Summer Internship Report

on

OpenHW Studio: Expanding the Arduino Simulator with New Components and Fixes

Submitted by

Mohit Sharma

B.Tech In Electrical Engineering

Aligarh Muslim University (ZHCET)

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.

Mohit Sharma
ALIGARH MUSLIM 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.

Mohit Sharma
ALIGARH MUSLIM UNIVERSITY

Role and Responsibilities

Across the course of the internship, my responsibilities were grouped into three primary streams that map directly onto the structure of this report. The first involved proposing, analyzing, and evaluating gamified progression mechanics for the component-unlock system to improve user engagemet and learning flow. The second focused on performing structured electrical, interface, functional, and integration testing on six simulated components (LED, Li-ion battery, NPN transistor, buzzer, rotary potentiometer, and OLED display), ensuring their correct behavior within the simulation environment. The third involved authoring comprehensive technical documentation for each of these six components, covering pin references, electrical characteristics, wiring guidance, operating principles, implementation notes, and example Arduino sketches to support both beginners and advanced users.

Index

Chapter 1

Introduction

OpenHW-Studio is an open-source, cloud-based electronics simulation and learning platform developed under the FOSSEE Project at IIT Bombay. It is designed for students, teachers, and engineers who need to study and validate embedded system behaviour without depending on physical hardware. The platform combines an in-browser circuit simulator, an instruction-level AVR emulator, a multi-mode code editor, and classroom-integrated tooling into a single coherent system, all running directly in the browser without plugins or desktop installation.

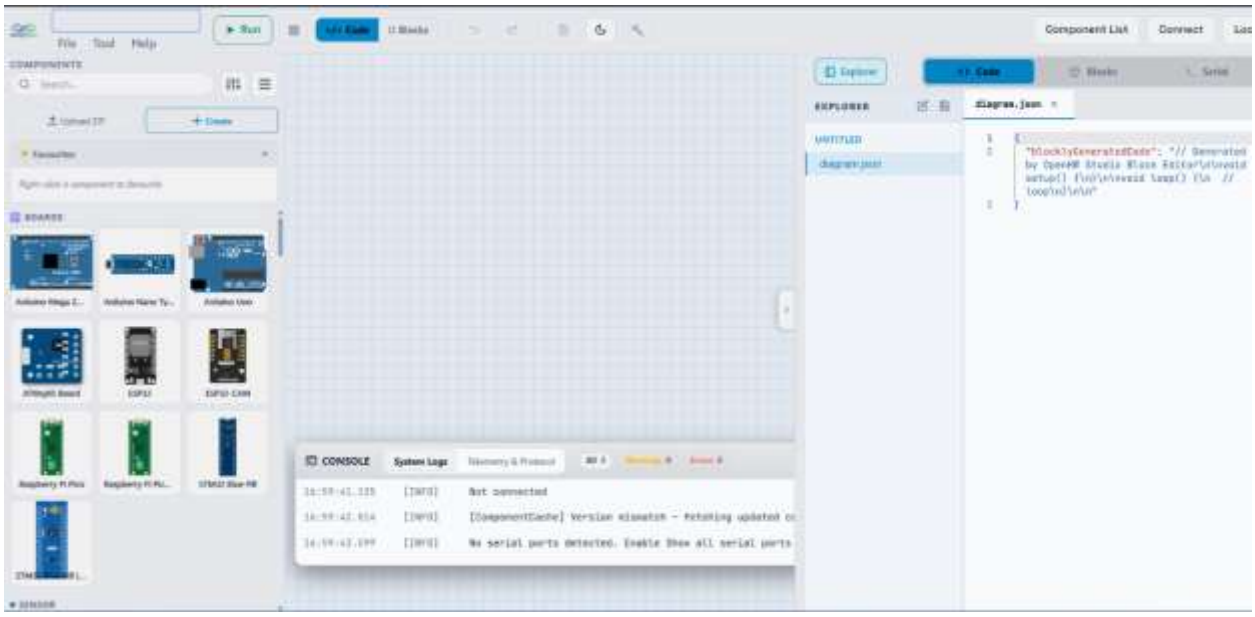
OpenHW-Studio is built and maintained collaboratively by a team of eighteen contributors on GitHub, with every component, simulation feature, and classroom tool developed, reviewed, and merged in the open. As part of this team, I worked on three areas of the project: designing the gamification layer that drives component progression, testing and validating the electrical and simulation behaviour of individual components, and documenting components for both contributors and end users.

