Summer Internship Report

on

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

Submitted by

Satvik Sharma

B.Tech in Computer Science Engineering(AIML)

VIT Bhopal University

Under the Guidance of

Prof. Prabhu Ramachandran

Principal Investigator, FOSSEE Project

Department of Aerospace Engineering

Indian Institute of Technology Bombay

June 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.

Satvik Sharma
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.

Satvik Sharma
VIT Bhopal University

Index

Chapter 1
Introduction

OpenHW Studio is a browser-based, open-source educational platform developed under the FOSSEE initiative at IIT Bombay, built to solve a problem that has long limited hands-on hardware education: access. Physical microcontroller kits, sensors, motor drivers, and displays are expensive, easy to damage, and rarely available in the quantities a classroom actually needs, which puts hands-on hardware learning out of reach for many students. OpenHW Studio addresses this by giving students, teachers, and engineers a fully interactive environment to design, simulate, and test hardware circuits, spanning Arduino Uno (AVR), Raspberry Pi Pico (RP2040), ESP32, and STM32 microcontroller cores, entirely inside a web browser, without requiring any physical hardware.

What distinguishes the platform from a simple circuit-diagram tool is that its simulation operates at the instruction level. Rather than approximating what a program should do, the emulator actually executes the compiled machine code produced from a student's Arduino sketch, and routes the resulting electrical signals to and from virtual components on the canvas in real time. A bug in a student's code, whether an inverted digital pin, a missing pinMode() call, or a miscalculated PWM value, produces the same incorrect behavior in the simulator that it would on real hardware, which is what makes the learning transferable rather than superficial.

The platform is built on a three-layer architecture: a React/Vite frontend, which handles the drag-and-drop workspace, component rendering, and all

user-facing interaction; a Node.js backend, which manages authentication, persistent user data, and classroom features; and a TypeScript-based hardware emulator, which is the computational core responsible for simulating both the microcontroller cores themselves and the dozens of individual components that can be wired to them. TypeScript's static typing plays a meaningful role in the emulator specifically, since correctness around pin types, register widths, and signal directions isn't a stylistic nicety here; it directly determines whether a simulated circuit behaves electrically correctly.

OpenHW Studio is fully open-source and organized under the GitHub organization github.com/OpenHW-Studio, spanning five repositories that mirror this architecture: OpenHW-studio-frontend, the React application; openhw-studio-emulator, the TypeScript simulation engine and component library; openhw-studio-backend, the Express/MongoDB API; openhw-studio-docs, developer and user documentation; and openhw-studio-examples, a library of pre-built demo projects. The live deployment is accessible at https://openhw-studio.fossee.in/.

The platform targets STEM learners, engineering students, and open-source contributors, and serves three distinct user roles, Guest, Student, and Teacher, through a shared workspace where components can be drag-dropped, wired, and simulated in real time. A Guest can explore the simulator without creating an account, lowering the barrier to a first look; a Student's activity, progress, and unlocked components persist against their profile; and a Teacher gains access to classroom-oriented tooling for monitoring and managing groups of students. Educational value is reinforced through three complementary systems I contributed to directly: a guided Tour Guide for onboarding new users into the simulator interface, a Gamified Component Progression system called the Arduino Adventure Map, in which advanced sensors and controllers unlock as students earn XP and badges rather than being available all at once, and a growing library of demo projects that give students a working reference point before they build independently.

1.1 Project Overview

As part of my summer internship, I contributed to the development and enhancement of OpenHW Studio across four of its five repositories, frontend, emulator, backend, and documentation, working on features that ranged from low-level simulation logic to full user-facing systems that students and teachers interact with directly.

My primary responsibility at the outset was building out the platform's initial set of UI pages, since much of the frontend's visual identity and page structure had not yet been established when I joined. There was no existing design system to extend, no established component library, and no fixed navigation pattern, which meant the early decisions I made around layout, typography, and page structure weren't just cosmetic choices but ones that every later contributor, myself included, would need to build against. This early UX work became the visual and structural foundation that later features, including the Tour Guide and the gamification interface, were built to match, giving the platform a consistent feel even as its feature set grew substantially over the course of the internship. It also meant that, unusually for a technical internship, some of my earliest work had almost nothing to do with algorithms or backend logic and everything to do with taste and consistency, decisions that wouldn't show up in a diff of clever code but would be felt every time someone navigated between two pages of the platform.

Alongside this, I was responsible for designing and implementing the onboarding Tour Guide on the simulator page, a system meant to orient first-time users inside what is, admittedly, a fairly dense interface once the full component library, wiring canvas, and code editor are all visible at once. Building this required thinking less like a frontend engineer and more like a teacher for a while, since the hardest part wasn't the technical implementation of the spotlight and tooltip overlay itself, but deciding what a genuinely new user needs to see first, second, and last, and how much explanation is too much before it starts to feel patronizing rather than helpful.

I also took on heavy, sustained contributions to the gamification system, spanning the full stack from the Arduino Adventure Map's frontend progression interface, where students see their XP, levels, and unlocked components represented visually, down to the MongoDB schemas and Express endpoints that persist that same data reliably on the backend. Working across both ends of this system meant I had to think about gamification not as a single feature but as a pipeline: a student's action in the simulator needed to update state correctly in the browser, sync to the server without giving the client any opportunity to tamper with it, and then reflect back into the UI in a way that felt immediate and rewarding rather than delayed or inconsistent. This full-stack ownership turned out to matter more than I initially expected, since a gamification system where the frontend and backend were built by different people with slightly different mental models of what XP or an unlock actually meant would have been much harder to keep coherent than one person reasoning about the entire pipeline end to end.

On the emulator side, my work centered on debugging and fixing twelve virtual components spanning sensors, actuators, displays, and input devices, several of which had incorrect pin mappings or faulty simulation logic that would have quietly misled any student relying on them, and some of which were missing outright and needed to be built from scratch across their manifest, logic, and UI layers. This part of the work was less about adding visible new features and more about making sure the simulation itself could be trusted, since a hardware simulator that behaves subtly incorrectly is arguably worse than one that simply doesn't support a component yet; a missing component is an obvious gap a student notices immediately, while a component that's wired correctly but simulated wrong teaches the wrong lesson without anyone realizing it until much later. I supplemented this with smaller-scale contributions to the docs repository, helping keep developer-facing documentation aligned with the features being added, so that the pace of development didn't leave the documentation permanently trailing behind what the codebase actually did.

Across all of this, the underlying goal stayed consistent: significantly expand the hardware simulation catalog, stabilize the execution pipeline so that expansion didn't come at the cost of reliability, and enrich the overall user experience, making OpenHW Studio a more complete and engaging tool for hardware learning, for a student encountering embedded systems for the first time as much as for an experienced contributor extending the platform further.

Chapter 2
Feature Additions

2.1 Hardware Emulator Development (openhw-studio-emulator)

A significant portion of the internship involved writing and fixing the simulation logic, pin validation, manifest definitions, and UI components for virtual electronic hardware. Every component in the emulator follows a consistent three-file architecture, and understanding this pattern is useful context for the fixes described below.

manifest.json defines the component's pins and metadata declaratively, describing how many pins it exposes, whether each is digital, analog, power, or ground, and any sensor-specific parameters like detection range. This is what the frontend's component registry and the microcontroller runners read to understand a component without parsing its simulation code. logic.ts implements the simulation state machine, the actual behavioral model of the component: how its internal state evolves in response to electrical signals it receives, and what signals it outputs in turn. This is where a light-level slider becomes an analog voltage, or an incoming SPI byte stream becomes pixel data on a display. ui.tsx is the React-based visual representation rendered on the workspace canvas, covering the component's appearance, its interactive controls such as sliders, switches, and indicator LEDs, and how it visually reflects state changes coming from logic.ts.

Because these three layers have to stay consistent with each other, a pin declared in the manifest has to be handled correctly in the logic and represented accurately in the UI, a bug in any one of the twelve components I worked on typically meant tracing an issue through all three files to find where the mismatch originated.

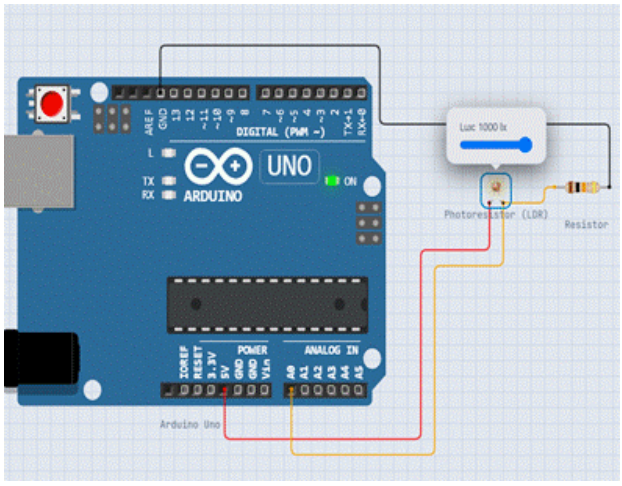
2.1.1 Sensor Integrations & Fixes

Photoresistor (LDR) & Photodiode

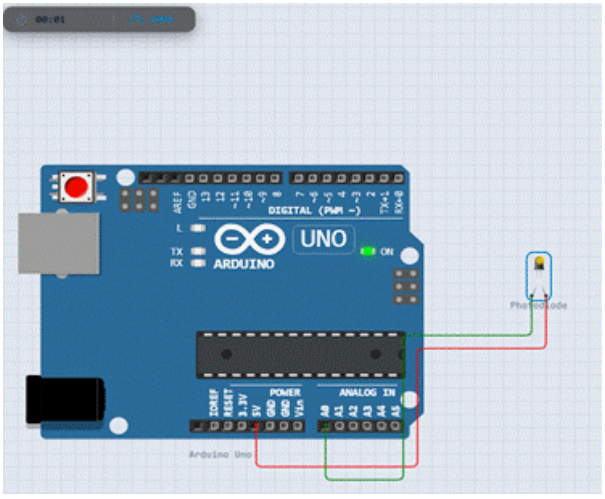
A photoresistor, or LDR, is a light-dependent resistor whose resistance decreases as ambient light increases, while a photodiode converts incident light directly into a current or voltage with a faster response time. Both are common ways of giving a microcontroller a sense of ambient brightness, and both appear frequently in beginner projects like automatic night lights or light-triggered alarms, which is part of why getting their simulation right mattered. Since there's no physical light source in a browser-based simulator, the component instead exposes a light-level slider as a stand-in for ambient brightness, and the core task was making that abstraction behave electrically like the real sensor would, so that code written against the simulated version would transfer correctly to real hardware later.

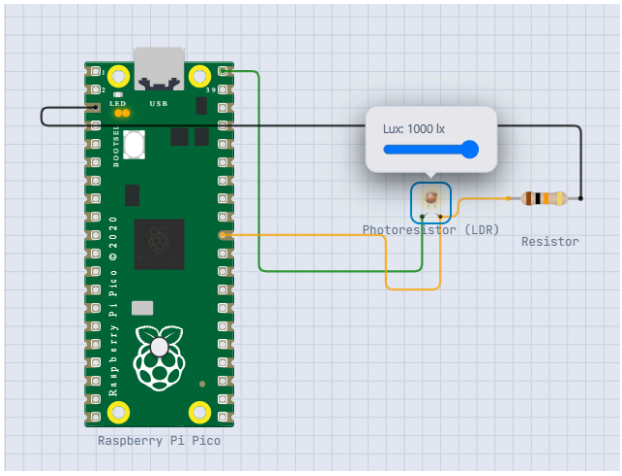
- Implemented and refined the analog voltage mapping logic in logic.ts, translating light-level slider values to accurate analog output voltages
- Updated ui.tsx to render a responsive light-level control panel for real-time user interaction
- Registered the component in index.ts to ensure correct export and integration into the frontend registry

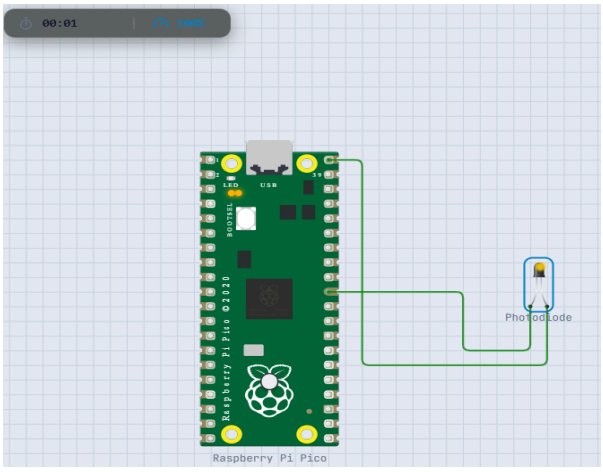
PHOTORESISTOR(LDR)

