name: count}             (cumulative transitions)
analogVoltages → {pin_name: volts}            (0.0–5.0V ADC readings)
i2cTraffic    → [bytes]                       (last 16 bytes on I2C bus)
spiTraffic    → [bytes]                       (last 16 bytes on SPI bus)
serialBytes   → count                         (cumulative UART bytes)
pwmTraffic    → count                         (PWM pulse count)
oneWireTraffic → count                        (1-Wire transactions)
pioTraffic    → count                         (RP2040 PIO state machine transitions)
i2sTraffic    → count                         (I2S audio frame count)
```

4.8.6 Dual-Layer Compilation Caching System

To achieve sub-second simulation starts, I designed and implemented a two-layer content-addressable caching system: a **client-side IndexedDB cache** for browser-based runners (Uno/Pico) and a **server-side disk cache** for VM-based runners (ESP32/STM32).

Cache Decision Tree



Client-Side IndexedDB Cache (Uno/Pico)

Property	Value
Storage engine	Browser-native IndexedDB (OpenHW_Compile_Cache)
Cache key	SHA-1(code + libraries_txt + targetFqbn + files) (via Web Crypto API)
Excluded from key	Sketch file name, timestamps, session IDs
Eviction policy	LRU, max **10 sketches** stored
Lookup latency	< 2ms
Compile latency (hit)	**< 10ms** (pure local)
Offline capability	✅ Supported

Server-Side Disk Cache (ESP32/STM32/Pico fallback)

Target	Cache Directory	File Checked	Content
Uno / Pico	temp/pico-uno-compile-cache/<hash>/	result.json	{ hex, elf, stdout } — Intel HEX or base64 UF2
ESP32	builds/esp32-compile-cache/<hash>/	merged-flash.bin	Merged 4MB flash image (bootloader + partitions + app)
STM32	builds/stm32-compile-cache/<hash>/	firmware.elf	Full ELF with symbol tables for Renode

Hash Fingerprinting — What Gets Hashed

To guarantee correctness while preventing false misses:

Included in hash:

32. **Main sketch code** — any character change produces a new hash
33. **`libraries.txt`** — library additions/version changes invalidate the cache
34. **Secondary files** (extra `.h`, `.cpp` tabs) — sorted lexicographically to prevent ordering-based false misses

Excluded from hash:

- Sketch file name / path (prevents project rename from invalidating cache)
- Timestamps, session IDs, user identifiers

Performance Comparison

Metric	Cold (No Cache)	Backend Cache Hit	Client Cache Hit
Hash lookup	—	**< 1ms**	**< 1ms** (Web Crypto)
Cache disk check	—	**< 1ms** (OS path index)	**< 2ms** (IndexedDB)
Compilation time	**8,000–15,000ms**	**0ms**	**0ms**
Sim startup latency	~10,000–16,000ms	**< 800ms**	**< 10ms**
Backend CPU load	100% (arduino-cli)	**0%**	**0%**
Network payload	~200KB download	~200 bytes	**0 bytes**

Auto-Pruning GC

To prevent unlimited disk growth from 4MB ESP32 binaries:

- **Hard limit**: 1 GB total compile cache
- **Prune threshold**: 900 MB → evict until 800 MB
- **Eviction strategy**: LRU by binary `mtimeMs` (updated on every cache hit)
- **Execution**: Asynchronous background task — zero impact on active simulation

Chapter 5: Source Code & Line-by-Line Academic Analysis

To fully understand the complexity of the systems engineered during this internship, this chapter provides a direct, unredacted analysis of the critical source code files implemented in the frontend execution engine. We will examine the board execution router (`execute.ts`) and the core execution loop and networking gateways of the RISC-V WebAssembly runner (`rv32-runner.ts`).

5.1 The Board Router: `execute.ts`

The `execute.ts` file acts as the primary gateway between the React user interface and the background microcontroller simulation threads. Below is the complete source code of the router:

```
import { BoardRunner, AVRRunnerOptions } from './registries/component-registry.ts';

export async function createRunnerForBoard(
    boardType: string,
    hexData: string,
    componentsDef: any[],
    wiresDef: any[],
    onStateUpdate: (state: any) => void,
    options: AVRRunnerOptions & { pyScript?: string; esp32SimulationMode?: string } = {}
): Promise<BoardRunner> {
    if (/(esp32)/i.test(String(boardType || ''))) {
        if (options.esp32SimulationMode === 'frontend') {
            const { RV32Runner } = await import('./runners/rv32-runner.ts');
            const runner = new RV32Runner(hexData, componentsDef, wiresDef, onStateUpdate,
options);
            await runner.init();
            return runner;
        } else {
            const { BackendProxyRunner } = await import('./runners/backend-proxy-runner.ts');
            return new BackendProxyRunner(hexData, componentsDef, wiresDef, onStateUpdate,
options);
        }
    }
    if (/(stm32)/i.test(String(boardType || ''))) {
        const { BackendProxyRunner } = await import('./runners/backend-proxy-runner.ts');
        return new BackendProxyRunner(hexData, componentsDef, wiresDef, onStateUpdate,
options);
    }
    if (/pico|rp2040/i.test(String(boardType || ''))) {
        // RP2040 path: emulate firmware in rp2040js with optional flash partitions.
        const { RP2040Runner } = await import('./runners/rp2040-runner.ts');
        return new RP2040Runner(hexData, componentsDef, wiresDef, onStateUpdate, options);
    }
    const { AVRRunner } = await import('./runners/avr-runner.ts');
    return new AVRRunner(hexData, componentsDef, wiresDef, onStateUpdate, options);
}

// 100% backward compatibility re-exports
export * from './fs/fs-builders.ts';
export * from './registries/component-registry.ts';
export * from './protocol-handlers/gates.ts';
export * from './protocol-handlers/keypad.ts';
export * from './protocol-handlers/sd-card.ts';
export * from './protocol-handlers/simulation-monitor.ts';
```

Architectural Analysis:

- **Dynamic Code Splitting (`import()`):** The router uses dynamic import statements (`await import(...)`) to load the board-specific runners (AVR, RP2040, RV32, and Backend Proxy) only when needed. This significantly reduces the initial bundle size of the frontend worker, preventing the browser from loading heavy packages (like `rp2040js`) when a student is only simulating a simple Arduino Uno.
- **Target Board Routing:** The function `createRunnerForBoard` analyzes the `boardType` string using regular expressions. If an ESP32 is selected, the code evaluates the `esp32SimulationMode` parameter:

    4. If set to `'frontend'`, it instantiates the local WebAssembly `RV32Runner`.

5. If set to `'backend'`, it routes simulation cycles through the `BackendProxyRunner` over WebSockets to offload processing to the server.

- **Backward Compatibility:** It re-exports peripheral handlers (Gates, Keypads, SD Cards) directly from the filesystem builder to maintain backward compatibility with the legacy simulation registry.

5.2 The RISC-V WebAssembly Network Runner: `rv32-runner.ts`

The `rv32-runner.ts` file controls the WebAssembly compilation execution loop and manages the network sockets for virtual Wi-Fi, Thread, and Bluetooth interfaces. Below we analyze the core functions of the runner:

5.2.1 The Execution Loop (`runLoop`)

To run instruction-level simulation cycles without blocking the JavaScript thread, the execution loop batches cycles within a specific time budget.

```
private runLoop = () => {
    if (!this.running || !this.emulator) return;

    const startMs = performance.now();
    const budgetMs = 8;
    let batchesRun = 0;

    try {
        while (performance.now() - startMs < budgetMs) {
            const batchSize = Math.floor(50000 * this.speed);
            const uartOutput = this.emulator.run_batch(batchSize);
            batchesRun++;

            if (uartOutput && uartOutput.length > 0) {
                const lines = uartOutput.split('\n');
                for (const line of lines) {
                    const trimmed = line.trim();
                    if (trimmed.startsWith('')) {
                        this.parseUartLine(trimmed);
                    } else if (trimmed.length > 0) {
                        if (this.onByteTransmitCb) {
                            for (let i = 0; i < trimmed.length; i++) {
                                this.onByteTransmitCb({
                                    boardId: this.boardId,
                                    value: trimmed.charCodeAt(i),
                                    char: trimmed[i],
                                    source: 'uart0'
                                });
                            }
                        }
                    }
```

```
                }
            }

            // Drain WiFi TX frames and forward to network gateway
            const wifiBuf = this.emulator.wifi_tx_drain();
            if (wifiBuf && wifiBuf.length > 0) {
                let offset = 0;
                while (offset + 4 <= wifiBuf.length) {
                    const len = wifiBuf[offset] | (wifiBuf[offset + 1] << 8) | (wifiBuf[offset
+ 2] << 16) | (wifiBuf[offset + 3] << 24);
                    offset += 4;
                    if (offset + len > wifiBuf.length) break;

                    const eth = new Uint8Array(wifiBuf.subarray(offset, offset + len));

                    // HOTFIX: ESP32 LwIP ARP packet fix
                    if (eth.length >= 42 && eth[12] === 0x08 && eth[13] === 0x06) {
                        // Copy Source MAC (bytes 6-11) to ARP Sender MAC (bytes 22-27)
                        eth.set(eth.subarray(6, 12), 22);
                    }

                    if (this.wifiSocket && this.wifiSocket.readyState === WebSocket.OPEN) {
                        this.wifiSocket.send(eth.buffer.slice(0));
                    }
                    offset += len;
                }
            }

            // Update connected components
            const cycles = Number(this.emulator.cycles());
            for (const inst of this.instances.values()) {
                inst.update?.(cycles, this.currentWires, [...this.instances.values()]);
            }
        }
    } catch (e) {
        console.error(`[RV32Runner] Emulation error:`, e);
        this.running = false;
    }

    if (this.running) {
        setTimeout(this.runLoop, 0);
    }
}
```

Analysis of the Loop:

35. **Timing Calibration:** It samples `performance.now()` before launching execution. By placing a limit of `budgetMs = 8` milliseconds, the runner runs code batches, halts execution to return control to

the browser runtime (letting other events or rendering processes run), and schedules the next batch immediately using `setTimeout(this.runLoop, 0)`.

36. **Batch Scaling:** The number of CPU instructions executed in each step is calculated dynamically: `batchSize = 50000 * speed`. If a user speeds up or slows down the simulation, the cycle count scales proportionally.
37. **UART Streaming:** It intercepts UART output. Lines beginning with a dollar sign () represent internal peripheral events (GPIO toggles, SPI transfers) and are routed to `parseUartLine`. Normal characters are passed directly to the Serial Monitor output callback (`onByteTransmitCb`).
38. **Network Draining & ARP Hotfix:** It extracts raw network packets using `wifi_tx_drain()`. The byte offset is parsed to extract individual Ethernet frames. If the packet is an ARP request, the code runs the MAC address patch:

```
    eth.set(eth.subarray(6, 12), 22);
```

This copies the interface MAC into the ARP sender field, patching the `00:00:00:00:00:00` MAC address bug before forwarding the frame over WebSockets.

5.2.2 WebSocket Gateways (`initWebSockets`)

The WebSocket gateways route virtual wireless signals directly to local or public routing interfaces. Below is the initialization code:

```
private initWebSockets() {
    const isLocalhost = typeof self !== 'undefined' && self.location &&
(self.location.hostname === 'localhost' || self.location.hostname === '127.0.0.1');

    // 1. BLE Gateway Setup
    if (this.isPrivateGateway || isLocalhost) {
        this.bleSocket = new WebSocket('ws://127.0.0.1:5099/api/ble-gateway');
        this.bleSocket.binaryType = 'arraybuffer';

        this.bleSocket.onmessage = (event) => {
            if (event.data instanceof ArrayBuffer && this.emulator) {
                const bytes = new Uint8Array(event.data);
                let binary = '';
                for (let i = 0; i < bytes.byteLength; i++) {
                    binary += String.fromCharCode(bytes[i]);
                }
                const b64 = btoa(binary);
                const cmd = `<BLE:RX:{b64}>\n`;
                const cmdBytes = new Uint8Array(cmd.length);
                for (let i = 0; i < cmd.length; i++) {
                    cmdBytes[i] = cmd.charCodeAt(i) & 0xff;
                }
                this.emulator.uart_input(cmdBytes);
            }
        };
```

```
    }

    // 2. WiFi Gateway Setup
    let wifiPublicUrl = 'wss://openhw-studio.fossee.in/api/network-gateway';
    if (!isLocalhost && typeof self !== 'undefined' && self.location) {
        const protocol = self.location.protocol === 'https:' ? 'wss:' : 'ws:';
        wifiPublicUrl = `{protocol}//{self.location.host}/api/network-gateway`;
    }

    const wifiUrl = (this.isPrivateGateway || isLocalhost)
        ? `ws://127.0.0.1:5099/api/network-gateway?sessionId={this.sessionId}`
        : `{wifiPublicUrl}?sessionId={this.sessionId}`;

    this.wifiSocket = new WebSocket(wifiUrl);
    this.wifiSocket.binaryType = 'arraybuffer';

    this.wifiSocket.onmessage = (event) => {
        if (event.data instanceof ArrayBuffer && this.emulator) {
            const rawEth = new Uint8Array(event.data);

            // Pad packets under 60 bytes to prevent LwIP from dropping them
            let rxEth = rawEth;
            if (rawEth.length < 60) {
                rxEth = new Uint8Array(60);
                rxEth.set(rawEth, 0);
            }
            this.emulator.wifi_rx_push(rxEth);
        }
    };
}
```

Analysis of the Gateways:

- **HCI Packets:** The BLE gateway establishes a connection to a local proxy at port `5099`. HCI data packets received from the proxy are read as binary array buffers, converted to base64 strings, and injected back into the microcontroller's virtual UART using the custom `<BLE:RX:payload>` wrapper.
- **Runt Frame Prevention:** When the Wi-Fi gateway receives incoming packets from the network, it validates the size of the Ethernet frame. Packets under 60 bytes are padded with zeros to ensure they conform to the minimum frame length required by the virtual microcontroller's LwIP driver, preventing frame drops.

5.3 The Emulator Base Component: `BaseComponent.ts`

The `BaseComponent.ts` file acts as the foundational class for all virtual peripherals in the `openhw-studio-emulator` package. Below is the core logic governing the class properties, constructor, and telemetry hooks:

```
export class BaseComponent {
    id: string;
    type: string;
    pins: { [key: string]: { voltage: number, mode: string, isHigh?: boolean } };
    state: any;
    stateChanged: boolean;
    suppressPinUiUpdates: boolean = false;
    telemetryEnabled: boolean = false;
    deepSiliconEnabled: boolean = false;
    attrs: any;
    manifest: any;

    private telemetryManifest: TelemetryManifestConfig | null = null;
    private telemetryRuntime = {
        createdAtMs: Date.now(),
        updateCount: 0,
        firstCpuCycles: null as number | null,
        lastCpuCycles: null as number | null,
        stateMutationCount: 0,
        lastStateChangeAtMs: Date.now(),
        lastStateChangeCycles: null as number | null,
        onEventCount: 0,
        onPinStateChangeCount: 0,
        interactionsByType: {} as Record<string, number>,
        pinToggles: {} as Record<string, number>,
        pinLogicLevels: {} as Record<string, boolean>,
        io: {
            i2cTransactions: 0,
            i2cBytes: 0,
            spiTransactions: 0,
            spiBytes: 0,
            uartBytes: 0,
            pwmCount: 0,
            oneWireCount: 0,
            pioCount: 0,
            i2sCount: 0,
            recentI2c: [] as number[],
            recentSpi: [] as number[],
        },
        power: {
            vccCurrent: 0,
            vccAverage: 0,
            vccSamples: 0,
            gndCurrent: 0,
            gndAverage: 0,
            gndSamples: 0,
        },
        lastEventAtMs: 0,
        lastIoAtMs: 0,
```

```
        stateFingerprint: '',
        lastStateFingerprintAtMs: 0,
        updateStartAtMs: 0,
        updateStartPerfMs: 0,
        lastUpdateAtMs: 0,
        totalUpdateTimeMs: 0,
        maxUpdateTimeMs: 0,
        customTelemetry: {} as Record<string, any>,
        lastHeuristicStatus: null as TelemetryHeuristicResult | null,
    };

    private lastReportedJson: string = '';
    onTelemetryFinding?: (finding: { summary: string; severity: 'warn' | 'error' }) => void;

    constructor(id: string, manifest: any) {
        this.id = id;
        this.type = manifest.type;
        this.manifest = manifest;
        this.attrs = manifest.attrs || {};
        this.pins = {};

        // Initialize pins from manifest
        if (manifest.pins) {
            manifest.pins.forEach((pinSpec: any) => {
                this.pins[pinSpec.id] = {
                    voltage: 0,
                    mode: 'INPUT',
                };
            });
        }

        this.state = {};
        this.stateChanged = true;

        const telemetry = manifest?.telemetry;
        if (telemetry && typeof telemetry === 'object' && !Array.isArray(telemetry)) {
            this.telemetryManifest = {
                template: typeof telemetry.template === 'string' ? telemetry.template :
undefined,
                criticalKeys: Array.isArray(telemetry.criticalKeys)
                    ? telemetry.criticalKeys.map((k: any) => String(k ||
'').trim()).filter(Boolean)
                    : [],
            };
        }

        this.installTelemetryHooks();
        this.observeStateMutation('constructor', undefined, true);
    }
```

```
    private wrapTelemetryMethod(
        methodName: string,
        before?: (...args: any[]) => void,
        after?: (result: any, ...args: any[]) => void
    ): void {
        const host = this as any;
        const current = host[methodName];
        if (typeof current !== 'function') return;
        if (current.__telemetryWrapped) return;

        const wrapped = (...args: any[]) => {
            if (this.telemetryEnabled && before) {
                try { before(...args); } catch { }
            }
            const result = current.apply(this, args);
            if (this.telemetryEnabled && after) {
                try { after(result, ...args); } catch { }
            }
            return result;
        };

        wrapped.__telemetryWrapped = true;
        wrapped.__telemetryOriginal = current;
        host[methodName] = wrapped;
    }

    private installTelemetryHooks(): void {
        this.wrapTelemetryMethod(
            'update',
            (cpuCycles: number) => {
                const startMs = Date.now();
                const startPerfMs = this.getPerfNowMs();
                this.telemetryRuntime.updateCount += 1;
                this.telemetryRuntime.updateStartAtMs = startMs;
                this.telemetryRuntime.updateStartPerfMs = startPerfMs;
                this.captureCpuCycles(cpuCycles);
            },
            (_result, cpuCycles: number) => {
                const endMs = Date.now();
                const startMs = Number(this.telemetryRuntime.updateStartAtMs || 0);
                const endPerfMs = this.getPerfNowMs();
                const startPerfMs = Number(this.telemetryRuntime.updateStartPerfMs || 0);
                if (startMs > 0 || startPerfMs > 0) {
                    const duration = Math.max(0, startPerfMs > 0 ? (endPerfMs - startPerfMs) :
(endMs - startMs));
                    this.telemetryRuntime.totalUpdateTimeMs += duration;
                    if (duration > this.telemetryRuntime.maxUpdateTimeMs) {
                        this.telemetryRuntime.maxUpdateTimeMs = duration;
```

```
                }
            }
            this.telemetryRuntime.lastUpdateAtMs = endMs;
            this.observeStateMutation('update', cpuCycles, false);
        }
    );
  }
}
```

Architectural Analysis:

- **Decoupled Property Mapping:** The `BaseComponent` constructor dynamically parses the `manifest.json` schemas of components (such as the LDR or MPU6050) to instantiate corresponding pin dictionaries (`this.pins`). This design isolates the frontend canvas coordinate space from the backend logic execution engine.
- **Non-Invasive Method Wrapping:** To track telemetry without altering individual sensor logic files, `wrapTelemetryMethod` creates a functional hook that intercepts calls to standard lifecycle methods (`update`, `onEvent`, `onPinStateChange`). It executes telemetry logging callbacks (`before` and `after`) to count pin toggles, transaction sizes, and cycle delays.
- **State Mutation Tracking:** The `observeStateMutation` method serializes the component state via `safeSerializeState` and compares it against `stateFingerprint` using a time threshold of 45ms. If a change is detected, `this.stateChanged` is set to `true`, prompting the runner to sync updates.

5.4 The Backend Caching & Compilation Controller: `compileController.js`

The `compileController.js` file handles user-submitted C++ code, interacts with backend compiler toolchains, and manages memory caching pools to prevent OOM server failure. Below is the caching architecture code:

```
function buildCompileRequestHash({ code, files, fqbn, builder, libraries_txt }) {
    const payload = {
        code: typeof code === 'string' ? code : '',
        files: stableSourceFiles(files),
        fqbn: String(fqbn || '').trim() || 'arduino:avr:uno',
        builder: String(builder || '').trim() || 'arduino-cli',
        libraries_txt: String(libraries_txt || '').trim(),
    };
    return crypto.createHash('sha1').update(JSON.stringify(payload)).digest('hex');
}

function getCompileResultFromCache(requestHash) {
    if (!requestHash) return null;

    // 1. Try in-memory cache
    pruneCompileResultCache();
```

```
    let hit = compileResultCache.get(requestHash);
    if (hit) {
        // Touch for simple LRU behavior.
        compileResultCache.delete(requestHash);
        compileResultCache.set(requestHash, hit);
        return {
            hex: hit.hex,
            artifactType: hit.artifactType,
            artifactName: hit.artifactName,
            elf: hit.elf,
            elfName: hit.elfName,
            gdb: hit.gdb,
            stdout: hit.stdout,
            diagnostics: hit.diagnostics || null,
        };
    }

    // 2. Try on-disk cache
    const cacheFile = path.join(PICO_UNO_CACHE_ROOT, requestHash, 'result.json');
    if (fs.existsSync(cacheFile)) {
        try {
            const data = JSON.parse(fs.readFileSync(cacheFile, 'utf8'));
            // Touch mtime to mark it as recently used
            const now = new Date();
            fs.utimesSync(cacheFile, now, now);
            fs.utimesSync(path.dirname(cacheFile), now, now);