Project Overview

OpenHW-Studio is structured around six independent systems that integrate seamlessly into a single embedded development environment: the in-browser circuit simulator, the three-level code editor, the compiler (running on a lightweight server), the shared emulator library consumed by both simulator and editor, the developer-tools layer (serial monitor and plotter), and the live classroom infrastructure. Each system was designed to function independently while integrating without seams, so that the simulator, compiler, and emulator together behave as a coherent whole rather than a loosely connected set of tools.

The circuit-building experience is centred on a drag-and-drop canvas with manual wiring, real-time electrical validation, and “smart auto-assist” — for example, automatically attaching a 220-ohm resistor to an LED or a contrast potentiometer to an LCD when these components are placed, removing a common source of beginner error. The entire simulator runs without plugins or desktop software, directly in the browser.

Firmware can be written in three modes depending on user experience level, with the mode selected before a project begins rather than bolted on afterward: a block-based Blockly editor with live C++ generation for beginners, partial scaffolding for intermediate users, and a full text editor for advanced users. This is paired with developer tools — a real-time serial monitor with timestamped output, pause/clear controls, and a multi-signal analog plotter for visualizing sensor data at full simulation speed — giving students access to the same diagnostic tools used by practising engineers.

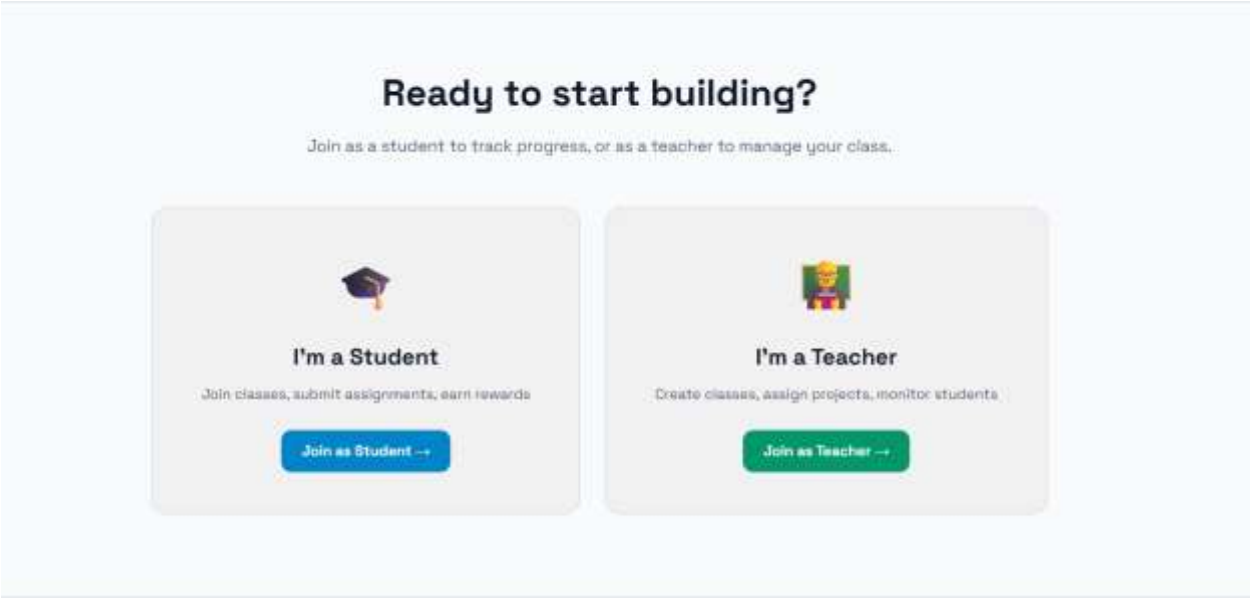


Underlying the simulation is an instruction-level AVR emulator running at a 16 MHz clock, rendering circuit and signal behaviour at 60 frames per second, so that firmware timing, interrupts, and pin-level voltage logic match what would be observed on physical hardware — the platform's stated principle being that “if it runs on OpenHW-Studio, it runs on the board.” The architecture is also offline-first: compiled machine code is cached locally in IndexedDB, projects auto-save every 2.5 seconds, and component uploads queue when offline and sync automatically on reconnection, so the backend is never a hard dependency for active work.