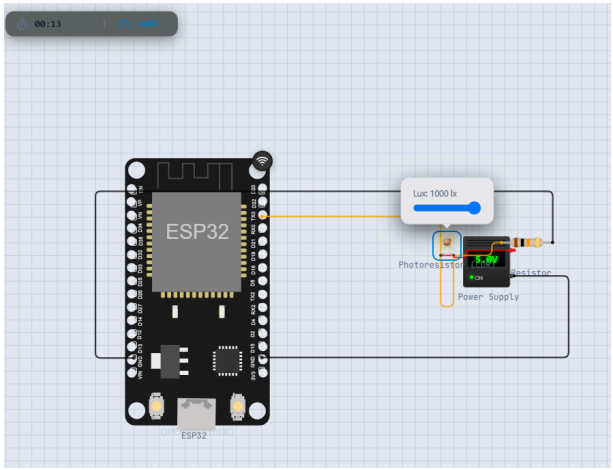


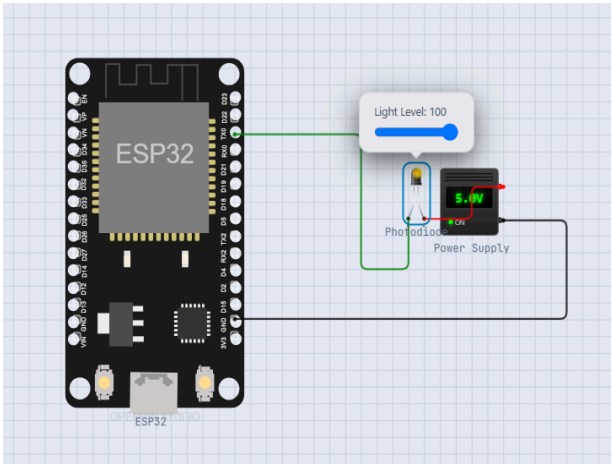
PHOTODIODE







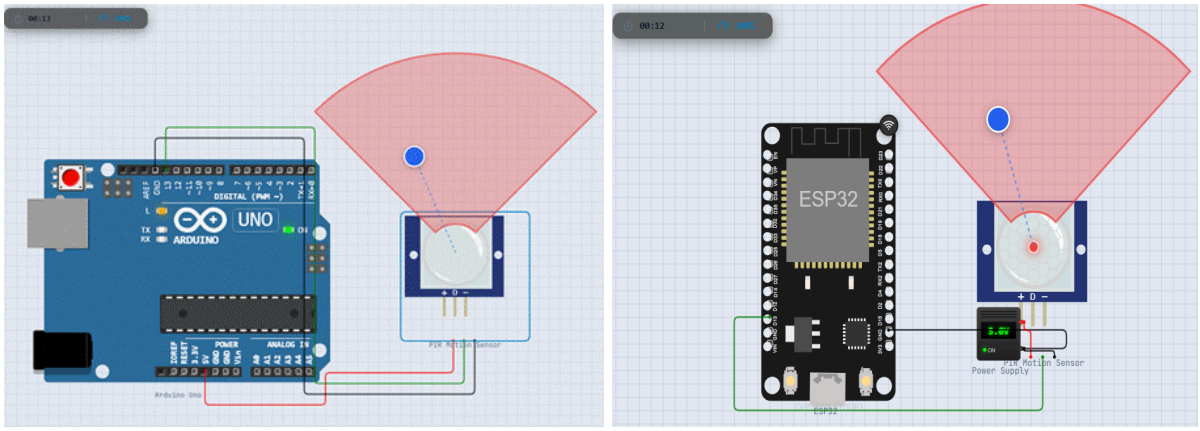


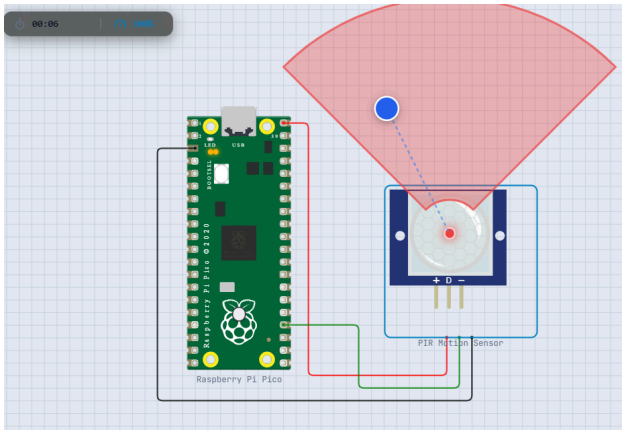


PIR Motion Sensor

A PIR, or passive infrared, motion sensor detects motion by sensing changes in infrared radiation, essentially body heat, moving across its field of view, and typically asserts a digital HIGH output when it detects movement. These sensors are common in security and automation projects, where a microcontroller needs a simple binary signal indicating whether something moved within range, without needing any more detail than that.

- Diagnosed and fixed incorrect manifest pin mappings that were preventing motion interrupt signals from reaching the microcontroller runner.
- Corrected the trigger logic to properly assert digital HIGH on motion detection and reset after a configurable timeout.

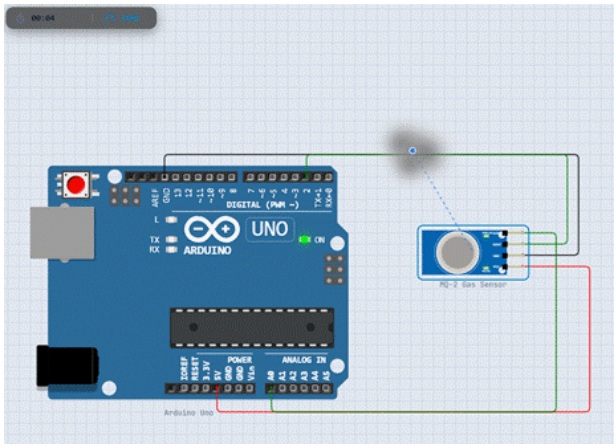


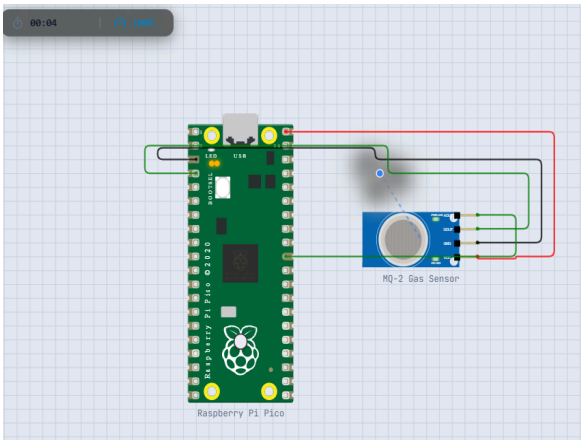


MQ2 Gas Sensor

The MQ2 is a widely used gas sensor that detects combustible and flammable gases, including LPG, propane, smoke, and methane, and typically exposes both an analog output proportional to gas concentration and a digital output that flips once concentration crosses a set threshold, the combination commonly used in simple gas-alarm circuits. This component didn't exist in the emulator at all, so I developed complete sensor support from scratch, which meant defining its behavior end to end rather than fixing an existing implementation.

- Developed complete sensor support from scratch, including:
  - manifest.json: defined analog and digital output pins, sensor metadata.
  - logic.ts: implemented gas concentration simulation with configurable threshold triggering.
  - ui.tsx: built a real-time gas level slider and threshold indicator panel.

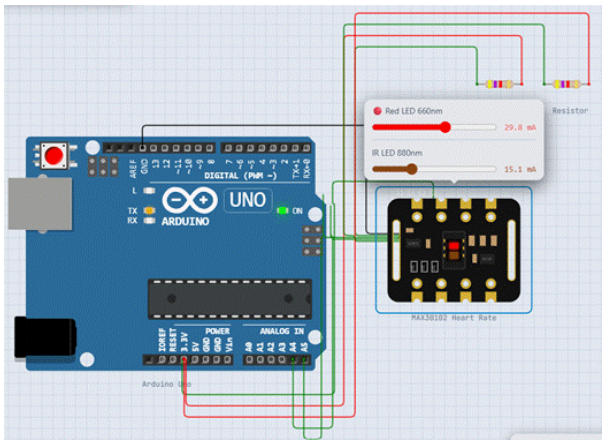


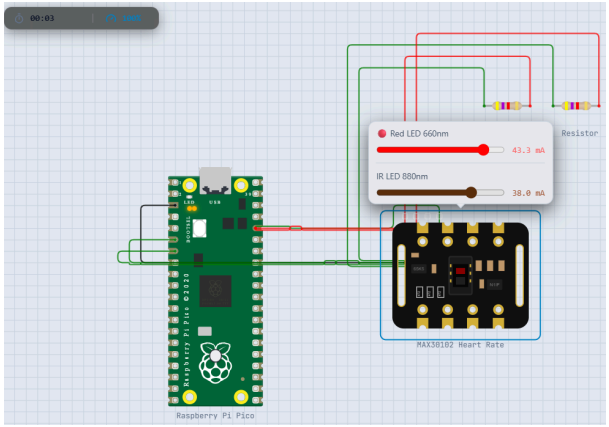


MAX30102

The MAX30102 is an integrated pulse oximeter and heart-rate sensor that communicates over I2C, commonly used in wearable and health-monitoring projects to measure heart rate in BPM and blood oxygen saturation, or SpO2. Because it communicates over I2C rather than exposing a simple analog value, its simulation has to correctly model that protocol layer in addition to producing believable sensor readings.

- Tweaked I2C data telemetry fields and adjusted validation behavior to align with realistic heart rate and SpO2 data ranges.
- Fixed edge cases in the simulation state that caused NaN outputs at boundary conditions.



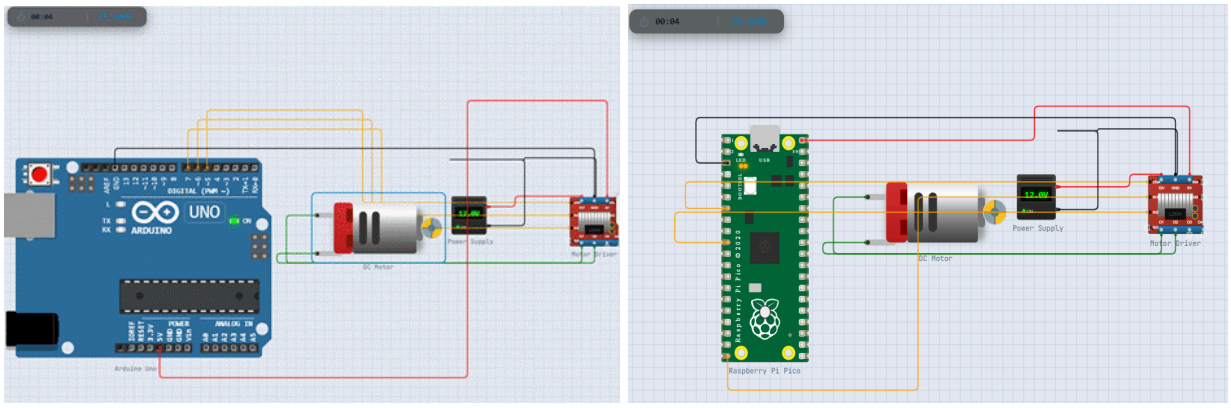


2.1.2 Actuators & Motors

DC Motors & L293D Motor Driver IC

Controlling a DC motor means controlling two independent things: speed, typically via PWM duty cycle, the proportion of time a signal spends HIGH within each cycle, which determines the average voltage and therefore power delivered to the motor, and direction, via an H-bridge, an arrangement of four switches that can reverse the polarity of current through the motor to spin it the other way. The L293D packages two independent H-bridges into a single chip, letting a microcontroller drive two DC motors, or one stepper motor, from a handful of logic pins.

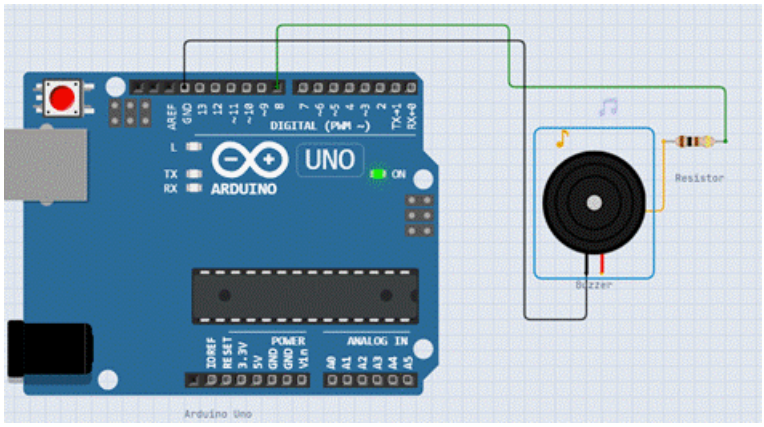
- Developed comprehensive simulation logic covering speed (PWM duty cycle) and direction (H-bridge state) for DC motors.
- Implemented a validation model for the L293D IC that checks enable, input, and output pin states against expected electrical behavior.
- Ensured correct multi-motor support for both single and dual H-bridge configurations.