            // Populate in-memory cache
            compileResultCache.set(requestHash, data);
            return data;
        } catch {
            return null;
        }
    }
    return null;
}

function setCompileResultCache(requestHash, payload) {
    if (!requestHash || !payload || !payload.hex) return;

    const data = {
        createdAt: Date.now(),
        hex: payload.hex,
        artifactType: payload.artifactType || null,
        artifactName: payload.artifactName || null,
        elf: payload.elf || '',
        elfName: payload.elfName || null,
        gdb: payload.gdb || null,
        stdout: payload.stdout || '',
```

```
        diagnostics: payload.diagnostics || null,
    };

    compileResultCache.set(requestHash, data);
    pruneCompileResultCache();

    // Persist to disk and prune
    try {
        const cacheDir = path.join(PICO_UNO_CACHE_ROOT, requestHash);
        fs.mkdirSync(cacheDir, { recursive: true });
        fs.writeFileSync(path.join(cacheDir, 'result.json'), JSON.stringify(data, null, 2),
'utf8');
        pruneUniversalCachePool();
    } catch (e) {
        console.error('[Compile Cache] Failed to write Pico/Uno cache to disk:', e.message);
    }
}
```

Architectural Analysis:

- **Hashed Request Mapping:** The function `buildCompileRequestHash` serializes all compilation parameters—including sanitizing source files using `stableSourceFiles` and sorting them alphabetically—to generate a SHA-1 hash. This ensures that even minor modifications to carriage returns or comments yield distinct hashes, preventing compile mismatches.
- **Two-Tiered Retrieval:** `getCompileResultFromCache` provides low-latency access:
    6. **Level 1 (Memory):** Querying a fast, in-memory JS Map `compileResultCache` that acts as a Least-Recently Used (LRU) cache.
    7. **Level 2 (Filesystem):** If a memory miss occurs, it queries the on-disk directory `PICO_UNO_CACHE_ROOT`. Upon hit, it updates the directory access time using `fs.utimesSync` to prevent it from being pruned by the cache collector.
- **OOM Mitigation:** Pruning routines (`pruneCompileResultCache` and `pruneUniversalCachePool`) enforce strict size constraints (max 120 cached builds in memory) and evict compilation objects older than 30 minutes, keeping memory usage stable during large concurrent class compiles.

Chapter 6: Week-by-Week Technical Logs

This chapter provides a week-by-week breakdown of my technical contributions across the three core repositories of OpenHW Studio from March 1, 2026, through June 30, 2026. Every week highlights specific engineering tasks, code modifications, and architectural decisions.

6.1 Frontend Weekly Logs (openhw-studio-frontend)

Week 1 (March 1 - 7, 2026)

- *Added wire waypoints, rotate button,wire snapping fixed canvas ending bounds added bounds for component boundary definition.*
- *Added bound to define boundary of component added reset buttom,tx,rx led on uno.*
- *Moved context menu for components to emulator folder quick-add search prevention updated ui for components.*
- *Moved context menu for components to emulator folder added collapsible component menu added menu option in canvas added option for zoom in/out added pannable canvas added double-click quick-add menu on canvas improved admin ui dashboard quick-add search prevention added light mode syntax highlighting.*
- *Move the component validation to components folder added option to add custom component zip added in-browser jsx transpilation added sandbox stage for testing new component added live pipline to approve or reject component in admin dashboard.*
- *Added new admin page and dashboard fixed search bar in component panel added live pipline to approve or reject component in admin dashboard added option to uninstall arduino library added polling to admin and simulator page for live upadtes.*
- *Chore: enforce entire component architecture validation in zips.*
- *Fix: add vercel.json to resolve sub-route 404s on page refresh.*
- *Fix: point library management to production url and pre-install adafruit neopixel library.*
- *Fix: point emulator dependency to public github url.*
- *Switched to https.*
- *Fixed error related to deployment.*
- *Added validation for circuit validation updated logic and ui for componenets.*
- *Removed root workspace updated the openhw-studio-frontend-danish package to install @openhw/emulator direct from its github repository branch or use npm link to link it to emulator folder modified execute.ts to add mapping matching reality added getconnectionvoltage so dc motor scan the standard wire connectionto adept their voltage.*
- *Fixed serial plotter added erial parser,label setup implemented overlapping traces & auto-scaling.*
- *Fixed serial monitor printing every character in new line fixed serial monitor input character missing.*
- *Improved ui for serial monitor added png import/export.*

- *Updated logic for servo,motor driver,potentiometer updated logic for serial monitor added serial plotter.*
- *Sync latest component changes onto danish branch.*

Week 2 (March 8 - 14, 2026)

- *Replace useauthstore with useauth in dashboard and signup pages to fix logout.*
- *Switch signinpage to use authcontext instead of authstore.*
- *Unregister old service worker to fix stale cache errors.*
- *Fix infinite navigate loop and auth state uninitialized redirect in protectedroute.jsx.*
- *Fix protectedroute useauth import error.*
- *Fix authprovider import and remove loginpage route in app.jsx.*
- *Feat: add offline caching and project persistence using indexeddb.*
- *Add documentation and descriptions for various components.*

Week 3 (March 15 - 21, 2026)

- *Docs: clarify frontend environment paths.*
- *Fix: update gamification simulator page import path.*
- *Chore: update danish branch with local changes.*
- *Fixed previous issue.*
- *Executed targeted codebase alterations in src/components/wokwi-pushbutton/doc/index.html. The diff analysis recorded: 1 file changed, 41 insertions(+), 24 deletions(-). This commit originally addressed: update.*
- *Fix: update main entry point to src/components/index.ts.*
- *Feat: add web serial hardware support and wire utilities.*
- *Feat: add circuit validation engine with real-world and short-circuit rules, smoke tests, and a new wokwi ldr module component..*
- *Add wokwi ldr and max7219 components with ui and logic.*
- *Save local working state for danish branch.*
- *Save local changes before merging.*

Week 6 (April 5 - 11, 2026)

- *Revert "sagar/work".*
- *Remove old simulatorpage poxy.*
- *Fix all the problem pointend by copilot.*
- *Added option to disable block coding section.*

- *Index on danish: 41e83ee refactor code structure for improved readability and maintainability added support for pico and pico w added login page for normal user added gdb debbuger added f1 menu added support for console ui/ux changes and fixed some bugs.*
- *Refactor code structure for improved readability and maintainability added support for pico and pico w added login page for normal user added gdb debbuger added f1 menu added support for console ui/ux changes and fixed some bugs.*
- *Feat: add wokwi raspberry pi pico component with gpio, uart, and power pin definitions.*

Week 8 (April 19 - 25, 2026)

- *Executed targeted codebase alterations in src/components/logic-mux-2to1/doc.ts, src/components/logic-mux-2to1/index.ts, src/components/max30102/doc.ts and other core modules. The diff analysis recorded: 18 files changed, 134 insertions(+), 1104 deletions(-). This commit originally addressed: Revert "K".*
- *Updated logic for all components.*
- *Chore: start prioritized backend/frontend hardening plan fix: harden component file ops and align backend/frontend defaults fix: finalize hardening for component paths, session secret, port and cors feat: harden auth, route protection, and dynamic render safety fix: tighten authorization messaging and rate-limit cleanup added f1 menu added option to change simulation speed addressed the js/code-injection vulnerability detected by codeql.*
- *Revert "updated the ws logic for the fe".*

Week 9 (April 26 - May 2, 2026)

- *Added ui for mobile.*
- *Added more validation rule added auto fix.*
- *Added quick fix.*
- *Minor chnages for validation.*
- *Added valiadtion engine.*
- *Add built in led to uno.*
- *Added e.stoppropagation() for double-click events.*
- *Fix: simulation is lagy when compoenet is dragged.*
- *Fix: name rotate with component fix: project name not updateing some ui chnages.*
- *Remove register context menu.*
- *Redesign top tool bar added unified glassmorphism to all menu option.*
- *Add:transpired components to indexeddb.*
- *Feat: auto-load permanently installed components at runtime.*
- *Fix: prevent nginx from swallowing backend image requests.*
- *Fix: add missing vite env args to dockerfile.*
- *Added checks to github.*

- *Added pull to sync automatically.*
- *Ready to deploy.*
- *Executed targeted codebase alterations in .github/workflows/notify-dashboard.yml, README.md. The diff analysis recorded: 2 files changed, 27 insertions(+), 6 deletions(-). This commit originally addressed: deplot ready.*
- *Ready to deploy on vm.*
- *Fix issue relatating to casing.*
- *Normalized casing for signuppage.jsx.*
- *Fix : hardware compat matrix.*
- *Added rollback added backend as dependency added nginx.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml. The diff analysis recorded: 1 file changed, 1 insertion(+), 1 deletion(-). This commit originally addressed: fix: repo name.*
- *Added ardunio cli for smoker test.*
- *Change npm ci to install.*
- *Upadated package-lock.json.*
- *Changed to vite 6.*
- *Switch to node 22 resolved dependency @openhw/emulator.*
- *Changed secret to var.*
- *Renamed folder, changes folder name accordingly.*
- *Feat: add deployment workflow and refactor emulator dependency paths in dockerfile and package.json.*
- *Feat: implement gamification engine and auth-protected admin dashboard infrastructure.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml. The diff analysis recorded: 1 file changed, 1 insertion(+). This commit originally addressed: fix deployment.*
- *Feat: integrate ci/cd and admin dashboard.*
- *Fixed signup page not visible.*
- *Fixed issue regrading pin mapping and sign page.*

Week 10 (May 3 - 9, 2026)

- *Fix: telematry issue add: doc.*
- *Add: more telematry to components.*
- *Added: color depending on pin.*
- *Add: autowiring and code for more componets fix: bug realated to telematary.*
- *Add: scale in garding to reduce time fix: buggy wire add: doc for autowire,autofix,autograding.*

- *Add: png export engine fix: auto grading add: new page /grade some minor change remove: mna solver.*
- *Fix issue related to temeplate.*
- *Move to cpu cycle clock instead of wall clock.*
- *Fix: landing page redirect issue.*
- *Executed targeted codebase alterations in src/components/wokwi-7segment/ui.tsx, src/components/wokwi-analog-joystick/ui.tsx, src/components/wokwi-arduino-nano/ui.tsx and other core modules. The diff analysis recorded: 19 files changed, 501 insertions(+), 240 deletions(-). This commit originally addressed: fix: scaling.*
- *Executed targeted codebase alterations in src/components/AutofixPreviewPanel.test.jsx, src/workers/autowiring.worker.ts. The diff analysis recorded: 2 files changed, 2 insertions(+), 2 deletions(-). This commit originally addressed: 0o0o0o0o0o0.*