The hardware roadmap is phased and purposeful. Phase 1, currently active, covers instruction-level emulation of the Arduino Uno and the Raspberry Pi Pico (RP2040). Phase 2 is planned to bring API-level support for the ESP32. Phase 3, further out on the roadmap, targets full ARM instruction-level emulation of the STM32 family. This phased approach lets the platform deepen accuracy on a small set of boards before expanding breadth.

Beyond simulation, OpenHW-Studio is built as an education-first platform rather than a simulator with classroom features added on. It serves three roles in a shared environment: guests, who get immediate, account-free access to the full simulator and can write Arduino or MicroPython code and save projects locally; students, who additionally get cloud project

saving, assignment submission, and gamification; and teachers, who can broadcast their screen, push circuit templates, lock and unlock student screens, and review structured JSON-plus-code project submissions with inline grading and per-student analytics. Students are guided through a gamified progression system, where access to advanced components unlocks as competence is demonstrated through points, badges, and levels — rather than simply through time spent on the platform.



Finally, the platform is open by default and extensible by design: every repository is public, custom components can be submitted, reviewed by an administrator, and deployed to all active sessions without a server restart, and the modular architecture is built to support new MCU targets, AI-assisted grading, and institutional administration without rebuilding the foundation. It is this open, modular structure that made it possible for an eighteen-member distributed team — including my own work on gamification, component testing, and documentation — to contribute to a single coherent platform.

Internship Objectives

–The primary objectives of the internship were:

- To understand the architecture and documentation workflow of OpenHW Studio.
- To contribute towards backend documentation maintenance and improvement.
- To establish standardized documentation practices for simulator components.
- To validate and improve component documentation across multiple hardware modules.
- To enhance documentation readability and maintainability through structured Markdown organization.
- To improve component categorization and navigation within the documentation website.
- To document compatibility across supported microcontroller platforms.
- To automate repetitive documentation-related tasks wherever feasible.
- To ensure successful documentation builds by identifying and resolving parser and rendering issues.
- To contribute towards creating a scalable documentation framework for future development.

Chapter 2

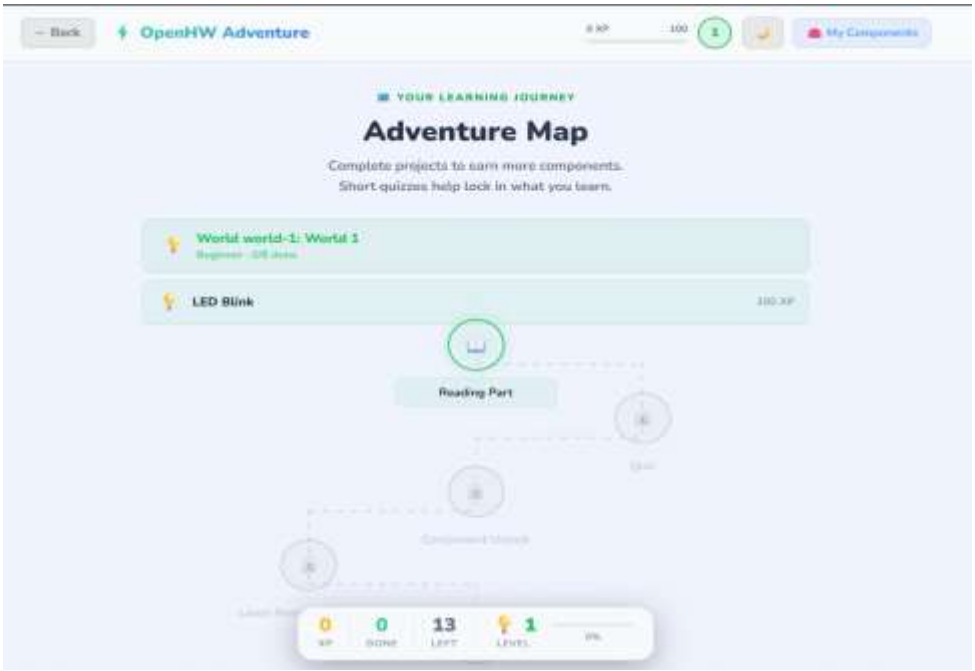
My Contributions

2.1 Gamified Component Progression System

My first area of contribution was the design of the gamification layer that governs how students unlock new electronic components as they progress through the platform. OpenHW-Studio's progression model is built around the idea that advanced components (such as displays, motors, and communication modules) should be unlocked through demonstrated competence rather than time spent on the platform, so the design work centred on what "demonstrated competence" should look like in practice.