Buzzer

A piezo buzzer produces an audible tone when driven with a PWM signal, where the frequency of that square wave determines pitch and its duty cycle affects perceived timbre and volume, commonly generated in Arduino code via the tone() function.

- Adjusted PWM protocol handler to correctly interpret frequency and duty cycle parameters from the microcontroller runner.
- Added validation layers to catch misconfigured pin modes and missing PWM signals.
- Updated the UI to display the simulated tone frequency in real time.



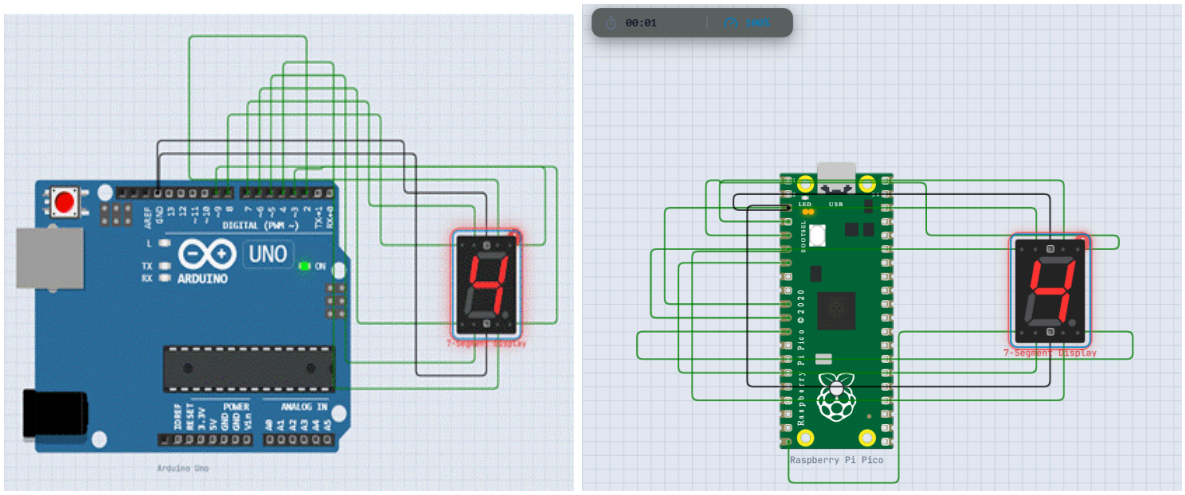
2.1.3 Displays & Indicators

7-Segment Display

A 7-segment display shows digits by lighting some combination of seven individual LED segments, driven either directly, with each segment individually wired to a pin, or through a BCD, or binary-coded decimal, decoder, which converts a 4-bit binary input into the correct segment pattern for digits 0 through 9 and, in extended use, hexadecimal A through F.

- Overhauled the BCD-to-segment decoder logic to correctly map all 16 hexadecimal digits (0-9, A-F).

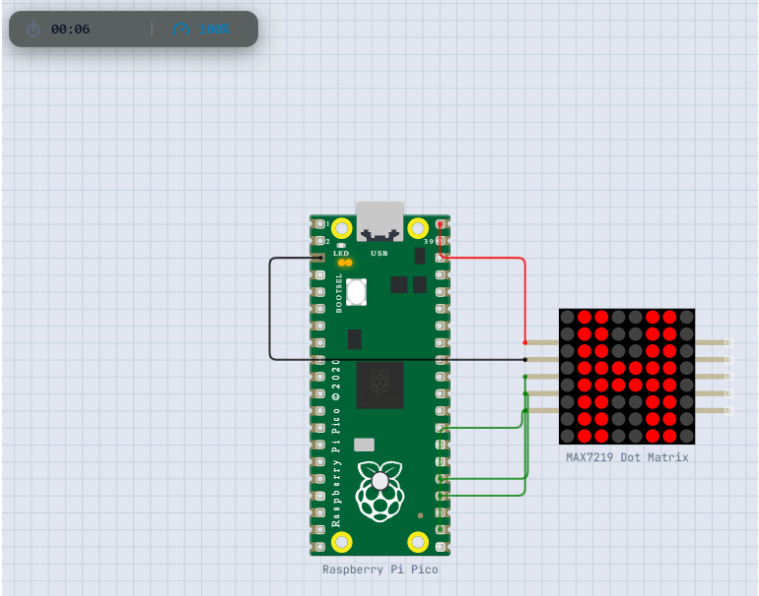
- Fixed direct pin-mapping mode to support individual segment control, resolving a long-standing simulation mismatch.
- Rewrote validation to detect incorrect wiring configurations and surface clear error messages.



MAX7219 LED Dot Matrix

The MAX7219 is an SPI-driven display driver IC that can control up to eight digits of 7-segment displays or a single 8x8 LED dot matrix. It works as a serial-in, parallel-out shift register: data is clocked in over SPI, and only takes visible effect once the LOAD/CS pin is toggled to latch that data to the actual display outputs all at once.

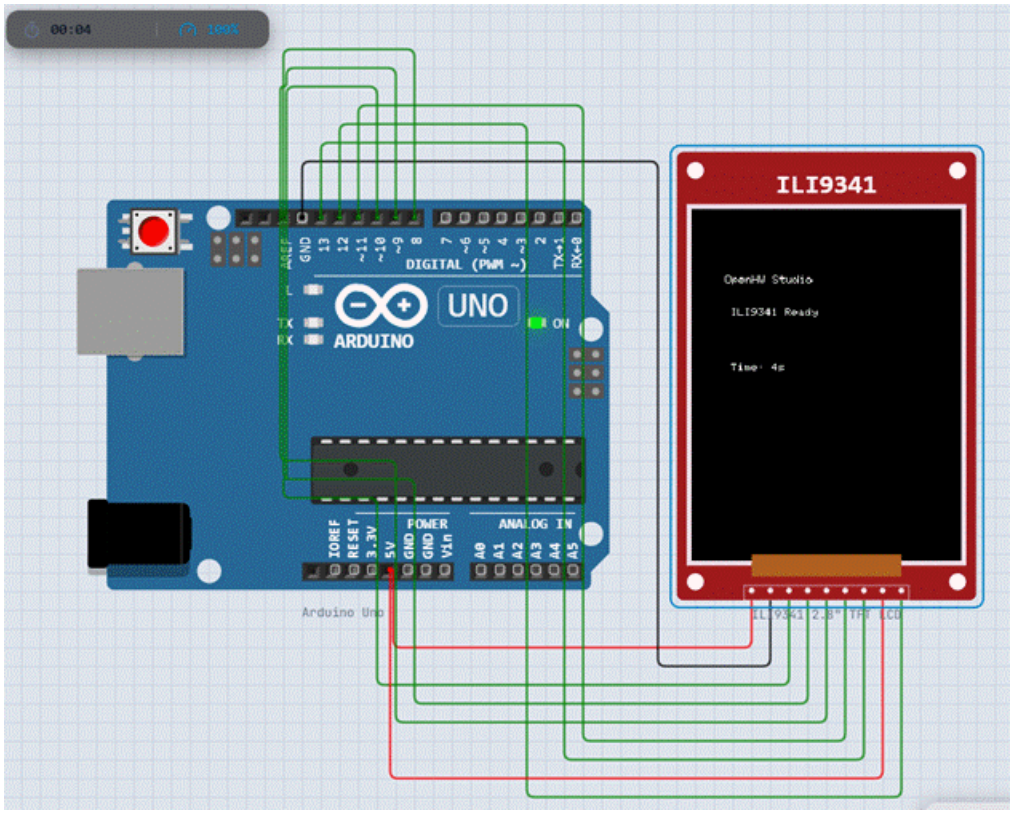
- Resolved simulation logic flaws in the SPI data latching sequence that caused incorrect row/column activation.
- Fixed register address decoding to accurately reflect the MAX7219 datasheet behavior.



ILI9341 TFT-LCD Display

The ILI9341 is a widely used TFT LCD driver, commonly paired with a 240x320 pixel display, controlled over SPI, and appears in many Arduino-compatible display shields.

- Diagnosed and corrected coordinate transformation logic causing off-by-one rendering errors in pixel draw commands.
- Improved SPI frame parsing to handle back-to-back command sequences without state corruption.

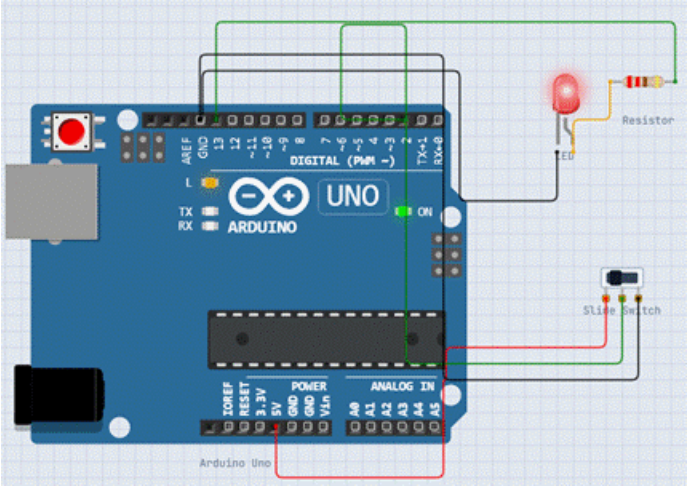


2.1.4 Input Devices

Slide Switch

A slide switch is a simple mechanical switch that provides a clean digital HIGH/LOW logic input, and is one of the most fundamental input components a student encounters.

- Added the slide switch component for both Raspberry Pi Pico (RP2040) and standard Arduino board targets.
- Patched pin manifest definitions to support configurable INPUT and INPUT_PULLUP modes.
- Ensured correct binary state toggling (HIGH/LOW) with proper debounce modeling in logic.ts.



2.2 Frontend & Simulation Pipeline (OpenHW-studio-frontend)

To support the expanded hardware catalog in the emulator, the React/Vite frontend required substantial updates across the component registry, microcontroller runners, telemetry pipelines, and user experience layers, since a component that works correctly in isolation is only useful once the rest of the platform knows how to display it, route signals to it, and let a student interact with it.

2.2.1 Component Registry & Microcontroller Runners

Component Registry Updates

component-registry.ts is effectively the frontend's manifest-of-manifests, a central file that aggregates every emulator component's metadata so the workspace toolbar knows what exists, what to label and icon it as, and how to instantiate it on the canvas when dragged in.

I continuously updated component-registry.ts to register each new hardware definition, including the MQ2 Gas Sensor, Photoresistor, and Slide Switch among others, as they were added, enabling them to appear as draggable components in the workspace toolbar. I also ensured consistent metadata alignment between the emulator's manifests and the frontend's component descriptors, since any mismatch here, such as a pin count or pin type disagreement, would cause the canvas to render something electrically inconsistent with what the emulator engine actually expected.

AVR Runner (avr-runner.ts)

The AVR runner is responsible for taking a student's compiled Arduino sketch and executing it against the simulated hardware, routing electrical signals between the running program and whatever components are wired up on the canvas.

I updated the AVR microcontroller runner to correctly route electrical signals to newly registered components, so new hardware wasn't just visually present but actually reachable by executing code. I resolved critical syntax errors in the compilation pipeline that were blocking program upload entirely for certain board configurations. I also fixed the proxy runner communication handlers responsible for real-time data exchange with the backend, ensuring signal and telemetry data kept flowing smoothly during simulation rather than stalling or desyncing.

RP2040 Runner (rp2040-runner.ts)

The Raspberry Pi Pico's RP2040 chip has a meaningfully different GPIO and peripheral architecture from AVR-based boards, with different pin numbering, ADC channels, and PWM handling, so supporting it well couldn't simply reuse AVR logic.

I extended the Raspberry Pi Pico runner to support the slide switch and additional sensor components, and aligned its pin routing logic with the actual RP2040 GPIO architecture to prevent signal conflicts, making sure two peripherals couldn't end up mapped to overlapping GPIO in a way that wouldn't reflect how the real chip behaves.

Telemetry & Project Utilities

I updated telemetryRegistry.js to correctly parse the runtime data streams coming from the MAX30102 and buzzer components, the layer that feeds real-time value displays and any live charts back to the UI, so it had to understand each new component's specific data format. I also patched projectUtils.js to correctly load and initialize pre-built demo projects, resolving path resolution failures that had been causing bundled example projects to fail to load correctly.