- *Add prebuilt autowiring wasm wrapper.*
- *Fixed the autowiring worker import resolution by switching the wrapper import to a dynamic import.*
- *Fix: workflow error.*
- *Moved the loading spinner keyframes out of index.html vite workers are forced to es format preview renderer now uses mutable locals in simulatorpage.jsx.*
- *Add: auto grading engine and worker add: /grade webpage for testing.*
- *Add api for telemtary data.*
- *Add: auto grading second phase.*
- *Add:initail page for autograde change engine to rust add: projectutils for chnaging curcuit some ui chnages.*
- *Executed targeted codebase alterations in public/autofix.wasm, src/pages/simulationpage/SimulatorPage.jsx, src/wasm/autofix.wasm and other core modules. The diff analysis recorded: 4 files changed, 70 insertions(+), 54 deletions(-). This commit originally addressed: fix auto fix.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml, src/components/AutofixPreviewPanel.jsx, src/pages/simulationpage/SimulationConsole.jsx and other core modules. The diff analysis recorded: 7 files changed, 264 insertions(+), 95 deletions(-). This commit originally addressed: some changes.*
- *Add: rule to led.*
- *Add: some secret add: some rule.*
- *Remove: autofix engine.*
- *Add: new autofix engine update: autofix panel bug: may be relaed to simulation.*
- *Add: maintainer mode add: nodalsolver update: workflow update: simulation related engine fix: breadboard issuer redesign admin page fix: catch().*

- *Fix: all gnd same fix: led logi update: some logic.*
- *Fix: png export update: admin dashboard fix: security issue.*
- *Remove dynamic ui of uno.*
- *Remove path filter.*
- *Add: dependency for linux.*
- *Addd: fall back for report.*
- *Executed targeted codebase alterations in .github/workflows/page-validation.yml. The diff analysis recorded: 1 file changed, 8 insertions(+). This commit originally addressed: finding error.*
- *Fix:comment issue.*
- *Fix:cache cause issue.*
- *Detailed report of workflow.*
- *Add:url btn of runner.*
- *Executed targeted codebase alterations in .github/workflows/autofix-ci.yml, .github/workflows/emulator-phased-validation.yml. The diff analysis recorded: 2 files changed, 3 insertions(+), 3 deletions(-). This commit originally addressed: oooooo.*
- *Add: cli and backend repo to workflow.*
- *Add: cache to workflow.*
- *Executed targeted codebase alterations in .github/workflows/autofix-ci.yml, .github/workflows/emulator-phased-validation.yml, src/circuit-validation/engine.js and other core modules. The diff analysis recorded: 4 files changed, 22 insertions(+), 10 deletions(-). This commit originally addressed: fix workflow.*
- *Fix landing page redirect.*
- *Move to vite 6.2.0.*
- *Executed targeted codebase alterations in .github/workflows/pr-checks.yml, src/pages/mobileui/SimulatorPage.jsx. The diff analysis recorded: 2 files changed, 17 insertions(+), 7 deletions(-). This commit originally addressed: ooooo.*
- *Fix: rollup failed to resolve import "@openhw/emulator".*
- *Executed targeted codebase alterations in .github/workflows/page-validation.yml, .github/workflows/pr-checks.yml. The diff analysis recorded: 2 files changed, 4 insertions(+), 4 deletions(-). This commit originally addressed: last.*
- *Add: rollup native package.*
- *Do not even know.*
- *Add: missing lightingcss package.*
- *Fix: nestead directaries.*
- *Add: install of @rollup/rollup-linux-x64-gnu@4.60.2.*

- *Solve problem regrading nested cd.*
- *Executed targeted codebase alterations in .github/workflows/pr-checks.yml. The diff analysis recorded: 1 file changed, 1 insertion(+), 1 deletion(-). This commit originally addressed: check workflow.*
- *Executed targeted codebase alterations in .github/workflows/pr-checks.yml. The diff analysis recorded: 1 file changed, 8 insertions(+), 11 deletions(-). This commit originally addressed: fix: workflow.*
- *Fix triple-path issue in workflows by removing job-level working-directory defaults.*
- *Added: more test to pr add: new page alignment-lab add: scroll in simualtion canvas.*
- *Added write permission to comment.*
- *Fix: menu not opening add: scoket helper to snap.*
- *Led autowiring,autocode completed.*
- *Add: page validation on pr added initial autocodieng autowiring.*
- *Added test for pr.*
- *Added: more rules add: circuit fixer add: breadboard snap.*
- *Added auto fix panel add: breadboard snap add: some test for github action fix: minor ui problem.*

Week 11 (May 10 - 16, 2026)

- *Remove 'wokwi-lcd1602-i2c' from execute.ts.*
- *Fix: lost data on refresh.*
- *Fix: wire endpoint mismatch.*
- *Renaming phase 5 (all completed) add: import zip from wokwi.*
- *Build wasm and copied.*
- *Renaming phase 4.*
- *Renaming phase 3.*
- *Executed targeted codebase alterations in remote_page.jsx, src/pages/ProjectAssessmentPage.jsx, src/pages/mobileui/SimulatorPage.jsx and other core modules. The diff analysis recorded: 8 files changed, 128 insertions(+), 10 deletions(-). This commit originally addressed: rename phase 2.*
- *Rename some components.*
- *Fix: right panel blinking effect fix: valiadtion error fix: increased z index of interactable components fix: refresh delete code some ui changes: libaray,right panle max width:1100,exploer:200.*
- *Refactor: routes for export,hardware,registery.*
- *Refactor rightpanel.*
- *Fix: components panel laggy animation.*

- *Make projectsidebar overlay.*
- *Executed targeted codebase alterations in src/pages/simulationpage/SimulatorPage.jsx. The diff analysis recorded: 1 file changed, 83 insertions(+), 32 deletions(-). This commit originally addressed: wire problem.*
- *Fix: ultrasonic.*
- *Executed targeted codebase alterations in src/pages/simulationpage/SimulatorPage.jsx. The diff analysis recorded: 1 file changed, 104 insertions(+), 53 deletions(-). This commit originally addressed: png maybe fix?.*
- *Moved to iframe.*
- *Fix: png export time.*
- *Fix: css of project sidebar.*
- *Refactor: projectsidebar,canvas bottom menu,f1 menu,gamification.*
- *Executed targeted codebase alterations in src/pages/simulationpage/SimulatorPage.jsx, .../simulationpage/components/CanvasPrimitives.jsx, .../simulationpage/components/CanvasSceneLayer.jsx and other core modules. The diff analysis recorded: 8 files changed, 866 insertions(+), 640 deletions(-). This commit originally addressed: fix phase 4.*
- *Move to dynamic interaction.*
- *Add: tour guide.*
- *Fix; rpoject panel.*
- *Executed targeted codebase alterations in src/pages/simulationpage/SimulatorPage.jsx. The diff analysis recorded: 1 file changed, 254 insertions(+), 14 deletions(-). This commit originally addressed: thanks gemini.*
- *Executed targeted codebase alterations in src/pages/simulationpage/SimulatorPage.jsx. The diff analysis recorded: 1 file changed, 29 insertions(+), 1 deletion(-). This commit originally addressed: little fix.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml, src/App.jsx, .../teacher/class-detail/StudentGradingPanel.jsx and other core modules. The diff analysis recorded: 24 files changed, 2185 insertions(+), 1535 deletions(-). This commit originally addressed: broken.*
- *Upadate: deploy workflow.*
- *Add: monaco ide fix: shortcut conflicts.*
- *Fix: docs url move: littlefs to public.*
- *Executed targeted codebase alterations in .env.example, .github/workflows/deploy.yml, .github/workflows/frontend-tests.yml and other core modules. The diff analysis recorded: 16 files changed, 58 insertions(+), 797 deletions(-). This commit originally addressed: add: docs.*
- *Add: keyboard shortcut.*
- *Fix: lock file and rollup dependency.*

- *Fix; roll up dependency.*
- *----------.*
- *Update: serial ui fix: workflow related problem.*
- *Fix build and test resolution: use vite-node with inlined deps for tests and emulator alias for workers.*
- *Fix build and test resolution: use vite-node for tests and emulator alias for workers.*
- *Fix build resolution: allow wasm files and use emulator alias in workers.*
- *Fix: gitignore was ignring wasm files.*
- *Before palette refactor.*
- *Add: new 2 pin ldr sensor add: ntc temp sensor.*
- *Add: bg color to sub menu refactor: autofix.*
- *Add: autowire for more components.*
- *Fix: runner can not create issue.*
- *Add: support for more components in autofix add: owner tag in json remove: autosetup.js move all code to projectutils.js.*
- *Remove: context menu.*
- *Add: componentcontext panel.*
- *Remove: auto fix validation.*
- *Remove: png download wasm.*
- *Add: telegarm notification.*
- *Add: telegram notification.*

Week 12 (May 17 - 23, 2026)

- *Executed targeted codebase alterations in src/pages/simulationpage/services/TelemetryManager.js, src/pages/simulationpage/utils/telemetryRegistry.js. The diff analysis recorded: 2 files changed, 5 insertions(+), 4 deletions(-). This commit originally addressed: ----------.*
- *Added: toaster nofication when url open in browser.*
- *Fix: example repo import.*
- *Changes in wire routes move exit side to f1 menu remove exit side overlay related code.*
- *Added grou[ing to wiring.*
- *Fix page validation check error.*
- *Executed targeted codebase alterations in src/App.jsx. The diff analysis recorded: 1 file changed, 25 insertions(+), 25 deletions(-). This commit originally addressed: fix lazy load.*
- *Fix aboutus page.*
- *Fix cli issue runner refactor completed.*

- *Refactor runner phase 6.*
- *Refactor runner phase 5.*
- *Refactor runner pahse 4 (completed).*
- *Refactor runner phase 3.*
- *Refactor phase 2.*
- *Runner refactor phase 1.*
- *Wriring first type.*
- *Fix height of validation engine fix: telematry for each components add fallback for more wokwi components.*
- *Add debug for each components add option in menu for more attr.*

Week 13 (May 24 - 30, 2026)

- *Fix: pinnot propagating in proxyrunner added map for stm32 in autowiring.*
- *Add stm32 mapped pin for stm32 in autowiring.*
- *Merge sharukh changes.*
- *Fix import path.*
- *Fix emualator path.*
- *Sync backend and frontend timer for esp add worker for display render fix: pico timer fix autowire for esp spi pin.*
- *Fix autowiring add log for spi on pico.*
- *Fix some spi leak to frontend and doen some work on wiring.*
- *Fix esp added ghost runner for esp added esp cam fix autowiring code for esp added telematry for esp.*
- *Added telematry to esp added runner worker for esp fix: esp pin was not tracing wire.*
- *Added bfs for wire.*
- *Fix: autowiring for esp remove: board selection from toptoolbox minor fix in registory and esp engine.*
- *Chnages related to telematry.*

Week 14 (May 31 - June 6, 2026)

- *Add config for live simulation.*
- *Fix nginx hijax fix abckend port to 5000.*
- *Fix login redirect.*
- *Feat: dynamic port forwarding, ip scaling, and public/private gateway separation and log for it.*
- *Change url for gateway.*
- *Change private gateway url.*

- *Fix gateway url.*
- *Move oauth verification to backend fix jwt expire error.*
- *Enbaled sharedarraybuffer aslo fix automatic logout.*
- *Fix photoresistor import.*
- *Add pico-w wifi chip emulation code.*
- *Fix esp wifi added room feature to gateway add network panel and wifi overlay.*
- *Fix qemu running simaution even when selected frontend.*