My specific contribution here was suggesting and selecting candidate mini-games and challenge formats that could be used to test a student's understanding of a component before unlocking the next tier — for example, short interactive tasks tied to wiring correctness, reading sensor output, or predicting circuit behaviour. I evaluated multiple game concepts against criteria such as educational relevance to the underlying component, simplicity of implementation within the existing Blockly/simulation environment, and how well each concept fit into a points-badges-levels structure, and proposed a shortlist of selections to the team for the progression system.

This work fed into the platform's broader points, badges, and level system described in the platform documentation, where component access is tied to engagement with these gamified checkpoints rather than passive usage.



2.2 Testing and Validation of Components

Li-ion Battery

Component Overview: the Li-ion battery component models a single-cell lithium-ion power source, the kind commonly used to power portable Arduino-based projects independently of a USB or wall supply.

Pin	Function
+ (Positive terminal)	Power output to circuit VCC rail
− (Negative terminal)	Connects to circuit GND

Table 2.2.2: Li-ion battery pin reference used during testing.

Electrical Characteristics Verified: nominal terminal voltage (approximately 3.7 V for a single cell), discharge-curve behaviour (a relatively flat voltage region between roughly 4.2 V fully

charged and 3.0 V discharged, rather than a linear drop), and correct current delivery to downstream components.

Test Methodology: the battery was wired as the sole power source for representative downstream loads (an LED circuit and a sensor module) at several simulated charge levels, to confirm that voltage and available current behaved consistently with the expected discharge profile rather than acting as an idealized fixed-voltage source.

Test Cases and Observations: downstream components operated correctly across the mid-range of the discharge curve; behaviour at both high (near 4.2 V) and low (near 3.0 V) simulated charge states was checked to confirm that components requiring a minimum operating voltage correctly stopped functioning once the simulated battery voltage dropped below that threshold, mirroring real low-battery behaviour.

Edge Case and Negative Testing: a fully depleted (0%) simulated charge state was also tested to confirm the battery correctly supplies no usable voltage rather than an unrealistic non-zero fallback value, and a short-circuit style load was tested to observe whether the simulator reflects any current-limiting behaviour consistent with a real cell.

NPN Transistor

Component Overview: the NPN bipolar junction transistor was tested in its most common beginner role — as a digital switch used to drive a higher-current load from a microcontroller's low-current digital output pin.

Pin	Function
Base (B)	Control input, via current-limiting resistor, from a digital pin
Collector (C)	Connects to the load (e.g., motor, relay coil, LED)
Emitter (E)	Connects to GND

Table 2.2.3: NPN transistor pin reference used during testing.

Electrical Characteristics Verified: correct cutoff behaviour (no collector current when base current is absent), correct switching into saturation once sufficient base current is supplied, and the base-collector current relationship (collector current scaling with base current up to the saturation limit, consistent with the part's current gain, hFE).

Test Methodology: a switching circuit was built where the transistor's base was driven through a current-limiting resistor from a digital pin, with the collector connected to a higher-current load (such as a motor or relay coil) and the emitter to ground. The digital pin was toggled to observe switching behaviour, and the base resistor value was varied to test the boundary between cutoff and saturation.

Test Cases and Observations: with the digital pin LOW, the load correctly remained off, confirming cutoff behaviour; once the pin was driven HIGH with a sufficiently low base resistor, the transistor correctly switched into saturation and the load activated; intermediate base-resistor values correctly produced partial (linear-region) switching behaviour rather than an unrealistic instantaneous on/off response.

Edge Case and Negative Testing: the base resistor was also tested at a deliberately high value to confirm the transistor remains in cutoff (load stays off) even with the control pin HIGH, verifying that the simulator models the base-current dependency rather than treating the base pin as a simple boolean switch.

Buzzer

Component Overview: the buzzer component was tested for both common Arduino-project buzzer types — simple digital (active) buzzers, and PWM-driven (passive) buzzers typically used with the Arduino tone() function to produce different pitches.

Pin	Function

+ (Signal)	Digital or PWM output pin
– (GND)	Connects to circuit GND

Table 2.2.4: Buzzer pin reference used during testing.

Electrical Characteristics Verified: clean ON/OFF response to a digital signal for the active-buzzer use case, and frequency-dependent behaviour (not just on/off state) for the passive-buzzer use case.

Test Methodology: for the active-buzzer case, a digitalWrite() HIGH/LOW toggle was used to confirm immediate, clean activation and deactivation. For the passive-buzzer case, the tone() function was used to sweep across several frequencies to confirm that the simulator's audible-equivalent feedback (or accompanying telemetry/visual indicator) varied with frequency rather than remaining constant.