2.2.2 Tour Guide System

I designed and implemented a comprehensive, interactive onboarding tour to help new users navigate the OpenHW Studio simulator interface without feeling lost in front of an unfamiliar, feature-dense workspace. By the time a first-time user reaches the simulator page, they're looking at a component toolbar, a wiring canvas, a code editor, and various simulation controls all at once, and without any guidance, it's easy for someone new to hardware simulation to not know where to start or what any of these panels are actually for. The Tour Guide exists specifically to close that gap, walking a user through the interface's key areas in a structured sequence rather than leaving them to explore blindly.

The **tour** is structured as a sequence of **fourteen steps**, each targeting a specific part of the simulator interface and building on the one before it, so a new user is walked through the workspace in the same order they'd naturally need to understand it: what things exist, how to place them, how to connect them, and how to run and debug the result.

- **Step 1: Welcome**

  A centered introduction with no specific target on the page, setting expectations that the walkthrough will take about a minute and giving a brief overview of what the platform lets a user do before diving into any specific interface element.

- **Step 2: Quick Add Portal**

Introduces double-clicking anywhere on the canvas to instantly search for and add a component, positioned as the fastest way to start building a circuit without first hunting through a sidebar.

- **Step 3: Component Movement**

  Demonstrates that any placed component can simply be dragged to reposition it, with the added detail that components snap to a grid, keeping circuit layouts clean and readable rather than scattered at arbitrary angles.

- **Step 4: Intelligent Wiring**

  Explains how pins are connected by clicking and dragging between them, with the simulator automatically calculating a clean path for the resulting wire rather than leaving the user to route it manually.

- **Step 5: Intelligent Autowiring**

  Introduces a more advanced wiring shortcut, where selecting a component and choosing Wire To from its right-click menu automatically adds any required supporting components, like resistors, and routes the necessary wires on its own.

- **Step 6: Components Palette**

  Shifts focus to the sidebar, introducing the full library of hundreds of available components, spanning boards, sensors, and displays, and reinforcing that any of them can be dragged straight from this palette onto the canvas.

- **Step 7: Save Your Work**

  Points to the File menu and covers the basics of saving a project, opening an existing one, importing files, or creating a local backup copy.

- **Step 8: Integrated IDE**

  Introduces the professional-grade code editor used to write firmware, noting its support for multi-file projects and real-time syntax checking.

- **Step 9: Visual Block Coding**

  Introduces Blockly as an alternative to writing raw code, letting users who prefer a visual approach build their firmware using drag-and-drop logic blocks instead.

- **Step 10: Library Manager**

  Shows users how to open the Libraries panel from the code section to browse and install community-built libraries for sensors, displays, and various communication protocols.

- **Step 11: Serial Monitor**

  Covers interacting with simulated hardware directly by sending commands and viewing debug output in real time as a program runs.

- **Step 12: Real-time Plotter**

A related view accessed from inside the Serial Monitor, used for visualizing high-frequency sensor data through a built-in oscilloscope-style telemetry display.

- **Step 13: System Console**

  Introduces where compilation logs, system warnings, and hardware connection status can all be checked in one place.

- **Step 14: Finish**

  Closes the tour with a centered message confirming the user is ready to start building, pointing them either toward creating their first project from scratch or exploring the examples library to see what's possible on the platform.

Key Components

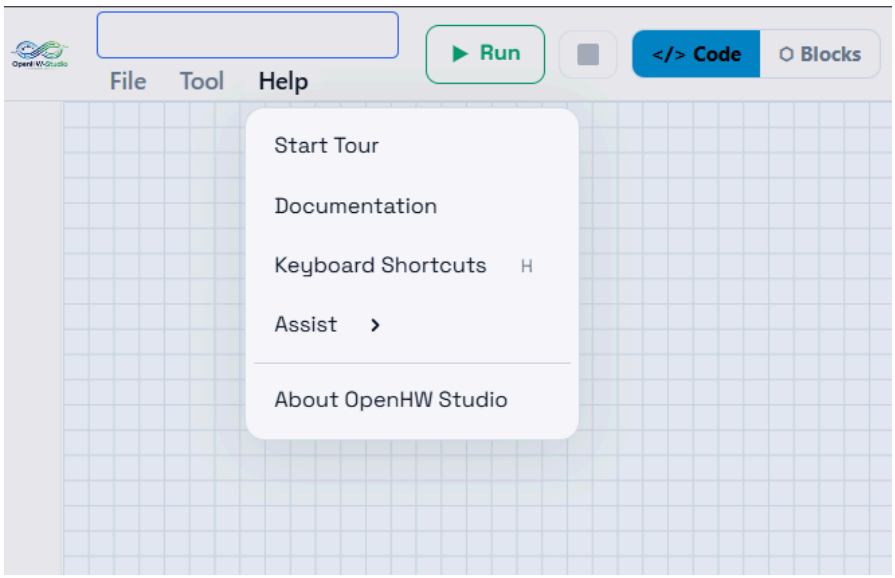
- **TourGuide.jsx** forms the core of this system as a React component implementing a step-by-step guided overlay. Each step highlights a specific part of the interface using an animated spotlight effect, dimming everything else on screen so the user's attention is drawn precisely to the element being explained, whether that's the component drawer, the canvas itself, or the run and simulate controls. The tooltip panels accompanying each step use a glassmorphism style, translucent backgrounds with a backdrop blur, which keeps the underlying interface partially visible rather than fully obscuring it, so the tour feels layered on top of the workspace rather than replacing it

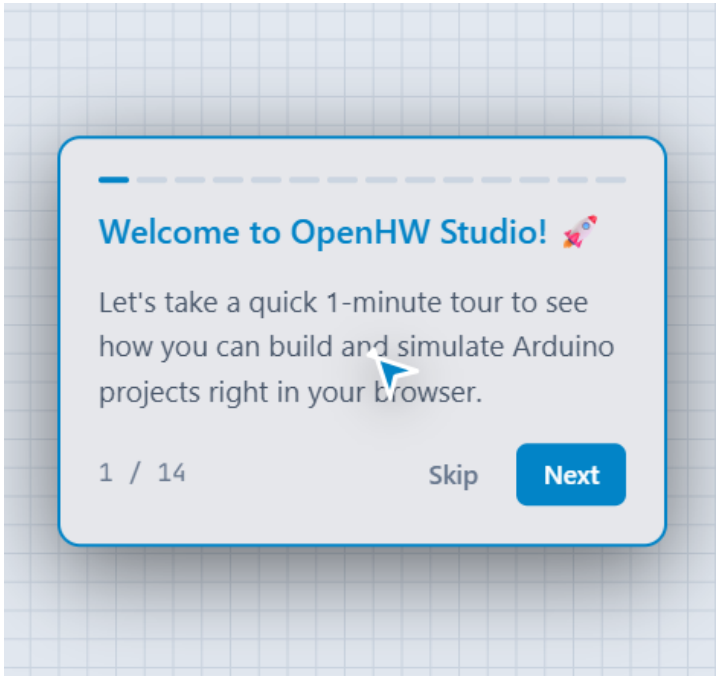
entirely. Transitions between steps use CSS keyframe animations, which was a deliberate choice to make the tour feel like a continuous, guided walkthrough rather than a series of abrupt jumps between disconnected states.

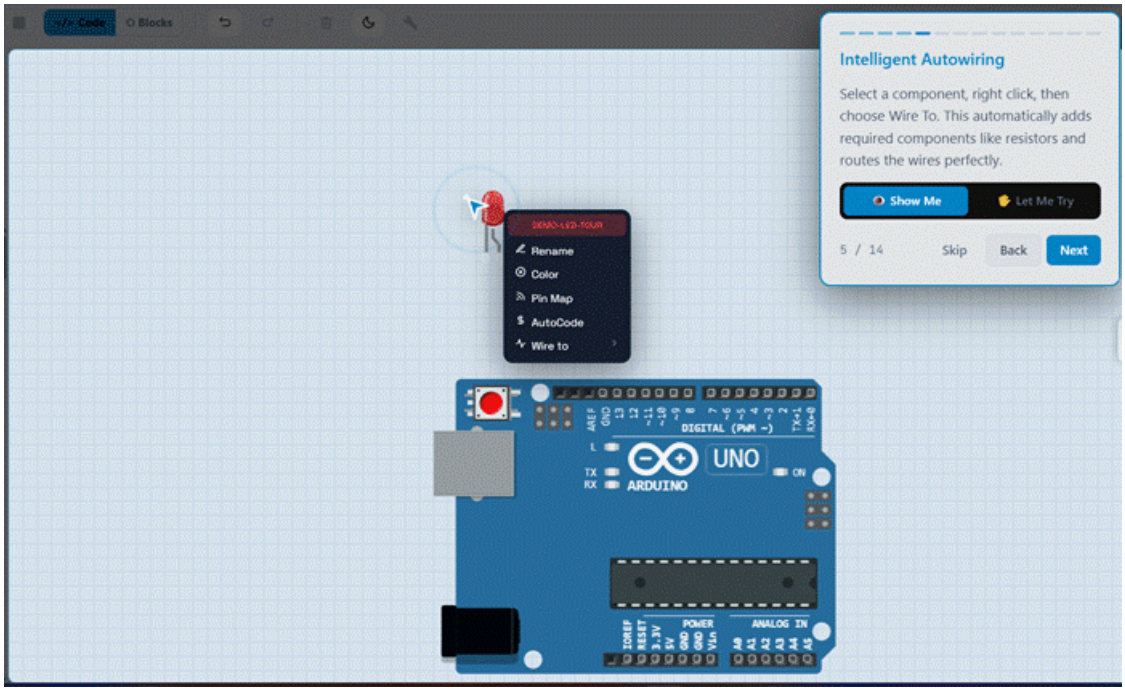
- Accessibility was a real consideration in building this out, not an afterthought. The tour supports keyboard navigation, letting users move between steps with the arrow keys and exit with Escape, for anyone who prefers not to rely on the mouse or is navigating primarily via keyboard. I also added ARIA attributes, including role, aria-label, and aria-live, so that screen readers correctly announce each step's content as the tour progresses, ensuring the onboarding experience is genuinely usable for students relying on assistive technology rather than only for sighted, mouse-driven users.

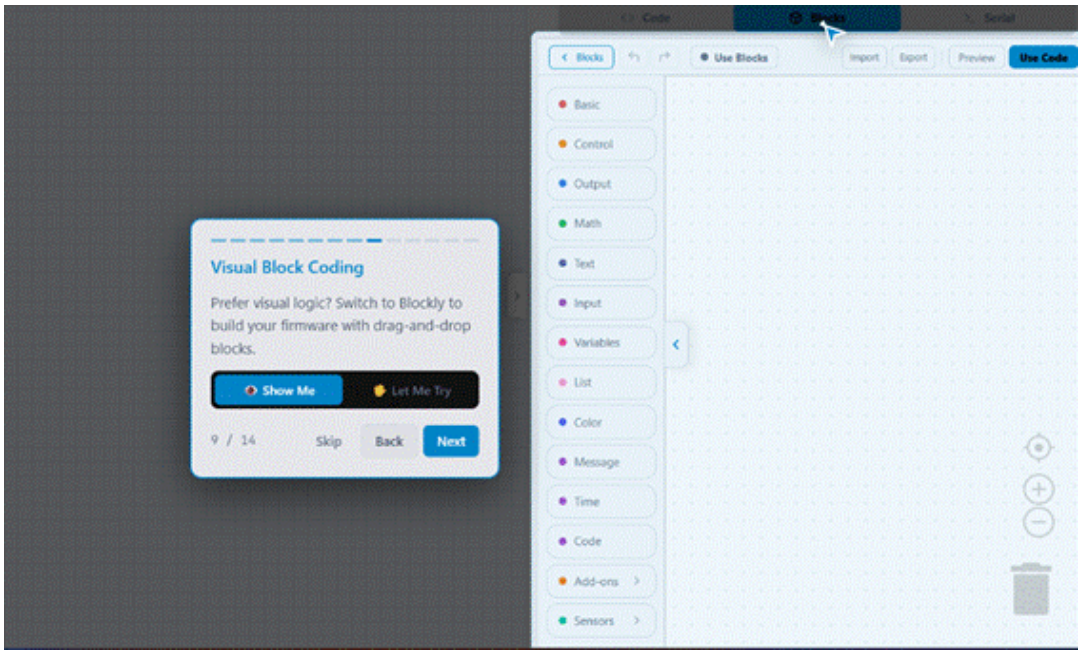
- Behind the visual layer, use**TourLogic.js** is a custom React hook that encapsulates all of the tour's state management separately from how it's rendered, which is a standard pattern in React for keeping logic testable and reusable independent of presentation. This hook handles step progression along with skip and restart controls, so a user can jump ahead or start over without the underlying state getting confused. It also handles DOM element targeting for the spotlight, dynamically recalculating which element to highlight and where to position it, which matters because the underlying layout isn't static; if a panel shifts position or the window resizes mid-tour, the spotlight needs to follow correctly rather than highlighting empty space. Finally, the hook manages persistence of tour completion state across sessions, so once a user has completed the tour, they aren't shown the full walkthrough again automatically the next time they log in.