- *Executed targeted codebase alterations in src/worker/runners/esp32-runner.ts. The diff analysis recorded: 1 file changed, 9 insertions(+), 5 deletions(-). This commit originally addressed: fix ideal time.*
- *Fix esp frontend engine bug engine is slow(migarte to wasm in future) added lot of protocal.*
- *Added vm recousce monitor in admin panel.*
- *Fix json parsing issue.*
- *Fix wrong port assinged.*
- *Added retry on backend error.*
- *Fix: execasync typo, renode isread case, libsdl2/net-tools, nginx routing.*
- *Executed targeted codebase alterations in nginx.conf. The diff analysis recorded: 1 file changed, 21 insertions(+). This commit originally addressed: fix see issue.*
- *Log will show docker name.*
- *Redesign admin dashboard.*
- *Upadate lock file.*
- *Added worker for network experiment for frontend esp engine added option to choose esp engine in f1 menu.*
- *Fix forever block.*
- *Added hook to reset use block flag.*
- *Reset all block flag when it is disabled.*
- *Fix block code is not genrating code.*
- *Add resource budget tab in admin dashboard.*
- *Changes how liabary work chnages libaray panel added cache for compiled binary.*

Week 15 (June 7 - 13, 2026)

- *Rebuild esp engine wasm next commit diffrent approch.*
- *Picow completed added : keep alive hack remove log for picow.*
- *Phase 3d (internet issue).*
- *Phase 3c (wifi connected).*

- *Phase 3b (booted).*
- *Revert "feat: add examples gallery page with filtering, thumbnails, and simul…".*
- *Executed targeted codebase alterations in src/worker/runners/rp2040-runner.ts. The diff analysis recorded: 1 file changed, 3 insertions(+), 3 deletions(-). This commit originally addressed: phase 3.*
- *Next commit will fallback to another method.*
- *Executed targeted codebase alterations in public/xtensa-core.wasm, src/worker/runners/esp32-runner.ts. The diff analysis recorded: 2 files changed, 63 insertions(+), 9 deletions(-). This commit originally addressed: phase 3b.*
- *Addinf wasm engine for esp.*
- *Fix clock for esp.*
- *Feat: sharedbuffer for display.*
- *Fix esp ssid issue.*
- *Fix liveclass room ws url.*
- *Add pre-fetch for worker runner.*
- *Add: earlyhardwaremessages queue to fiix data got lost when starting worker.*
- *Increase websocket timeout.*
- *Add comment for cache bust.*

Week 16 (June 14 - 20, 2026)

- *Remove gateway adress,add ip and port address.*
- *Remove esp frontend code due to copyright issue.*

Week 17 (June 21 - 27, 2026)

- *Changed esp32 engine added ir component some bug fix add option for custom uild server.*

Week 18 (June 28 - July 4, 2026)

- *Added support for rv32.*

Week 19 (July 5 - 11, 2026)

- *Revert "ui redesign".*

6.2 Emulator Weekly Logs (openhw-studio-emulator)

Week 1 (March 1 - 7, 2026)

- *Added bound to define boundary of component added reset buttom,tx,rx led on uno.*
- *Added wire waypoints, rotate button,wire snapping fixed canvas ending bounds added bounds for component boundary definition.*

- *Moved context menu for components to emulator folder quick-add search prevention updated ui for components.*
- *Double-click search prevention added light mode syntax highlighting updated ui for components.*
- *Double-click quick-add menu on canvas change context menu ui for component improved admin ui dashboard.*
- *Moved context menu for components to ui.tsx added collapsible component menu added option for zoom in/out.*
- *Move the component validation to components folder added option to add custom component zip added in-browser jsx transpilation added sandbox stage for testing new component added live pipline to approve or reject component in admin dashboard.*
- *Executed targeted codebase alterations in README.md. The diff analysis recorded: 1 file changed, 21 insertions(+). This commit originally addressed: update readme.*
- *Modular design for components move the component validation to components folder fixed search bar in component panel added option to add custom component zip added a new admin page added in-browser jsx transpilation added sandbox stage for testing new component added live pipline to approve or reject component in admin dashboard added option to uninstall arduino library added polling to admin and simulator page for live upadtes added options to backup/import for components.*
- *Added validation for circuit validation updated logic and ui for componenets.*
- *Merge with commit b30e65c of openhw-studio-backend added validation for circuit validation added dockerfile for deployment.*
- *Implemented chached compilation web worker use already available hex file, bypassing the backend compiler completely. changed the flag from --build-path to --output-dir to trick arduino-cli to use its permanent global cache system for the core files properly mapped pin for componenet.*
- *Removed root workspace updated the openhw-studio-frontend-danish package to install @openhw/emulator direct from its github repository branch or use npm link to link it to emulator folder.*
- *Modified execute.ts to add mapping matching reality added getconnectionvoltage so dc motor scan the standard wire connectionto adept their voltage.*
- *Updated both potentiometer ui wrapper to directly render web component with a uselayouteffect to attach a native dom listener.*
- *Added animation loop in update fuction.*
- *Fixed wss2812b the wrapper now takes the 24-bit hexadecimal colors fixed the matrix dimension propagation to parse strings into base-10 integers prior to rendering..*
- *Updated logic for servo,motor driver,potentiometer updated logic for serial monitor added serial plotter.*
- *Added png expoet/import added i2c,spi communication (see docs for more).*
- *Sync latest component changes onto danish branch.*

Week 2 (March 8 - 14, 2026)

- *Add documentation and descriptions for various components.*
- *Add documentation for various components in openhw studio.*

Week 3 (March 15 - 21, 2026)

- *Fixed previous issue.*
- *Executed targeted codebase alterations in src/components/wokwi-pushbutton/doc/index.html. The diff analysis recorded: 1 file changed, 41 insertions(+), 24 deletions(-). This commit originally addressed: update.*
- *Fix: update main entry point to src/components/index.ts.*
- *Add new examples and evaluation criteria for various arduino projects add multiboard support.*
- *Feat: add circuit validation engine with real-world and short-circuit rules, smoke tests, and a new wokwi ldr module component..*
- *Add wokwi ldr and max7219 components with ui and logic.*

Week 6 (April 5 - 11, 2026)

- *Revert "sorry for the lots of changes ".*
- *Feat: add wokwi raspberry pi pico component with gpio, uart, and power pin definitions.*

Week 8 (April 19 - 25, 2026)

- *Executed targeted codebase alterations in src/components/logic-mux-2to1/doc.ts, src/components/logic-mux-2to1/index.ts, src/components/max30102/doc.ts and other core modules. The diff analysis recorded: 18 files changed, 134 insertions(+), 1104 deletions(-). This commit originally addressed: Revert "K".*
- *Updated logic for all components.*

Week 9 (April 26 - May 2, 2026)

- *Added more validation rule added auto fix.*
- *Added valiadtion engine.*
- *Add built in led to uno.*
- *Remove register context menu.*
- *Executed targeted codebase alterations in .github/workflows/notify-dashboard.yml, README.md. The diff analysis recorded: 2 files changed, 27 insertions(+), 6 deletions(-). This commit originally addressed: deplot ready.*

Week 10 (May 3 - 9, 2026)

- *Add: more telematry to components.*
- *Add: autowiring and code for more componets fix: bug realated to telematary.*
- *Fix issue related to temeplate.*

- *Executed targeted codebase alterations in src/components/wokwi-7segment/ui.tsx, src/components/wokwi-analog-joystick/ui.tsx, src/components/wokwi-arduino-nano/ui.tsx and other core modules. The diff analysis recorded: 19 files changed, 501 insertions(+), 240 deletions(-). This commit originally addressed: fix: scaling.*
- *Add api for telemtary data.*
- *Add: rule to led.*
- *Add: some secret add: some rule.*
- *Remove: autofix engine.*
- *Fix: all gnd same fix: led logi update: some logic.*
- *Remove dynamic ui of uno.*
- *Remove path filter.*
- *Add:url btn of runner.*
- *Executed targeted codebase alterations in .github/workflows/autofix-ci.yml, .github/workflows/emulator-phased-validation.yml. The diff analysis recorded: 2 files changed, 3 insertions(+), 3 deletions(-). This commit originally addressed: oooooo.*
- *Add: cli and backend repo to workflow.*
- *Executed targeted codebase alterations in .github/workflows/autofix-ci.yml, .github/workflows/emulator-phased-validation.yml, src/circuit-validation/engine.js and other core modules. The diff analysis recorded: 4 files changed, 22 insertions(+), 10 deletions(-). This commit originally addressed: fix workflow.*
- *Added write permission to comment.*
- *Added test for pr.*
- *Added: more rules add: circuit fixer add: breadboard snap.*

Week 11 (May 10 - 16, 2026)

- *Renaming phase 5 add: lcd 16x2.*
- *Renaming phase 4.*
- *Renaming phase 2.*
- *Executed targeted codebase alterations in src/components/index.ts, src/components/wokwi-cd74hc4067/manifest.json, src/components/wokwi-ldr-module/manifest.json and other core modules. The diff analysis recorded: 9 files changed, 16 insertions(+), 16 deletions(-). This commit originally addressed: rename phase 2.*
- *Fix: validation fix: servo,potentiometer rename:some components.*
- *Lot of ui and manifest changes.*
- *Fix: validation api.*
- *Executed targeted codebase alterations in src/circuit-validation/AUTOFIX_ARCHITECTURE.md, src/circuit-validation/AUTOFIX_IMPROVEM*

ENTS.md. The diff analysis recorded: 2 files changed, 509 deletions(-). This commit originally addressed: remove: docs.

- *Remove: autofix test.*
- *Add: new 2 pin ldr sensor add: ntc temp sensor.*
- *Add: autowire for more components.*
- *Remove: context menu.*
- *Remove: auto fix validation.*
- *Add: telegram notification.*

Week 12 (May 17 - 23, 2026)

- *Fix: ws2812b color bug.*
- *Add autowire and autocode for more components add new component.*
- *Fix 15px distance and also other components.*
- *Fix cli issue runner refactor completed.*
- *Refactor runner phase 6.*
- *Refactor runner phase 5.*
- *Added manas components.*
- *Fix autocodeing and autowiring for lcd fix: valiadation engine.*
- *Fix valiadtion error add 4 pin ush button add attribute.*

Week 13 (May 24 - 30, 2026)

- *Ili9341 logic change and also fix rotation in autocode.*
- *Add stm32 blueprint.*
- *Add seperate display worker and renderer for ili9341 and oled remeber to add neopixel and dot matrix in future.*
- *Fix: tft display autocodeing fix i2c.*
- *Revert "update src/components/index.ts".*
- *Fix esp added esp cam added telematry for esp add esp to validation.*
- *Change prefix with openhw.*
- *Fix:pwm traffic.*

Week 14 (May 31 - June 6, 2026)

- *Add pico-w chip emulation code.*
- *Executed targeted codebase alterations in src/components/openhw-ssd1306-oled/logic.ts. The diff analysis recorded: 1 file changed, 1 insertion(+), 1 deletion(-). This commit originally addressed: remove log.*
- *Tft lcd sync at 30fps.*

- *Added lot of protocal.*
- *Adding wifi environment for esp and pico w added openhw ap component.*
- *Minor ui chnaged.*

Week 15 (June 7 - 13, 2026)

- *Picow completed.*
- *Phase 3f (log removed).*
- *Phase 3e (intenet fixed).*
- *Phase 3d (internet issue).*