Test Cases and Observations: digital toggling produced correctly clean transitions with no spurious activation; frequency sweeps via tone() produced correspondingly different simulated output, confirming the buzzer model tracks frequency rather than simply toggling a fixed tone on and off.

Edge Case and Negative Testing: the noTone() function was tested mid-sweep to confirm the buzzer correctly silences immediately rather than completing the current cycle, and rapid back-to-back tone()/noTone() calls were tested to check for any audible glitching or stuck-on state in the simulation.

Rotary Potentiometer

Component Overview: the rotary potentiometer is a three-terminal variable resistor whose wiper position determines the analog voltage seen by a connected analog input pin, and is the standard component used to teach analogRead() in introductory Arduino courses.

Pin	**Function**

Terminal 1	Connects to 5V
Wiper (middle terminal)	Connects to an analog input pin (e.g., A0)
Terminal 2	Connects to GND

Table 2.2.5: Rotary potentiometer pin reference used during testing.

Electrical Characteristics Verified: linearity of the analog output across the full mechanical range of the wiper, and correct mapping of wiper position to the expected analogRead() range (0–1023 on a 10-bit ADC).

Test Methodology: the potentiometer was wired in a standard voltage-divider configuration between 5 V and GND, with the wiper connected to an analog input pin. The simulated wiper position was swept from one extreme to the other while the analogRead() value was logged via the Serial Monitor and plotter.

Test Cases and Observations: the analogRead() value scaled linearly with wiper position across the full sweep, with no discontinuities or non-linear regions near either extreme; the value correctly reached close to 0 and close to 1023 at the two ends of the sweep, confirming correct interfacing with the analog input pin.

Edge Case and Negative Testing: the potentiometer was also tested with the wiper terminal accidentally swapped with one of the outer terminals, to confirm the simulator's auto-assist/validation logic flags this as an incorrect wiring configuration rather than silently producing a misleading reading.

OLED Display

Component Overview: the OLED display tested is a standard I2C-driven module (commonly using the SSD1306 driver, at the conventional I2C address 0x3C), used in projects requiring simple text or graphic output without a dedicated graphics controller.

Pin	Function
VCC	Power supply input (3.3V–5V)
GND	Ground reference
SCL	I2C clock line
SDA	I2C data line

Table 2.2.6: OLED display pin reference used during testing.

Electrical Characteristics and Interface Verified: correct I2C address recognition, correct initialization sequence, and correct rendering of drawing primitives sent over I2C.

Test Methodology: the display was wired over I2C (SDA/SCL) to an Arduino Uno, and a representative sketch using a standard display library (such as Adafruit_SSD1306) was used to initialize the display and draw text, lines, and simple shapes. The sketch was also used to update the display content mid-simulation to test live-refresh behaviour.

Test Cases and Observations: the display correctly initialized when powered and addressed correctly over I2C; text and basic shape primitives rendered correctly; updated content written by the sketch during simulation was correctly reflected on the display without requiring a simulation restart, confirming correct live I2C communication between the emulated microcontroller and the display component.

Edge Case and Negative Testing: an incorrect I2C address was deliberately specified in the sketch to confirm that the display correctly fails to initialize (rather than silently working anyway), and SDA/SCL lines were swapped to confirm the simulator's wiring validation correctly flags the misconfiguration.

Across all six components, testing involved building representative circuits on the simulation canvas, running firmware through the platform's programming modes, and cross-checking simulated behaviour (timing, voltage response, and signal output) against expected real-world electrical characteristics. Where discrepancies were found, these were reported back

to the development team for correction, contributing to the platform's broader goal of “simulation that behaves like real silicon.”

2.3 Documentation of Components

My third area of contribution was documenting the six components I tested, so that both contributors and end users (students and teachers) could understand their correct usage within the platform.

Documentation Structure and Standards

For each component, documentation followed a consistent five-part structure to keep the platform's component library easy to navigate and consistent across contributors.

- Component overview — a short description of the physical device being modeled and its typical real-world use cases.
- Pin reference table — listing every exposed pin, its function, and correct connections (e.g., VCC/GND, signal lines, communication lines where applicable).
- Electrical characteristics — operating voltage range, current draw, and any thresholds relevant to correct operation (e.g., LED forward voltage, transistor saturation behaviour, potentiometer range).
- Wiring guidance — a description of a correct minimal circuit, including any resistors or supporting components required (e.g., current-limiting resistor for the LED, pull-down/pull-up considerations for the transistor and buzzer circuits).
- Example Arduino sketch — a minimal working sketch demonstrating correct usage of the component, along with expected Serial Monitor or visual output.