- The Tour Guide was integrated into SimulatorPage.jsx with a trigger accessible from the toolbar, so any user, new or returning, can re-launch the full onboarding tour at any time. This was an important design decision on its own: rather than treating onboarding as a one-time event a user either sees once or never again, keeping a persistent, easy-to-find trigger means a returning student who forgot how a particular feature works, or a teacher who wants to demonstrate the interface to a class, can pull up the same guided walkthrough on demand, without needing to clear their account data or find some hidden reset option to see it again.




Welcome to OpenHW Studio!
Let's take a quick 1-minute tour to see how you can build and simulate Arduino projects right in your browser.
1 / 14
Skip
Next


Code
Blocks
Intelligent Autowiring
Select a component, right click, then choose Wire To. This automatically adds required components like resistors and routes the wires perfectly.
Show Me
Let Me Try
5 / 14
Skip
Back
Next
Rename
Color
Pin Map
AutoCode
Wire to
UNO
ARDUINO



2.2.3 Contribution in Gamification (Arduino Adventure Map)

To enhance user engagement and provide a structured learning path for hardware simulation, I developed a comprehensive gamification engine for the OpenHW platform. The motivation behind this work came from a fairly common problem in self-directed technical learning: a platform that hands a beginner the full breadth of its tools all at once, dozens of components, an open canvas, and a code editor, tends to produce paralysis rather than progress, since there's no clear starting point and no obvious next step once a student finishes their first small project. Central to addressing this is the Arduino Adventure Map, an interactive, node-based progression interface I designed to gamify the educational journey of learning embedded systems and basic electronics, giving students a concrete path to follow rather than an open-ended sandbox to figure out alone.

The Adventure Map transforms standard curriculum modules into themed Worlds, for example Circuit Basics, Signal Control, and Machines &

Sensors, each grouping concepts that build on one another, so a student moves from fundamentals like LEDs, resistors, and switches toward more involved topics like PWM and analog signal control, and eventually toward motors and complex sensors. The grouping into Worlds isn't arbitrary; it reflects a genuine dependency structure in the underlying material, since understanding PWM meaningfully requires already understanding digital versus analog signals, and working confidently with motor drivers or multi-pin sensor modules requires comfort with basic wiring and code structure first. By organizing content this way, the Adventure Map's structure mirrors how the concepts actually build on each other electrically and logically, not just how they happen to be grouped in a syllabus.

Rather than providing unrestricted access to all simulation tools immediately, which, with dozens of components on offer, could easily overwhelm a beginner, the system employs a linear unlocking mechanism I built into its core logic. Students must sequentially master hardware concepts, completing prerequisites before advancing to more complex components and circuits, similar in spirit to a skill tree in game design, which keeps motivation high through frequent, achievable milestones rather than one distant, intimidating goal. This design choice is doing double duty pedagogically and psychologically. Pedagogically, gating access to advanced components like motor drivers or SPI-based displays behind mastery of simpler prerequisites means a student can't accidentally attempt a circuit that assumes knowledge they don't yet have, avoiding the frustrating experience of failing repeatedly for reasons that have nothing to do with the actual concept being tested. Psychologically, breaking the learning journey into a visible sequence of nodes, each unlockable in a relatively short session, gives students the kind of frequent, tangible sense of progress that a single large "learn embedded systems" goal never could, borrowing directly from what makes skill trees and unlockable content effective in game design more broadly: the reward of completing something is immediate, visible, and clearly connected to what comes next, rather than being deferred until some distant point of full mastery.

System Architecture and Progression Flow

The progression system is driven by a state-managed GamificationContext that tracks user achievements, experience points (XP), and unlocked items.

The learning flow for a single project module consists of five distinct, interconnected nodes:

1. Theory & Reading: Introduces the fundamental physics and logic of the hardware component.
2. Knowledge Check (Quiz): Validates theoretical understanding before practical application.
3. Component Unlock: Visually rewards the user by officially adding the new component (e.g., a Servo Motor or RGB LED) to their personal inventory.
4. Guided Demonstration: Provides a step-by-step walkthrough of the expected circuit.
5. Practical Assessment: The final phase where the user transitions into the Web Simulator to independently build and program the circuit.

Automated Circuit and Code Assessment Engine

The most technically complex segment of the gamification engine is the Project Assessment System. To allow for autonomous learning without human grading intervention, a custom automated evaluation engine was engineered to validate student submissions directly from the browser's sessionStorage.

Upon clicking "Submit Assessment" in the simulator, the engine retrieves a JSON payload containing the user's placed components, wiring array, and C++ source code. The evaluation algorithm then grades the submission across three primary criteria:

A. Component Validation

The engine parses the JSON payload to ensure the correct hardware blocks are present on the canvas. It compares the type and quantity of user-placed components against a strictly defined project configuration object, ensuring no required pieces are missing or bypassed.

B. Topological Wiring Analysis

Rather than simply checking for direct point-A-to-point-B connections, the wiring validation algorithm performs a dynamic topological trace of the electrical circuit.

- Path Tracing: It traces paths through passive components. For example, it validates that an Arduino digital pin connects to a Resistor, verifies the other end of the Resistor connects to the LED Anode, and traces the LED Cathode back to a valid Ground (GND) pin.
- Alternative Configurations: The algorithm accommodates multiple correct approaches, accepting valid alternative connection strategies (e.g., utilizing different available GND pins on the microcontroller) without penalizing the student.

C. Static Code Analysis (C++ Parser)

To evaluate the Arduino code written by the student, a custom static analysis tool was implemented using Regular Expressions (Regex). The parser executes several checks:

- Variable to Pin Mapping: It dynamically identifies user-defined integer variables and maps them to the physical hardware pins.
- Setup Validation: It verifies that necessary configuration functions, such as pinMode(), are declared with the correct I/O direction.
- Behavioral Logic: It checks for specific behavioral implementations, such as ensuring digitalWrite alternates between HIGH and LOW states and that accurate time delays (delay()) are utilized to achieve the expected hardware output.

Scoring Mechanism and Feedback Loop

Each of the three evaluation criteria (Components, Wiring, Code) is assigned a weight and scored out of 100%. The system aggregates these sub-scores into a final Total Score.

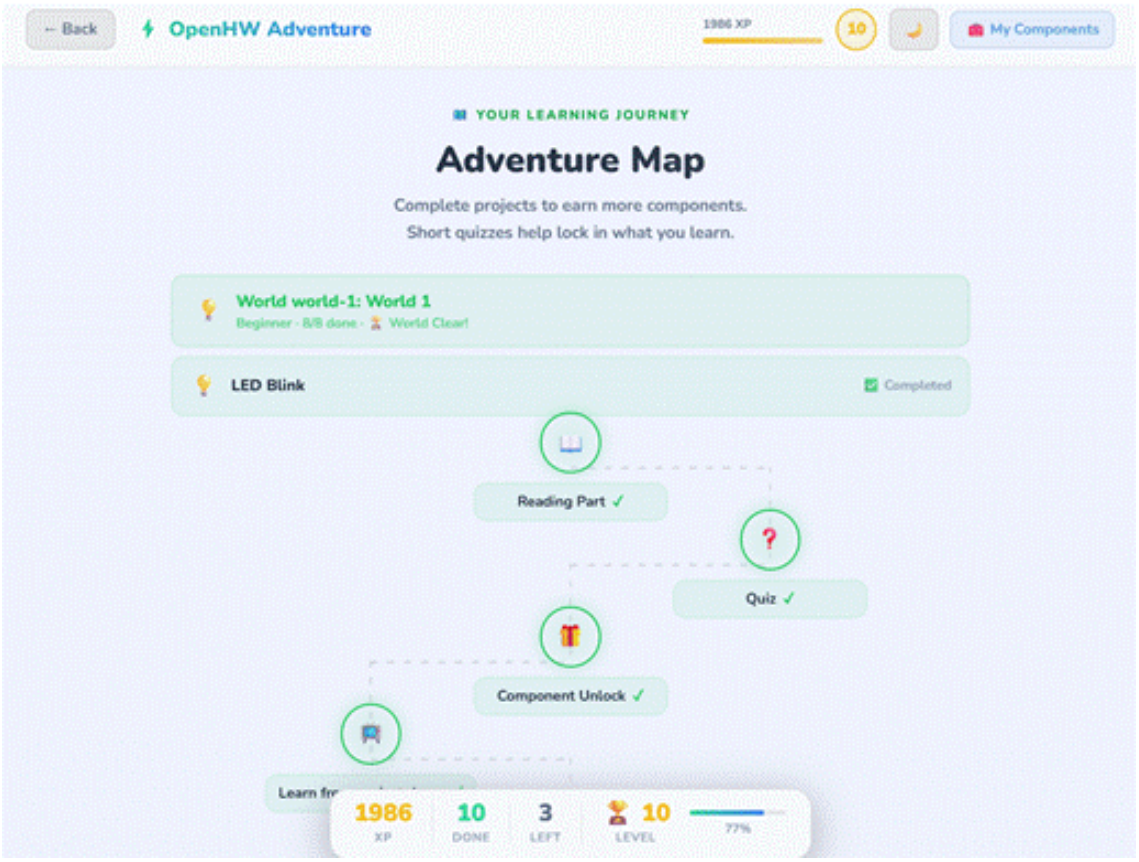
If the Total Score meets the predefined passingThreshold (e.g., 80%), the engine triggers the success state:

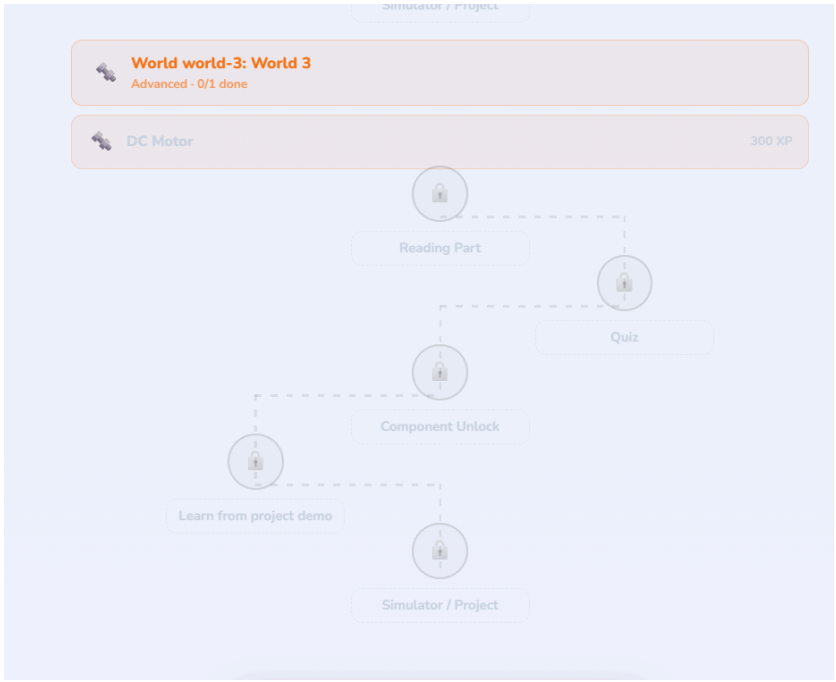
- The project is marked complete in the backend database.
- Experience Points (XP) and virtual Coins are awarded to the user profile.
- The Adventure Map unlocks the next sequential learning node.

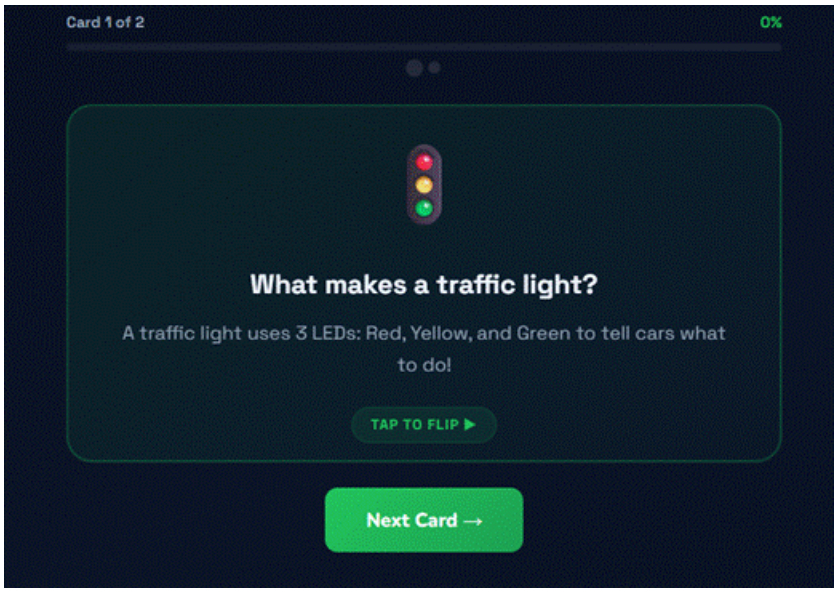
If the submission falls below the threshold, the system provides targeted, actionable feedback. Instead of a generic failure message, the UI displays the exact criteria that failed (e.g., "Missing valid path: Pin 9 → Resistor" or "pinMode should configure the correct output pin"), enabling the student to return to the simulator, debug their work, and resubmit.