- *Phase 3c (wifi connected).*
- *Phase 3b (booted).*
- *Executed targeted codebase alterations in src/components/openhw-pico-w/logic.ts. The diff analysis recorded: 1 file changed, 119 insertions(+), 165 deletions(-). This commit originally addressed: phase 3.*
- *Executed targeted codebase alterations in .../openhw-pico-w/cyw43/Cyw43Emulator.ts, .../openhw-pico-w/cyw43/PioBusSniffer.ts, src/components/openhw-pico-w/logic.ts and other core modules. The diff analysis recorded: 3 files changed, 115 insertions(+), 75 deletions(-). This commit originally addressed: phase 2c.*
- *Executed targeted codebase alterations in .../openhw-pico-w/cyw43/Cyw43Emulator.ts, .../openhw-pico-w/cyw43/PioBusSniffer.ts, src/components/openhw-pico-w/logic.ts and other core modules. The diff analysis recorded: 3 files changed, 44 insertions(+), 8 deletions(-). This commit originally addressed: phase 2b.*
- *Executed targeted codebase alterations in .../openhw-pico-w/cyw43/PioBusSniffer.ts, src/components/openhw-pico-w/logic.ts. The diff analysis recorded: 2 files changed, 94 insertions(+), 36 deletions(-). This commit originally addressed: phase 2.*
- *Next commit will fallback to another method.*
- *Add stm frontend move display render to sharedmemory.*
- *Add debounce logic to lcd1604 and lcd2004.*

Week 16 (June 14 - 20, 2026)

- *Executed targeted codebase alterations in src/components/openhw-pico-w/logic.ts. The diff analysis recorded: 1 file changed, 40 insertions(+), 2 deletions(-). This commit originally addressed: added ip logic.*

Week 17 (June 21 - 27, 2026)

- *Added ir components.*

Week 18 (June 28 - July 4, 2026)

- *Added support for rv32.*

6.3 Backend Weekly Logs (openhw-studio-backend)

Week 1 (March 1 - 7, 2026)

- *Added wire waypoints, rotate button,wire snapping fixed canvas ending bounds added bounds for component boundary definition.*
- *Added bound to define boundary of component added reset buttom,tx,rx led on uno.*
- *Added index folder for server data.*
- *Fixed 505 added logic to handle missing index.ts.*
- *Emulator_components_path added startup initialization logic to ensure that both the temp and data/components directories exist updated to pre-create the data/components.*
- *Moved context menu for components to emulator folder quick-add search prevention updated ui for components.*
- *Moved context menu for components to emulator folder added collapsible component menu added menu option in canvas added option for zoom in/out added pannable canvas added double-click quick-add menu on canvas improved admin ui dashboard quick-add search prevention added light mode syntax highlighting.*
- *Added live pipline to approve or reject component in admin dashboard added in-browser jsx transpilation added sandbox stage for testing new component.*
- *Move the component validation to components folder added option to add custom component zip added in-browser jsx transpilation added sandbox stage for testing new component added live pipline to approve or reject component in admin dashboard.*
- *Added new admin page and dashboard fixed search bar in component panel added live pipline to approve or reject component in admin dashboard added option to uninstall arduino library added polling to admin and simulator page for live upadtes.*
- *Chore: enforce entire component architecture validation in zips.*
- *Fix: add vercel.json to resolve sub-route 404s on page refresh.*
- *Fix: pre-install common libraries (neopixel, stepper, servo) in docker image.*
- *Fix: point library management to production url and pre-install adafruit neopixel library.*
- *Fix: point emulator dependency to public github url.*
- *Switched to https.*
- *Fixed error related to deployment.*
- *Added validation for circuit validation updated logic and ui for componenets.*
- *Removed root workspace updated the openhw-studio-frontend-danish package to install @openhw/emulator direct from its github repository branch or use npm link to link it to emulator folder modified execute.ts to add mapping matching reality added getconnectionvoltage so dc motor scan the standard wire connectionto adept their voltage.*
- *Updated server.js to fall back to the environment variables provided by render's dashboard if the local file is missing.*

- *Merge with commit b30e65c of openhw-studio-backend implemented chached compilation web worker use already available hex file, bypassing the backend compiler completely. changed the flag from --build-path to --output-dir to trick arduino-cli to use its permanent global cache system for the core files properly mapped pin for componenet.*
- *Merge with commit b30e65c of openhw-studio-backend.*
- *Fixed serial plotter added erial parser,label setup implemented overlapping traces & auto-scaling.*
- *Fixed serial monitor printing every character in new line fixed serial monitor input character missing.*
- *Remove commented project structure section from readme.*
- *Executed targeted codebase alterations in README.md. The diff analysis recorded: 1 file changed, 2 insertions(+), 2 deletions(-). This commit originally addressed: update readme.*
- *Improved ui for serial monitor added png import/export.*
- *Updated logic for servo,motor driver,potentiometer updated logic for serial monitor added serial plotter.*
- *Sync latest component changes onto danish branch.*

Week 2 (March 8 - 14, 2026)

- *Feat: enhance cors configuration and improve security in usercontroller.*
- *Refactor: remove old classroom and component features while adding static file serving for examples. merge aaditya and sager repo.*
- *Replace useauthstore with useauth in dashboard and signup pages to fix logout.*
- *Switch signinpage to use authcontext instead of authstore.*
- *Unregister old service worker to fix stale cache errors.*
- *Fix infinite navigate loop and auth state uninitialized redirect in protectedroute.jsx.*
- *Fix protectedroute useauth import error.*
- *Fix /api/compile/compile 404 error by mapping to /.*
- *Fix authprovider import and remove loginpage route in app.jsx.*
- *Fix: skip non-file entries when backing up installed components.*
- *Feat: add offline caching and project persistence using indexeddb.*
- *Add documentation and descriptions for various components.*

Week 3 (March 15 - 21, 2026)

- *Fix: remove hardcoded env-file startup flag.*
- *Fix: keep backend running without mongodb.*
- *Docs: clarify frontend environment paths.*
- *Fix: make example and emulator paths portable.*
- *Fix: update gamification simulator page import path.*

- *Executed targeted codebase alterations in remotes.txt. The diff analysis recorded: 1 file changed, 4 deletions(-). This commit originally addressed: remove file.*
- *Executed targeted codebase alterations in .env.example, README.md, branch_commits.txt and other core modules. The diff analysis recorded: 14 files changed, 268 insertions(+), 128 deletions(-). This commit originally addressed: fixed.*
- *Index on develop: f6edd5f merge branch feature/google-auth into develop.*
- *Chore: update danish branch with local changes.*
- *Fixed previous issue.*
- *Executed targeted codebase alterations in src/components/wokwi-pushbutton/doc/index.html. The diff analysis recorded: 1 file changed, 41 insertions(+), 24 deletions(-). This commit originally addressed: update.*
- *Fix: update main entry point to src/components/index.ts.*
- *Feat: add web serial hardware support and wire utilities.*
- *Added suport for multiple boards during simulation.*
- *Feat: add circuit validation engine with real-world and short-circuit rules, smoke tests, and a new wokwi ldr module component..*
- *Add wokwi ldr and max7219 components with ui and logic.*
- *Save local working state for danish branch.*
- *Save local changes before merging.*

Week 6 (April 5 - 11, 2026)

- *Remove old simulatorpage poxy.*
- *Resolved all the problem pointend by copilot.*
- *Fix all the problem pointend by copilot.*
- *Added option to disable block coding section.*
- *Refactor code structure for improved readability and maintainability added support for pico and pico w added login page for normal user added gdb debbuger added f1 menu added support for console ui/ux changes and fixed some bugs.*
- *Feat: enhance user role management and update compile routes.*
- *Feat: add wokwi raspberry pi pico component with gpio, uart, and power pin definitions.*

Week 8 (April 19 - 25, 2026)

- *Executed targeted codebase alterations in src/components/logic-mux-2to1/doc.ts, src/components/logic-mux-2to1/index.ts, src/components/max30102/doc.ts and other core modules. The diff analysis recorded: 18 files changed, 134 insertions(+), 1104 deletions(-). This commit originally addressed: Revert "K".*
- *Updated logic for all components.*

- *Chore: start prioritized backend/frontend hardening plan fix: harden component file ops and align backend/frontend defaults fix: finalize hardening for component paths, session secret, port and cors feat: harden auth, route protection, and dynamic render safety fix: tighten authorization messaging and rate-limit cleanup added f1 menu added option to change simulation speed addressed the js/code-injection vulnerability detected by codeql.*
- *Chore: start prioritized backend/frontend hardening plan fix: harden component file ops and align backend/frontend defaults fix: finalize hardening for component paths, session secret, port and cors feat: harden auth, route protection, and dynamic render safety fix: tighten authorization messaging and rate-limit cleanup.*
- *Revert "added the ws logic for be".*

Week 9 (April 26 - May 2, 2026)

- *Added ui for mobile.*
- *Added more validation rule added auto fix.*
- *Executed targeted codebase alterations in src/controllers/autofixController.js, src/routes/api.js. The diff analysis recorded: 2 files changed, 36 insertions(+). This commit originally addressed: added auto fix.*
- *Added quick fix.*
- *Minor chnages for validation.*
- *Added valiadtion engine.*
- *Add built in led to uno.*
- *Added e.stoppropagation() for double-click events.*
- *Fix: simulation is lagy when compoenet is dragged.*
- *Fix: name rotate with component fix: project name not updateing some ui chnages.*
- *Remove register context menu.*
- *Add: process.env.allowed_origins.*
- *Redesign top tool bar added unified glassmorphism to all menu option.*
- *Add:transpired components to indexeddb.*
- *Feat: auto-load permanently installed components at runtime.*
- *Feat: add public components endpoint for runtime injection.*
- *Fix: resolve all docker container filesystem paths.*
- *Fix: resolve correct emulator components path for docker.*
- *Fix: prevent nginx from swallowing backend image requests.*
- *Fix: resolve correct examples directory path for docker.*
- *Feat: auto-upgrade admin users based on env variable.*
- *Fix: add missing vite env args to dockerfile.*
- *Fix: map frontend directly to port 80.*

- *Fix: remove --env-file flag from start script.*
- *Remove adafruit bme280 library.*
- *Add: check to github.*
- *Added checks to github.*
- *Added pull to sync automatically.*
- *Added to pull to sync automatically.*
- *Ready to deploy.*
- *Executed targeted codebase alterations in .github/workflows/notify-dashboard.yml, README.md. The diff analysis recorded: 2 files changed, 27 insertions(+), 6 deletions(-). This commit originally addressed: deplot ready.*
- *Ready to deploy on vm.*
- *Ready to deploy on docker.*
- *Fix issue relatating to casing.*
- *Normalized casing for signuppage.jsx.*
- *Fix : hardware compat matrix.*
- *Added rollback added backend as dependency added nginx.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml. The diff analysis recorded: 1 file changed, 1 insertion(+), 1 deletion(-). This commit originally addressed: fix: repo name.*
- *Added ardunio cli for smoker test.*
- *Change npm ci to install.*
- *Upadated package-lock.json.*
- *Removed force flag.*
- *Changed to vite 6.*
- *Switch to node 22 resolved dependency @openhw/emulator.*
- *Changed secret to var.*
- *Renamed folder, changes folder name accordingly.*
- *Feat: implement component controller with lifecycle management for submitting, approving, and installing emulator components..*
- *Feat: implement live simulation management, github deployment orchestration, and backend routing structure.*
- *Feat: add deployment workflow and refactor emulator dependency paths in dockerfile and package.json.*
- *Feat: implement gamification engine and auth-protected admin dashboard infrastructure.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml. The diff analysis recorded: 1 file changed, 1 insertion(+). This commit originally addressed: fix deployment.*

- *Fix deployement.*
- *Feat: integrate ci/cd and admin dashboard.*
- *Fixed signup page not visible.*
- *Fixed issue regrading pin mapping and sign page.*

Week 10 (May 3 - 9, 2026)

- *Fix: telematry issue add: doc.*
- *Add: more telematry to components.*
- *Added: color depending on pin.*
- *Add: autowiring and code for more componets fix: bug realated to telematary.*
- *Add: scale in garding to reduce time fix: buggy wire add: doc for autowire,autofix,autograding.*
- *Add: png export engine fix: auto grading add: new page /grade some minor change remove: mna solver.*
- *Fix issue related to temeplate.*
- *Move to cpu cycle clock instead of wall clock.*
- *Fix: conflict role with admin.*
- *Fix: landing page redirect issue.*
- *Executed targeted codebase alterations in src/components/wokwi-7segment/ui.tsx, src/components/wokwi-analog-joystick/ui.tsx, src/components/wokwi-arduino-nano/ui.tsx and other core modules. The diff analysis recorded: 19 files changed, 501 insertions(+), 240 deletions(-). This commit originally addressed: fix: scaling.*
- *Executed targeted codebase alterations in src/components/AutofixPreviewPanel.test.jsx, src/workers/autowiring.worker.ts. The diff analysis recorded: 2 files changed, 2 insertions(+), 2 deletions(-). This commit originally addressed: 0o0o0o0o0o0.*
- *Add prebuilt autowiring wasm wrapper.*
- *Fixed the autowiring worker import resolution by switching the wrapper import to a dynamic import.*
- *Fix: workflow error.*
- *Moved the loading spinner keyframes out of index.html vite workers are forced to es format preview renderer now uses mutable locals in simulatorpage.jsx.*
- *Upadtate: lock file to match package file.*
- *Add: auto grading engine and worker add: /grade webpage for testing.*
- *Fix:admin page notification fix.*
- *Add api for telemtary data.*
- *Add: auto grading second phase.*

- *Add:initail page for autograde change engine to rust add: projectutils for chnaging curcuit some ui chnages.*
- *Executed targeted codebase alterations in public/autofix.wasm, src/pages/simulationpage/SimulatorPage.jsx, src/wasm/autofix.wasm and other core modules. The diff analysis recorded: 4 files changed, 70 insertions(+), 54 deletions(-). This commit originally addressed: fix auto fix.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml, src/components/AutofixPreviewPanel.jsx, src/pages/simulationpage/SimulationConsole.jsx and other core modules. The diff analysis recorded: 7 files changed, 264 insertions(+), 95 deletions(-). This commit originally addressed: some changes.*
- *Add: rule to led.*
- *Executed targeted codebase alterations in .github/workflows/deploy.yml. The diff analysis recorded: 1 file changed, 1 insertion(+), 1 deletion(-). This commit originally addressed: added secret.*
- *Add: some secret add: some rule.*
- *Remove: autofix engine.*
- *Add: new autofix engine update: autofix panel bug: may be relaed to simulation.*
- *Add: maintainer mode add: nodalsolver update: workflow update: simulation related engine fix: breadboard issuer redesign admin page fix: catch().*
- *Add: notify api to admin update: admin dashboard add: me as codeowner.*
- *Fix: all gnd same fix: led logi update: some logic.*
- *Fix: png export update: admin dashboard fix: security issue.*
- *Add: proper api to communicate with vm fix: some secuity issue.*
- *Remove dynamic ui of uno.*
- *Remove path filter.*
- *Remove pathh filter.*
- *Add: dependency for linux.*
- *Addd: fall back for report.*
- *Executed targeted codebase alterations in .github/workflows/page-validation.yml. The diff analysis recorded: 1 file changed, 8 insertions(+). This commit originally addressed: finding error.*
- *Fix:comment issue.*
- *Fix:cache cause issue.*
- *Detailed report of workflow.*
- *Add: url and detail comment.*
- *Add:url btn of runner.*
- *Executed targeted codebase alterations in .github/workflows/autofix-ci.yml, .github/workflows/emulator-phased-validation.yml. The diff*

analysis recorded: 2 files changed, 3 insertions(+), 3 deletions(-). This commit originally addressed: oooooo.

- *Add: cli and backend repo to workflow.*
- *Add: cache to workflow.*
- *Executed targeted codebase alterations in .github/workflows/autofix-ci.yml, .github/workflows/emulator-phased-validation.yml, src/circuit-validation/engine.js and other core modules. The diff analysis recorded: 4 files changed, 22 insertions(+), 10 deletions(-). This commit originally addressed: fix workflow.*
- *Fix: cli repo name.*
- *Fix:sketch nae in workflow.*
- *Fix landing page redirect.*
- *Move to vite 6.2.0.*
- *Executed targeted codebase alterations in .github/workflows/pr-checks.yml, src/pages/mobileui/SimulatorPage.jsx. The diff analysis recorded: 2 files changed, 17 insertions(+), 7 deletions(-). This commit originally addressed: ooooo.*
- *Fix: rollup failed to resolve import "@openhw/emulator".*
- *Executed targeted codebase alterations in .github/workflows/page-validation.yml, .github/workflows/pr-checks.yml. The diff analysis recorded: 2 files changed, 4 insertions(+), 4 deletions(-). This commit originally addressed: last.*
- *Add: rollup native package.*
- *Do not even know.*
- *Add: missing lightingcss package.*
- *Fix: nestead directaries.*
- *Add: install of @rollup/rollup-linux-x64-gnu@4.60.2.*
- *Solve problem regrading nested cd.*
- *Executed targeted codebase alterations in .github/workflows/pr-checks.yml. The diff analysis recorded: 1 file changed, 1 insertion(+), 1 deletion(-). This commit originally addressed: check workflow.*
- *Executed targeted codebase alterations in .github/workflows/pr-checks.yml. The diff analysis recorded: 1 file changed, 8 insertions(+), 11 deletions(-). This commit originally addressed: fix: workflow.*
- *Fix triple-path issue in workflows by removing job-level working-directory defaults.*
- *Fix:work flow error.*
- *Added: more test to pr add: new page alignment-lab add: scroll in simualtion canvas.*
- *Added write permission to comment.*
- *Added write permission for comment.*

- *Fix: menu not opening add: scoket helper to snap.*
- *Led autowiring,autocode completed.*
- *Add: page validation on pr added initial autocodieng autowiring.*
- *Add: test for pr.*
- *Added test for pr.*
- *Added: more rules add: circuit fixer add: breadboard snap.*
- *Add: rout for autofix.*
- *Added auto fix panel add: breadboard snap add: some test for github action fix: minor ui problem.*

Week 11 (May 10 - 16, 2026)

- *Remove hard coded jwt expiry remove validation api.*
- *Delete .github/workflows/external-watchdog.yml.*
- *Remove old image.*
- *Upadate: deploy.yml.*
- *Upadate: workflow.*
- *Issue: shareable link.*
- *Executed targeted codebase alterations in compiler_Backend.md. The diff analysis recorded: 1 file changed, 29 deletions(-). This commit originally addressed: remove: doc.*
- *Fix: agent and backend deployement.*
- *Change: allways build agent.*
- *Fix: haelt agent issue.*
- *Fix build and test resolution: use vite-node with inlined deps for tests and emulator alias for workers.*
- *Fix build and test resolution: use vite-node for tests and emulator alias for workers.*
- *Fix build resolution: allow wasm files and use emulator alias in workers.*
- *Add: valiadtion api.*
- *Fix: gitignore was ignring wasm files.*
- *Before palette refactor.*
- *Add: new 2 pin ldr sensor add: ntc temp sensor.*
- *Add: bg color to sub menu refactor: autofix.*
- *Add: autowire for more components.*
- *Fix: runner can not create issue.*
- *Add: support for more components in autofix add: owner tag in json remove: autosetup.js move all code to projectutils.js.*

- *Remove: context menu.*
- *Add: componentcontext panel.*
- *Remove: auto fix validation.*
- *Remove: png download wasm.*
- *Add: telegram notification add: health agent docker.*
- *Add: telegarm notification.*
- *Add: telegram notification.*

Week 13 (May 24 - 30, 2026)

- *Remove code for pin propagation.*
- *Fix: spi related issue.*
- *Executed targeted codebase alterations in scratch_test.cjs, src/controllers/compileController.js, src/esp32/controller/compileController.js and other core modules. The diff analysis recorded: 14 files changed, 2314 insertions(+), 1 deletion(-). This commit originally addressed: add stm32.*
- *Fix: docker cache support.*
- *Executed targeted codebase alterations in Dockerfile. The diff analysis recorded: 1 file changed, 2 insertions(+), 1 deletion(-). This commit originally addressed: fix: esp url.*
- *Fix: esp32 library issue and spi libaray issue.*
- *Fix: esptool issue.*
- *Fix ci issue sync frontend and backend timer added flush for spi.*
- *Fix some spi leak to frontend.*
- *Fix esp added support for i2c,spi,gpio,adc added support for esp cam.*
- *Added compiled cache added notfier when esp is ready.*
- *Added log to see the pin state changes and fix some bug.*
- *Upadte workflow and docker for esp.*

Week 14 (May 31 - June 6, 2026)

- *Remove pcap downlod from esp backend engine.*
- *Trigger rebuild for esp and stm.*
- *Fix:qemu pacp dump and.*
- *Fix nginx hijack.*
- *Fix login redirect.*
- *Feat: dynamic port forwarding, ip scaling, and public/private gateway separation.*
- *Change url for gateway.*
- *Fix github release 403 error by adding write permissions.*

- *Update release workflow to support gateway tags.*
- *Fix release-gateway workflow.*
- *Trigger rebuild.*
- *Fix rendo version.*
- *Fix frontend esp gateway add public gateway docker add workflow to genarate apps for gateway.*
- *Fix spi cs flush.*
- *Fix qemu staring simaution when frontend is selected.*
- *Chnage renode versio to use 1.16.1 added lot of protocal to qemurunner and helper file.*
- *Added vm recousce monitor in admin panel and move caliabartion and other resosuce monitor to healath agent.*
- *Fix modejs is killing itself.*
- *Fix: force rp2040 compiler toolchain download during docker build.*
- *Fix: execasync typo, renode isread case, libsdl2/net-tools, nginx routing.*
- *Protected more routes.*
- *Fix session_secret bug and force worker rebuild.*