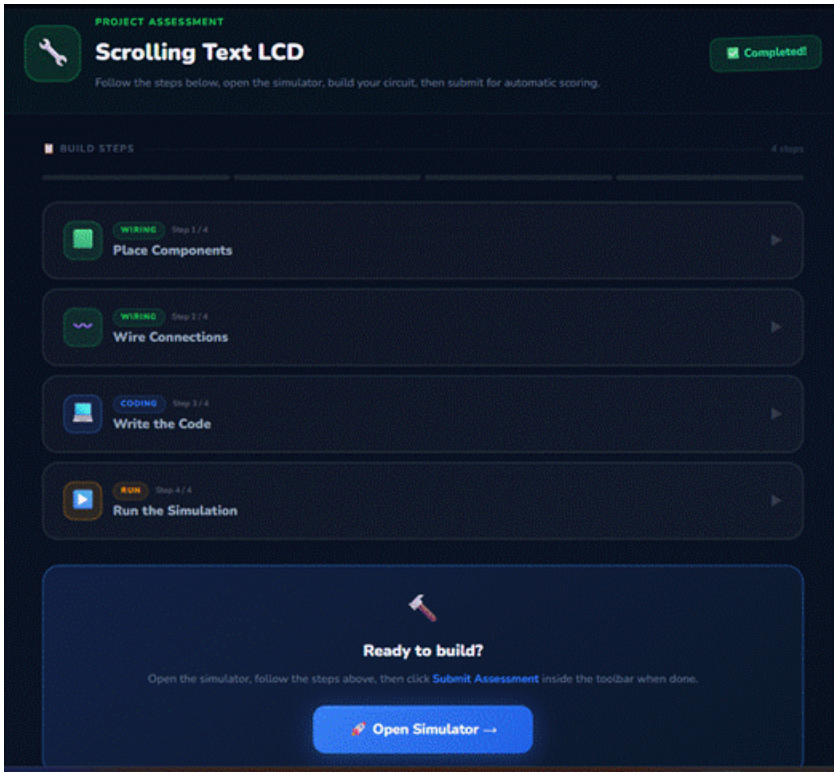
5. Outcomes

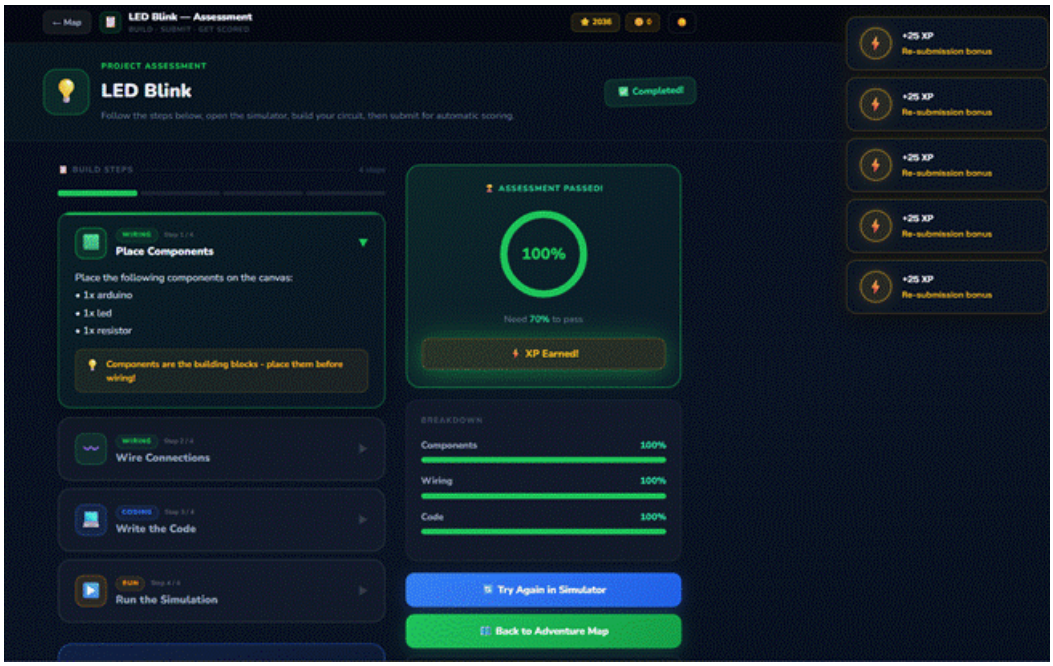
The implementation of the Arduino Adventure Map and the automated assessment engine successfully shifted the platform from a simple sandbox simulator into an intelligent, self-paced educational platform. It significantly reduced the need for manual grading while ensuring students fully comprehend the underlying electronics and software architecture before progressing to advanced topics, building genuine understanding at each stage rather than letting students advance on guesswork.


← Back
OpenHW Adventure
1986 XP
10
My Components
YOUR LEARNING JOURNEY
Adventure Map
Complete projects to earn more components.
Short quizzes help lock in what you learn.
World world-1: World 1
Beginner · 8/8 done · World Clear!
LED Blink
Completed
Reading Part ✓
Quiz ✓
Component Unlock ✓
1986
XP
10
DONE
3
LEFT
10
LEVEL
77%


World world-3: World 3
Advanced · 0/1 done
DC Motor
300 XP
Reading Part
Quiz
Component Unlock
Learn from project demo
Simulator / Project


Card 1 of 2
0%
What makes a traffic light?
A traffic light uses 3 LEDs: Red, Yellow, and Green to tell cars what to do!
TAP TO FLIP ▶
Next Card →


PROJECT ASSESSMENT
Scrolling Text LCD
Completed!
Follow the steps below, open the simulator, build your circuit, then submit for automatic scoring.
BUILD STEPS
4 steps
WIRING
Step 1 / 4
Place Components
WIRING
Step 2 / 4
Wire Connections
CODING
Step 3 / 4
Write the Code
RUN
Step 4 / 4
Run the Simulation
Ready to build?
Open the simulator, follow the steps above, then click Submit Assessment inside the toolbar when done.
Open Simulator →


← Map
LED Blink — Assessment
BUILD · SUBMIT · GET SCORED
2036
0
PROJECT ASSESSMENT
LED Blink
Completed!
Follow the steps below, open the simulator, build your circuit, then submit for automatic scoring.
BUILD STEPS
4 steps
Place Components
Place the following components on the canvas:
• 1x arduino
• 1x led
• 1x resistor
Components are the building blocks - place them before wiring!
Wire Connections
Write the Code
Run the Simulation
ASSESSMENT PASSED!
100%
Need 70% to pass
XP Earned!
BREAKDOWN
Components 100%
Wiring 100%
Code 100%
Try Again in Simulator
Back to Adventure Map
+25 XP
Re-submission bonus
+25 XP
Re-submission bonus
+25 XP
Re-submission bonus
+25 XP
Re-submission bonus
+25 XP
Re-submission bonus

2.2.4 Foundational UI/UX and Web Page Creation

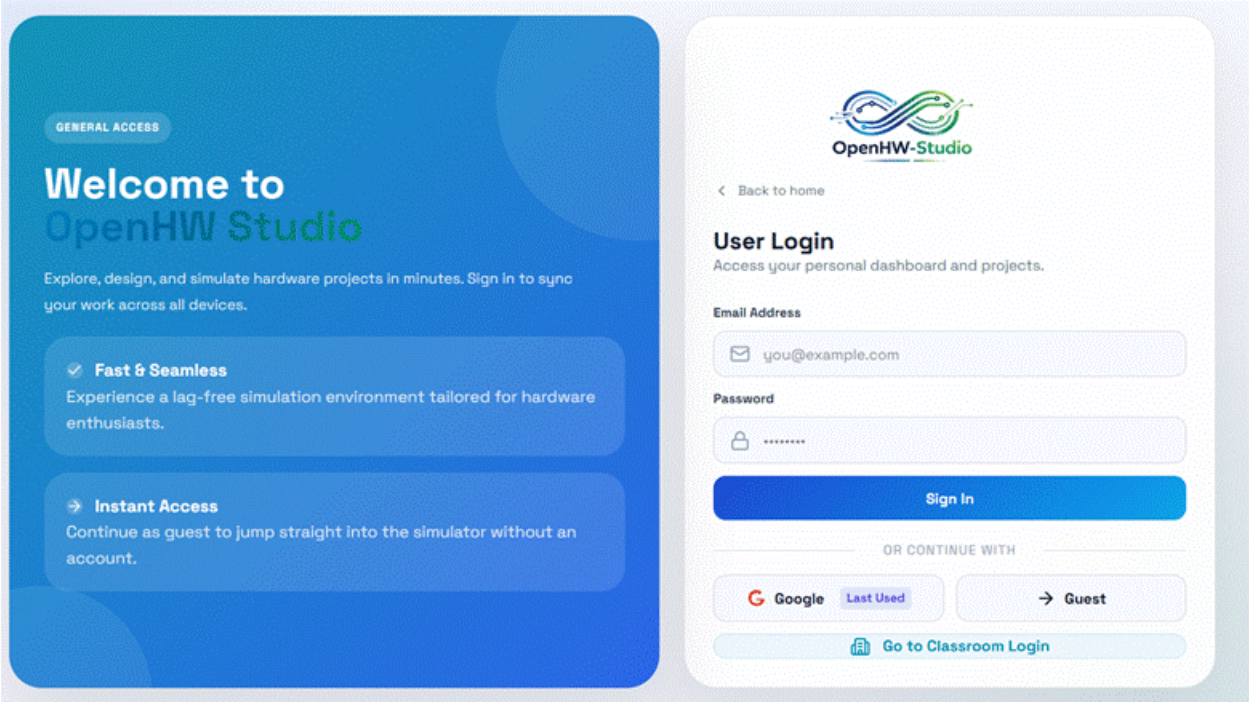
At the inception of my contributions, I took on the core responsibility of spearheading the platform's frontend creation. Since much of OpenHW Studio's visual identity hadn't yet been established, I designed and developed almost all of the primary web pages, effectively establishing the overarching visual theme and User Experience paradigm for the platform, including the typography, color language, and recurring interface patterns that later features would need to stay visually consistent with. At the point I began this work, the frontend repository had a functioning simulator canvas and the beginnings of a component system, but the surrounding pages, the landing page, authentication flows, dashboards, and navigation, existed either as rough placeholders or not at all, which meant these weren't small styling decisions but ones that would shape how every subsequent contributor approached the platform's look and feel.

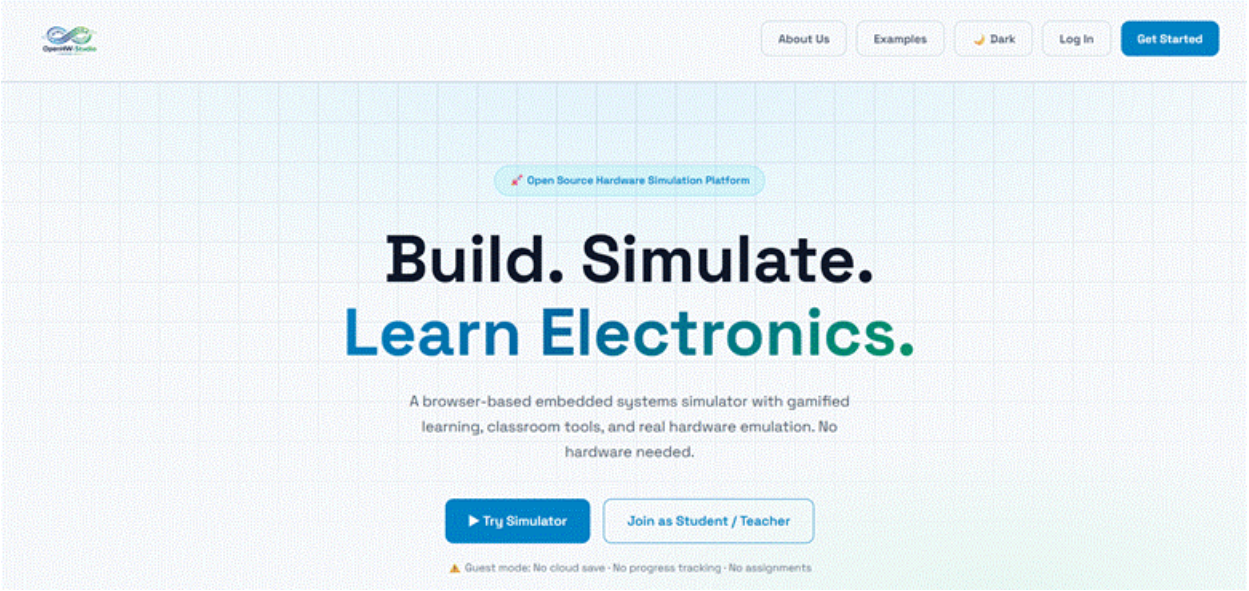
I worked through the primary user-facing pages one at a time, starting with the landing page, since that's the first impression any new user, whether a curious student or a teacher evaluating the platform for classroom use, would actually see. From there I moved through the authentication pages, the dashboard views for both Student and Teacher roles, and the general navigation shell that wraps around the simulator itself. Rather than treating each page as an isolated design task, I worked to establish a consistent visual language early, deciding on a color palette, a typography scale, spacing conventions, and recurring UI patterns like card layouts, button styles, and modal structures, so that once these decisions were made, extending them to new pages became a matter of applying an existing system rather than reinventing choices each time.