- *Added storage section in health agent fix issue in admin dashboard.*
- *Remove ccache added new api for ping add more api fro admin dashboard.*
- *Stm borad url fix.*
- *Chnage in deploy workflow.*
- *Added max3020x library.*
- *Added log to see compile fail for esp.*
- *Add point system for compile and simualtion service to prevent oom.*
- *Added credit based system to prevent oom.*
- *Thanks velxio for helping regarding esp.*
- *Crete seperate worker for esp and stm added libery pool with 500mb cached added compiled binary pool 1gb added hot reload to ep and stm added ccache for faster compilation added support to compile with esp idf.*
- *Revert "fix: obscure revealing backend/db/user details in console logs".*

Week 15 (June 7 - 13, 2026)

- *Executed targeted codebase alterations in src/config/libraries.json. The diff analysis recorded: 1 file changed, 3 insertions(+), 5 deletions(-). This commit originally addressed: Revert "Km".*
- *Exculde helper file when engine is frontend.*

- *Executed targeted codebase alterations in src/esp32/controller/compileController.js. The diff analysis recorded: 1 file changed, 20 insertions(+), 16 deletions(-). This commit originally addressed: fix #define.*
- *Fix cache size issue.*
- *Refactor arduino_cli_path and update execfile calls.*
- *Delete src/utils/arduinocli.js.*
- *Update livesimulationservice.js.*
- *Update livesimulationsession.js.*
- *Remove ip address from session creation.*
- *Revert "feat: initialize libraries configuration with permanent and cached li…".*
- *Component cache will see for all the file chnage in component folder.*
- *Increase webscoket data limit.*
- *Add catch block to catch stm error.*

Week 16 (June 14 - 20, 2026)

- *Ad bridge mode in private gateway.*
- *Executed targeted codebase alterations in last_avrdude_log.txt, package-lock.json, package.json and other core modules. The diff analysis recorded: 4 files changed, 21 insertions(+), 437 deletions(-). This commit originally addressed: Revert "Km".*

Week 17 (June 21 - 27, 2026)

- *Added esp32c3 updated docker config.*

Week 18 (June 28 - July 4, 2026)

- *Min sacn patial.*
- *Executed targeted codebase alterations in src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 1 file changed, 22 insertions(+), 7 deletions(-). This commit originally addressed: nim work.*
- *Executed targeted codebase alterations in src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 1 file changed, 8 insertions(+), 3 deletions(-). This commit originally addressed: ble advert.*
- *Executed targeted codebase alterations in .../controller/compileController.js, src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 2 files changed, 126 insertions(+), 58 deletions(-). This commit originally addressed: added psram.*
- *Executed targeted codebase alterations in .../controller/compileController.js, src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 2 files changed, 55 insertions(+), 69 deletions(-). This commit originally addressed: ble 4.*

- *Executed targeted codebase alterations in .../controller/compileController.js, src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 2 files changed, 69 insertions(+), 6 deletions(-). This commit originally addressed: ble adv.*
- *Executed targeted codebase alterations in src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 1 file changed, 36 insertions(+). This commit originally addressed: ble 6.*
- *Executed targeted codebase alterations in src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 1 file changed, 27 insertions(+), 7 deletions(-). This commit originally addressed: ble 3.*
- *Executed targeted codebase alterations in .../controller/compileController.js, src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 2 files changed, 54 insertions(+), 21 deletions(-). This commit originally addressed: ble 2.*
- *Executed targeted codebase alterations in .../controller/compileController.js, src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 2 files changed, 342 insertions(+), 49 deletions(-). This commit originally addressed: ble started.*
- *Added support for rv32.*
- *Change port to 5001 for compiling.*

Week 19 (July 5 - 11, 2026)

- *Mock gatt and bypass nim.*
- *Executed targeted codebase alterations in .../controller/compileController.js, src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 2 files changed, 54 insertions(+), 24 deletions(-). This commit originally addressed: e.*
- *Executed targeted codebase alterations in src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 1 file changed, 39 insertions(+), 5 deletions(-). This commit originally addressed: w.*
- *Executed targeted codebase alterations in src/compiler(esp32)/utils/RV32BridgeHelper.h. The diff analysis recorded: 1 file changed, 50 insertions(+), 22 deletions(-). This commit originally addressed: m.*

Chapter 7: Conclusion & Future Scope

7.1 Summary of Deliverables

My summer internship with FOSSEE at IIT Bombay resulted in key feature expansions and architectural enhancements across all layers of the OpenHW Studio stack. By focusing on emulation accuracy, multi-threaded performance, security hardening, and compiler compilation scaling, I succeeded in making the platform a robust, classroom-ready virtual laboratory tool.

My core deliverables include:

- **Development of Nine Virtual Peripherals:** Built the simulation logic, SVG visual interfaces, and custom validation checks for ILI9341 TFT, ESP-CAM,OLED Display, Cahracter Display, IR Receiver/Remote.
- **Simulation Pipeline Overhaul:** Integrated the Monaco Editor, engineered background Web Worker execution threads (`execute.ts`, `avr-runner.ts`, `rp2040-runner.ts`), and optimized graphic rendering for display workers using `SharedArrayBuffers` to hit 30fps.
- **WASM ESP32 Networking Engine (`rv32-runner.ts`):** Implemented cycle-budgeted WebAssembly loops, added socket routing gateways for BLE, Wi-Fi, and Thread, built Ethernet packet padding (to prevent LwIP frame drops), and patched the ARP request `00:00:00:00:00:00` MAC address bug.
- **Backend Compiler Caching Pools:** Implemented SHA-1 hashing, a 500MB library memory cache, and a 1GB compiled binary cache, reducing server compilation times from 8s to under 200ms for cached code. Added a serialized queue manager and point-based throttling to prevent OOM server failures under heavy classroom loads.
- **DevOps & Monitoring Automation:** Containerized the backend compiler with pre-cached toolchains in Docker, orchestrated GitHub Actions workflows (`deploy.yml`), and built the Telegram `health-agent` dashboard alert system.
- **Automated Rust WASM Grading Engine:** Architected the client-side evaluation engine by compiling Rust grading logic into WebAssembly modules. This system executes high-speed boolean matrix evaluations against student netlists in background threads, seamlessly syncing assignment scores to the teacher dashboard.
- **Auto-Wire, Auto-Fix & Validation Physics:** Engineered automated Breadth-First Search (BFS) routing for Auto-Wiring, programmed rigorous short-circuit detection algorithms (e.g., lethal VCC-to-GND shorts), and built the highly complex "Auto-Fix" GUI, which mathematically calculates missing netlist routes and renders dynamic SVG splines to visually guide student corrections.
- **Full-Stack Simulation Pipeline Architecture:** Designed an aggressive dual-layer caching strategy (IndexedDB client cache and RAM/Disk server pools) that completely bypasses compilation latency.

Orchestrated an advanced Web Worker architecture utilizing zero-copy MessageChannels and multi-architecture runners (avr-runner.ts, rp2040-runner.ts, esp32-runner.ts and cloud proxies) to isolate CPU execution ticks, manage complex memory state interception, and run secure, background telemetry and grading engines without blocking the React UI thread.

7.2 Future Research and Development Directions

While the current version is highly stable and performant, several promising features were initiated during the final weeks of June that provide a clear roadmap for future development:

7.2.1 RISC-V (RV32) Architecture support

The initial backend compilation profiles and emulator address map hooks for the `rv32` (RISC-V 32-bit) instruction set have been committed. Future work involves completing the WASM instruction execution loops to enable full in-browser simulation of open-source RISC-V microcontrollers.

7.2.2 Bluetooth Low Energy (BLE) Emulation

The initial communication channels and WebSockets configurations for BLE are established (`ble started`, `ble 2`, `ble 3`). The next phase requires completing the virtual Host Controller Interface (HCI) layer. This will allow the simulator to establish wireless BLE connections, letting students connect their mobile phones directly to the running web simulation.

7.2.3 ML-Assisted Circuit Debugging

Using the telemetry API developed to stream component values to the HUD, future developers could feed real-time circuit state variables and compilation diagnostic lines into a Large Language Model (LLM). This AI agent could provide context-aware, conversational hints to assist students when their circuits or codes fail assessments, mimicking the guidance of a human lab assistant.

7.2.4 Desktop Application via Electron

Currently, the platform is restricted by browser sandbox limitations (such as memory limits and strict CORS policies). A critical future goal is to package the entire React frontend and the WASM emulator engines into an Electron-based desktop application. This will allow OpenHW Studio to run as an offline-first, native executable on Windows, macOS, and Linux, giving the emulator direct, unrestricted access to the host machine's file system and raw USB ports for even faster hardware flashing.

References

1. React Documentation, Meta Platforms, Inc. Available: https://react.dev/
2. Vite Build Tool, Evan You. Available: https://vitejs.dev/
3. AVR8js: AVR Emulator in JavaScript, Uri Shaked. Available: https://github.com/wokwi/avr8js
4. Arduino CLI Documentation, Arduino. Available: https://arduino.github.io/arduino-cli/
5. ESP-IDF Programming Guide, Espressif Systems. Available: https://docs.espressif.com/projects/esp-idf/
6. Node.js API Documentation, OpenJS Foundation. Available: https://nodejs.org/api/
7. WebAssembly Specification, W3C. Available: https://webassembly.github.io/spec/
8. IndexedDB API, Mozilla Developer Network. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
9. SharedArrayBuffer Documentation, MDN Web Docs. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer
10. Rust Programming Language, Rust Foundation. Available: https://www.rust-lang.org/
11. Arduino-IRremote Library Documentation, Armin Joachimsmeyer. Available: https://github.com/Arduino-IRremote/Arduino-IRremote
12. DallasTemperature Arduino Library Documentation, Miles Burton. Available: https://github.com/milesburton/Arduino-Temperature-Control-Library
13. OpenHW Studio GitHub Organization. Available: https://github.com/OpenHW-Studio
14. Renode Framework, Antmicro. Available: https://renode.io/
15. Monaco Editor, Microsoft. Available: https://microsoft.github.io/monaco-editor/
16. rp2040js: Raspberry Pi Pico Emulator, Uri Shaked. Available: https://github.com/wokwi/rp2040js
17. MicroPython, George Robotics Ltd. Available: https://micropython.org/