By translating the core idea into tangible, interactive layouts and navigation structures, I created an accessible, intuitive environment that bridged the gap between complex hardware simulation and a user-friendly educational interface. This mattered particularly for OpenHW Studio's target audience, since a platform simulating microcontroller pin states and SPI protocols risks feeling intimidating before a student even opens the simulator, so the surrounding pages needed to feel approachable and clearly organized,

lowering the perceived barrier to entry rather than adding to it. I paid close attention to how users would move between different parts of the platform, ensuring navigation felt predictable and that a student could always tell where they were within the broader structure, whether they were browsing available projects, checking their gamification progress, or deep inside a simulation.

This foundational work set the stage for all subsequent feature integrations. Because the visual and structural language was already established, later additions such as the Tour Guide overlay, the gamification interface, and the classroom tooling could be built to feel like natural extensions of the same product rather than bolted-on features with their own inconsistent look and feel. When I later built the Tour Guide's glassmorphism-styled tooltip panels, for instance, I was extending an existing visual vocabulary rather than introducing a new one, and the same was true for the Arduino Adventure Map's node-based interface, which needed to feel like it belonged on the same platform as the pages built months earlier. In practice, this meant that as the internship progressed and more contributors, including myself, added increasingly complex features, the platform grew without accumulating the kind of visual inconsistency that tends to show up in fast-moving, multi-contributor codebases where no one owns the overall design direction.


GENERAL ACCESS
Welcome to
OpenHW Studio
Explore, design, and simulate hardware projects in minutes. Sign in to sync your work across all devices.
Fast & Seamless
Experience a lag-free simulation environment tailored for hardware enthusiasts.
Instant Access
Continue as guest to jump straight into the simulator without an account.
OpenHW-Studio
Back to home
User Login
Access your personal dashboard and projects.
Email Address
you@example.com
Password
Sign In
OR CONTINUE WITH
Google
Last Used
Guest
Go to Classroom Login


About Us
Examples
Dark
Log In
Get Started
Open Source Hardware Simulation Platform
Build. Simulate.
Learn Electronics.
A browser-based embedded systems simulator with gamified learning, classroom tools, and real hardware emulation. No hardware needed.
Try Simulator
Join as Student / Teacher
Guest mode: No cloud save · No progress tracking · No assignments

OpenHW-Studio

LIVE PLATFORM | Dark | Dashboard

CLOUD-BASED EMULATION ENGINE

We built the lab that lives in a browser.

OpenHW-Studio is a modern open-source electronics simulation and learning platform built for students, teachers, and engineers. Get instruction-level accuracy and classroom-grade tooling right in your browser—no physical hardware required.

Try Simulator | Back to Home

18 Contributors. One open platform.

OpenHW-Studio is built by a distributed team of engineers collaborating openly under the FOSSEE program at IIT Bombay.

MD Md. Danish /danish9661	SS Satvik Sharma /Satvik-Sharma511 PROFILE	SS Sagar Seth /lightning-sagar	BN B Naga Krishna Manohar /KrishnaManohar101	VS Viraj Shah /virajsh4h	AP Aaditya Pranav /adityapranav989-ai
AS Akshat Singh Tomar /akshat440	PS Poojitha S K /geekypooky	KG Kartikay Goel /Kartikay-goel	K Kiran /kirangore20117-code	MK Manas Krishna Neigapula /manasneigapula	PM Pratham Mittal /PrathamMittal07
RR Rashmitha Rani B N /Rashmitha018	R Rima /rima46-bit	RJ Ritesh Jadhav /RiteshJadhav283	MS Mohit Sharma /sharmamohit-devops	S Sharukhh /Sharukhh69	S Snehal /snehal-git-hub

2.3 Backend Enhancements (openhw-studio-backend)

Designed and Implemented the MongoDB Gamification Data Models

I architected the UserProgress schema, designing a robust Mongoose schema to persistently track the user's gamification state, including experience points, virtual coins, current level, daily streaks, and unlocked components arrays. This schema functions as the single source of truth for everything a student has earned on the platform, so getting its structure right early mattered a great deal, since every downstream feature, the Adventure Map's visual progress, the leaderboard concept mentioned in future work, and the XP-based unlocking logic itself, all depend on reading and writing against this same document reliably. I made deliberate choices around how to model streaks and unlocked components as part of this schema rather than as separate collections, since a student's gamification state is almost always read and updated together as a single unit, and keeping it in one document meant the frontend's GamificationContext could hydrate itself with a single query rather than assembling state from several different sources on every load.

I also engineered sub-document schemas for historical tracking, implementing schemas for QuizAttempts and CompletedProjects to securely log user timestamps, scores, pass/fail states, and total XP earned per module. Unlike the UserProgress schema, which represents a student's current state, these sub-documents exist specifically to preserve history that current state alone would otherwise discard. If a student retakes a quiz or resubmits a project, the current-state fields update to reflect their latest attempt, but the underlying QuizAttempts and CompletedProjects records retain every prior attempt with its own timestamp and outcome, which is what makes it possible to later analyze patterns like how many attempts a concept typically takes, or which modules students tend to struggle with and retry most often, exactly the kind of historical signal the predictive analytics features described in the future work chapter would need to train against.

I developed intelligent instance methods, encapsulating core gamification business logic directly into the Mongoose model itself, with methods like progress.addXP(), progress.unlockComponent(), and progress.updateStreak(), to ensure data consistency and prevent frontend manipulation. The reasoning behind this design choice was fairly deliberate: XP, coins, and unlocked components function as the platform's internal currency, and if the logic for awarding or modifying them lived in route handlers or, worse, was ever trusted from client-provided values, a student could trivially inflate their own XP by intercepting and modifying a network request. By instead encapsulating this logic as methods on the model itself, addXP() handles the arithmetic and any related side effects like triggering a level recalculation, unlockComponent() handles adding an entry to the unlocked array while guarding against duplicates, and updateStreak() handles the date comparison logic needed to determine whether a streak continues, resets, or increments, every code path that touches gamification state, regardless of which route eventually calls it, goes through the same consistent, trusted logic rather than each endpoint reimplementing its own version and risking subtle inconsistencies between them.

Engineered the Leveling and Progression Algorithms

I built the automated leveling system, creating a dynamic threshold algorithm that evaluates a user's accumulated XP against a predefined LEVEL_THRESHOLDS array, automatically recalculating and persisting level-ups on the server side. Concretely, the array stores the cumulative XP required to reach each successive level, level two might require 100 XP, level three 250, level four 450, and so on, and whenever a student's XP total changes, whether from completing a quiz, finishing a project, or maintaining a streak, the algorithm walks this array to determine the highest threshold their current XP has crossed, which becomes their level. This check runs as part of the same server-side operation that awards XP in the first place, so a level-up is never a separate, out-of-sync process that

could theoretically fall behind or contradict the XP total it's supposed to reflect.

Rather than storing a student's level as an independently editable field, the algorithm treats level as a derived value, computed from their current total XP against the threshold array, which means a student's displayed level is always guaranteed to be internally consistent with their actual XP rather than something that could drift out of sync or be manipulated independently. This distinction matters more than it might initially seem to. If level were stored as its own field, updated by a separate write whenever a level-up condition happened to be noticed, there would always be some risk of the two values disagreeing, a bug in one code path incrementing XP without triggering the corresponding level check, for instance, or a race condition between concurrent requests leaving the level field stale. By instead defining level purely as a function of XP, recalculated every time XP changes rather than cached and updated separately, that entire category of inconsistency becomes structurally impossible rather than something that has to be carefully guarded against.

Structuring the thresholds as an array specifically, rather than a fixed formula, was a deliberate choice too, since it means the difficulty curve for leveling up isn't locked into a single mathematical relationship and can be tuned by simply editing the array, letting early levels come quickly to keep new students motivated while later levels require proportionally more XP, without needing to touch the underlying leveling logic itself. Had I instead hardcoded the threshold as a formula, something like level times a fixed multiplier, adjusting the pacing of progression later would mean changing the actual leveling algorithm and reasoning about how that formula change affects every level simultaneously. With an array, each threshold is an independent, directly editable value, so the platform's designers can flatten the curve early on to reward new students quickly for their first few actions, then steepen it at higher levels to keep long-term progression meaningful, entirely through configuration rather than code changes, and without any risk of introducing a bug into the leveling logic itself in the process.

Developed the Gamification RESTful API

I built secure Express.js endpoints, developing the /api/gamification/state and /api/gamification/unlocks routes to handle fetching and syncing the gamification state between MongoDB and the frontend's GamificationContext. I split these into two distinct endpoints rather than a single monolithic one because they serve different purposes with different expected call patterns: the state endpoint is what hydrates the frontend with a student's full current progress, XP, level, coins, and streak, typically on initial page load or context initialization, while the unlocks endpoint deals specifically with the unlocked-components list, which the component registry and Adventure Map need to check against far more frequently as a student navigates the platform, without needing to pull the entire progress object each time.

I implemented secure data updates using JWT authentication middleware, protectRoute, to verify a student's identity before allowing any PUT requests to mutate XP, complete projects, or unlock new hardware components in their personal toolbox. Every mutating request first passes through this middleware, which validates the JWT included in the request, extracts the authenticated user's identity from it, and only then allows the request to proceed to the actual route handler, at which point that same verified identity is used to scope the database operation to that specific student's document. This closes off what would otherwise be a fairly obvious vulnerability, since without this check in place, nothing would technically stop one authenticated student's request from attempting to modify another student's progress simply by changing an ID in the request payload.

I also optimized database queries, creating static findOrCreate methods to seamlessly initialize new user profiles on their first login, reducing frontend overhead and ensuring a smooth onboarding experience. Without this method, a brand-new student's first request to the state endpoint would either need to be preceded by a separate profile-creation step, adding friction and an extra round trip before they could see any gamification data at all, or the endpoint itself would need conditional logic scattered across

the codebase to handle the case of a missing document. The findOrCreate pattern instead consolidates that logic into a single, reusable static method on the model, which either returns the student's existing progress document if one exists, or transparently creates a fresh one with sensible default values, XP at zero, no unlocked components, and so on, if it doesn't, so the rest of the application can simply assume a progress document always exists once this method has been called, without needing to special-case new users anywhere else in the code.

2.4 Summary

These contributions represent a significant expansion of OpenHW Studio's capabilities across all layers of the stack.

Gamified Learning Ecosystem

I architected and implemented the Arduino Adventure Map across both the React frontend and Node.js backend, transforming the platform from a bare sandbox into a structured, node-based educational journey, featuring automated leveling, XP tracking, daily streaks, and persistent component unlocks. This wasn't a single feature added on top of the existing platform but a full-stack system spanning the entire path from user interaction to persisted state: a student's action on the Adventure Map in the browser needed to trigger the right update through the GamificationContext, sync reliably to the MongoDB-backed UserProgress schema on the server, and then reflect back into the UI in a way that felt immediate and rewarding. Building this across both ends meant the frontend's visual progression, worlds unlocking, nodes lighting up, levels increasing, was always a faithful

reflection of trustworthy, server-verified state rather than something the client could get out of sync or manipulate independently.

Automated Assessment Engine

I engineered a robust, automated grading system capable of verifying student simulator submissions without human intervention. By leveraging topological circuit tracing and regex-based C++ static analysis, the engine autonomously scores hardware assignments and provides actionable feedback. This addressed a genuine scaling problem inherent to the platform's ambitions: a system designed to teach hardware concepts to any number of self-directed students can't realistically depend on a human reviewing every submitted circuit and codebase, so the assessment engine needed to substitute for that judgment reliably enough that a passing grade actually meant the student had built something electrically and logically correct, not just something that superficially resembled the right answer.

Expanded Hardware Catalog

The emulator's component library grew substantially under my contributions, spanning passive sensors, motor drivers, biometric modules, multi-segment displays, and input devices, making OpenHW Studio viable for a much wider range of educational projects. Each of these additions meant building or repairing a component across its full manifest, logic, and UI layers, so that it behaved electrically the way its real-world counterpart would, which is what makes the difference between a simulator students can trust and one that merely looks the part.

Stable Simulation Pipeline

Microcontroller runners for both AVR and RP2040 targets were stabilized and optimized under this work, enabling accurate signal routing and reliable code execution for all newly added components within the simulation environment. Expanding the component catalog would have been of limited value on its own if the underlying runners couldn't reliably route signals to and from those new components, so a meaningful part of this effort was

making sure growth in the catalog didn't come at the cost of the execution pipeline's reliability.

Improved Educational Value & UX

By integrating the interactive Tour Guide alongside the Gamification system and resolving demo project routing issues, the barrier to entry was drastically lowered. Learners with no prior hardware experience can now immediately engage with working examples in a structured, highly rewarding environment. Taken together, these pieces address onboarding from two different angles at once: the Tour Guide orients a student inside the interface itself, while the Adventure Map orients them within the broader learning journey, so a complete beginner has both a clear sense of how to use the platform and a clear sense of what to do next, rather than facing either a confusing interface or an open-ended blank canvas with no starting point.

2.5 References

1. https://github.com/OpenHW-Studio
2. https://github.com/OpenHW-Studio/OpenHW-studio-frontend
3. https://github.com/OpenHW-Studio/openhw-studio-emulator
4. https://github.com/OpenHW-Studio/openhw-studio-backend
5. https://github.com/OpenHW-Studio/openhw-studio-examples
6. https://github.com/OpenHW-Studio/openhw-studio-docs
7. https://openhw-studio.fossee.in/
8. https://fossee.in/ — FOSSEE, IIT Bombay

Chapter 3
Conclusion

This internship has been a deeply enriching and rewarding experience, both technically and creatively. Contributing to OpenHW Studio not only challenged me but also gave me the opportunity to grow significantly as a developer. The process of designing, implementing, debugging, and refining each feature felt less like working through a checklist and more like building something genuinely worth using, a platform I could imagine an actual classroom relying on. There were moments, particularly while tracing the topological wiring logic in the assessment engine or hunting down why the MAX7219's SPI latching sequence occasionally misfired, where a problem sat unresolved for longer than I would have liked, and the eventual solution felt less like a checklist item completed and more like a genuine puzzle worked through, which is a different and more satisfying kind of learning than most coursework offers.

Working within a live, open-source codebase also taught me lessons that go beyond any single feature. I learned to write code that other contributors could read and extend without needing me to explain it in person, to think carefully about edge cases, like the NaN boundary conditions in the MAX30102 simulation, or the alternative valid wiring paths the assessment engine needed to accept, before they became bugs or false failures in someone else's hands, and to balance technical correctness with the practical realities of a platform still actively evolving around me as I worked. This last point in particular was a genuine adjustment. In a coursework setting, requirements are fixed before you start; here, the component registry, the runner architecture, and even the visual language I was establishing early on were all things I sometimes had to revisit and reconcile with decisions made elsewhere in the codebase around the same time, which meant staying flexible without letting that flexibility compromise correctness.

Collaborating alongside a wider team of contributors, each working on different parts of the same ecosystem, gave me a much deeper appreciation for what sustained, real-world open-source development actually looks like, the coordination, the small compromises, and the shared responsibility for a codebase that outlives any one contributor's involvement in it. Knowing that the gamification schemas I wrote, or the emulator components I fixed, would be built on by contributors after me changed how I approached the work itself; it pushed me toward writing things clearly and defensively rather than just functionally, since the cost of an unclear decision wouldn't just be paid by me.

I am sincerely grateful to Prof. Prabhu Ramchandran, Mr. Rajesh Kushalkar, Mr. Pratik Bhosale, and Mr. Pratik Nemane for their constant guidance, support, and encouragement throughout the internship. Their feedback and insights played a vital role in shaping the direction and outcome of every contribution made, and their willingness to engage seriously with both the small implementation details and the larger architectural decisions helped me develop a much sharper sense of what "done" actually means on a platform meant for real students to use.

I believe this upgraded version of OpenHW Studio holds great potential as a free, accessible educational and research tool for hardware learning. By combining structured, live-classroom management, gamified progression, and real instruction-level circuit simulation, it can empower students and educators who don't have access to physical lab setups, making hardware education more intuitive and far less intimidating for newcomers. The gamification layer in particular feels like more than a surface-level addition; it reframes learning hardware as a guided journey rather than an intimidating blank canvas, which could meaningfully lower the barrier to entry for students encountering embedded systems for the first time. I've come to see this as the internship's most transferable insight: that good educational tooling isn't primarily about the depth of the underlying simulation, though that matters too, but about designing the path a beginner takes through it.

Looking ahead, I'm excited about the possibilities this platform opens up, and the role it could play in fostering innovation and accessible learning in embedded systems education. This experience has left me with a stronger foundation in full-stack development and a genuine enthusiasm for continuing to contribute to open-source educational tools beyond the scope of this internship. If anything, working on OpenHW Studio has made open-source contribution feel less like an abstract ideal I might get around to eventually and more like something I actively want to keep doing, on this platform and elsewhere.

Chapter 4
Future Work

While the current iteration of OpenHW Studio successfully establishes a robust foundation for virtual hardware education, there are several key areas where the platform can be expanded to further enhance the educational experience and technical capabilities.

4.1 Comprehensive Gamification Ecosystem Expansion

The immediate next step for the gamification engine is to scale the curriculum content that currently sits on top of the infrastructure already built during this internship.

Full Component Integration would expand the Arduino Adventure Map to encompass all 80 to 100 available components in the emulator library, including dedicated learning nodes, quizzes, and automated assessments for advanced modules like biometric sensors, OLED displays, and complex motor drivers. At present, the Adventure Map covers a meaningful subset

of the catalog, enough to validate the entire system end to end, from the GamificationContext through the UserProgress schema to the assessment engine, but the majority of components still sit outside any structured learning path. Extending coverage to the full catalog is largely a content and configuration problem rather than an architectural one, since the underlying systems, the five-node learning flow, the XP and unlock logic, and the topological wiring and regex-based code assessment, were built generically enough to support any component rather than being hardcoded around the specific ones covered so far. What this expansion mainly requires is writing new theory content, quiz questions, and project configuration objects for each remaining component, along with defining the correct wiring and code-pattern expectations the assessment engine should validate against for each one.

Capstone "Boss" Projects would introduce multi-stage, complex projects that require students to combine four to five previously unlocked components, for example building a complete Smart Home system using a temperature sensor, LCD, servo, and buzzer together, to test holistic understanding rather than isolated, single-component skills. The current assessment engine validates a project against a single, fixed configuration object describing expected components, wiring, and code patterns, and extending it to capstone projects would mean the same three evaluation criteria, component validation, topological wiring analysis, and static code analysis, need to reason about a significantly larger and more interconnected circuit, where a resistor or ground pin might legitimately serve multiple components at once rather than belonging to a single isolated path. This is less a new system than a stress test of the existing one at greater scale and complexity.

Dynamic Leaderboards & Social Learning would implement global and classroom-specific leaderboards based on XP and streak counts, to foster healthy competition among students. The historical QuizAttempts and CompletedProjects schemas built during the internship already log the timestamped, per-module data that a leaderboard would need to rank students meaningfully, whether by total XP, current streak length, or

completion speed on specific modules, so this feature is largely additive: a new aggregation query and a frontend view rather than a change to how gamification data is captured in the first place.

4.2 Advanced Live Classroom & Collaboration

The Live Classroom feature can be evolved from a monitoring tool into a deeply collaborative environment.

Real-Time Multiplayer Simulation would implement synchronous, multi-user breadboard editing, similar to Google Docs or Figma, allowing a group of students to wire a circuit and write code collaboratively on the same virtual canvas in real time. This is a meaningfully harder problem than it might first appear, since the current wiring and canvas state is designed around a single user editing a single session's state. Supporting true concurrent multiplayer editing would likely require a conflict-resolution strategy suited to simultaneous edits, such as Conflict-free Replicated Data Types (CRDTs) or Operational Transformation, the same class of techniques that underpin tools like Google Docs, so that two students moving components or drawing wires at the same moment converge on a consistent shared state rather than overwriting or corrupting each other's changes. This would also need a real-time transport layer, most likely WebSockets, to push each collaborator's changes to everyone else's canvas with low enough latency that the experience feels genuinely live rather than laggy.

Predictive Analytics for Educators would enhance the teacher dashboard with machine learning that analyzes student assessment failures and time spent on task, automatically flagging students who are struggling with specific concepts, like pull-up resistors or PWM, so teachers can intervene early rather than only after a student has fallen noticeably behind. This feature is a fairly direct extension of the QuizAttempts and CompletedProjects schemas already built, since those sub-documents already capture exactly the kind of longitudinal, per-student, per-module signal, repeated failed attempts, unusually long time-to-completion, specific criteria that keep failing on the assessment engine, that a model would

need to identify a struggling student before a teacher would otherwise notice.

4.3 AI-Powered Evaluation and Tutoring

AST-Based & AI Code Analysis would upgrade the current regex-based static code analyzer to utilize Abstract Syntax Trees, or integrated Large Language Models, allowing the system to understand complex, non-standard student logic and provide highly specific, conversational debugging hints. The regex-based parser built during this internship works well for recognizing known, expected patterns, confirming a pinMode() call exists with the right direction, or that digitalWrite() alternates HIGH and LOW, but it fundamentally pattern-matches against text rather than understanding code structurally, which means a student who solves a problem correctly but writes it in an unanticipated way, using a helper function, a different variable naming convention, or a loop structure the regex patterns weren't written to recognize, risks being marked incorrect despite functionally correct logic. An AST-based approach would instead parse the code into its actual syntactic structure, letting the system reason about what the code does rather than merely how it's phrased, and pairing this with an LLM could go further still, generating natural-language, context-specific feedback tailored to a student's actual mistake rather than only a fixed message tied to a specific failed pattern check.

Physics-Accurate Component Failure would enhance the simulation engine to actively teach safety. If a student wires an LED without a current-limiting resistor or creates a short circuit, the emulator would visually simulate the component burning out, with a smoke animation, and explain the underlying electrical physics behind the failure, turning what is currently either a silent non-event or a generic error message into an actual teaching moment. This is one of the more common and important mistakes beginners make with real hardware, since without a current-limiting resistor, an LED will draw far more current than it's rated for and burn out almost immediately, and giving students a safe, consequence-free way to see that failure happen, and understand why, mirrors a lesson that's usually only learned the hard way with physical components.

4.4 Hardware-in-the-Loop (Physical Hardware Integration)

To bridge the gap between virtual simulation and real-world engineering, future iterations should incorporate physical hardware deployment.

Web Serial Flashing would utilize the Web Serial API, a browser API that allows web pages to communicate directly with serial devices connected over USB, to let students take code they've verified in the OpenHW virtual simulator and compile or flash it directly to a physical Arduino or ESP32 plugged into their computer, entirely from the browser, without needing any separate desktop toolchain like the Arduino IDE. This would close the loop between simulation and reality in a fairly direct way: a student could debug and iterate on their circuit and code entirely in the safety of the simulator, then, once it works correctly, flash that exact verified program straight to real hardware, with the browser itself acting as the bridge between the two.

Hybrid Telemetry would allow the virtual dashboard in the studio to read and plot live telemetry data coming from a physical sensor array in the real world, rather than only from simulated components. This would let the same real-time plotting and serial monitoring tools students already learn to use in simulation apply directly to hardware they've wired themselves, reinforcing that the skills built in the simulator transfer completely, rather than being a separate, simulator-only skill set.

4.5 Mobile and Cross-Platform Accessibility

To make hardware education truly universally accessible, future development should focus on optimizing the WebGL canvas and UI components for touch devices. The current workspace assumes a mouse-and-keyboard interaction model throughout, precise click-and-drag wiring, right-click context menus for autowiring, and a code editor built around keyboard input, none of which map cleanly onto touch gestures without deliberate redesign. Creating a dedicated tablet experience, with touch-friendly component placement, wiring gestures, and a mobile-adapted code or block-editing interface, would allow students in schools lacking high-end computer labs to learn embedded systems using

only standard mobile devices, meaningfully extending the platform's reach into environments where a full desktop computer lab simply isn't available, which is often precisely the environment where a free, browser-based hardware simulator would matter most.