Since OpenHW-Studio's component library is open and extensible — with every repository public and custom components reviewable and deployable without a server restart — accurate, consistent component documentation is necessary both for new contributors extending the library and for teachers building classroom assignments around specific

components. The same five-part structure was applied to each of the six components I tested, summarized below.

LED Documentation

Pin	Function
Anode (+)	Connects to current-limiting resistor / digital or PWM output pin
Cathode (−)	Connects to GND

Table 2.3.2: LED pin reference, as published in the documentation page.

The LED documentation page opens with a short description identifying the component as a current-driven, polarity-sensitive semiconductor light source, and lists the two physical terminals (anode and cathode) alongside the pin reference table above.

Electrical characteristics documented: forward voltage threshold of approximately 1.8–2.2 V, the requirement for a current-limiting resistor (with 220 Ω noted as the platform's default auto-assist value), and a caution note explaining that omitting the resistor in a real circuit — even though the simulator auto-assists — would risk damaging a physical LED.

Wiring guidance documented: the anode connects through the resistor to a digital or PWM-capable pin, and the cathode connects to GND; reversed polarity is explicitly called out as a common beginner mistake that results in no illumination rather than damage, since LEDs are reverse-blocking at typical Arduino voltage levels.

The accompanying example sketch (Appendix A.1) demonstrates PWM brightness control using analogWrite(), chosen because it exercises both the basic digitalWrite() use case implicitly and the more advanced PWM fading behaviour in a single example.

Li-ion Battery Documentation

Pin	Function
+ (Positive terminal)	Power output to circuit VCC rail
− (Negative terminal)	Connects to circuit GND

Table 2.3.3: Li-ion battery pin reference, as published in the documentation page.

The Li-ion battery documentation page describes the component as a single-cell lithium-ion power source intended for portable, USB-independent circuits, and explains the two-terminal interface shown above.

Electrical characteristics documented: the nominal 3.7 V single-cell voltage, the approximate 4.2 V (full charge) to 3.0 V (discharged) operating range, and a note that the voltage is not constant across this range — it stays relatively flat through most of the discharge cycle before dropping more steeply near depletion.

Wiring guidance documented: the positive terminal feeds the circuit's VCC rail and the negative terminal connects to GND, with a reminder that any downstream component with a minimum operating voltage requirement (as tested for several components in Section 2.2) should be expected to stop functioning once the simulated battery voltage drops below that minimum.

The accompanying example sketch (Appendix A.2) shows the battery powering a simple LED indicator circuit, kept deliberately minimal so the focus stays on the power-source behaviour rather than the load.

NPN Transistor Documentation

Pin	Function
Base (B)	Control input, via current-limiting resistor, from a digital pin
Collector (C)	Connects to the load (e.g., motor, relay coil, LED)
Emitter (E)	Connects to GND

Table 2.3.4: NPN transistor pin reference, as published in the documentation page.

The NPN transistor documentation page introduces the component as a current-controlled switch and lists the base, collector, and emitter pin functions shown above, since correctly identifying these three terminals is the most common point of confusion for students encountering a transistor for the first time.

Electrical characteristics documented: the cutoff/saturation switching behaviour, the role of the base current-limiting resistor in setting the operating point, and a note on current gain (hFE) explaining why collector current scales with, but is much larger than, base current.

Wiring guidance documented: the base connects through a resistor to a digital pin, the collector connects to the load (with the load's other terminal to VCC), and the emitter connects to GND — explicitly distinguished from a low-side vs. high-side switching arrangement to avoid a common wiring error.

The accompanying example sketch (Appendix A.3) demonstrates the transistor switching a load on and off via a digital pin, directly mirroring the test methodology described in Section 2.2.3.

Buzzer Documentation

Pin	Function
+ (Signal)	Digital or PWM output pin
– (GND)	Connects to circuit GND

Table 2.3.5: Buzzer pin reference, as published in the documentation page.

The buzzer documentation page distinguishes between active (digital on/off) and passive (PWM/tone-driven) buzzer types up front, since the two require different code patterns despite sharing the same two-pin interface.

Electrical characteristics documented: the simple HIGH/LOW activation behaviour expected for an active buzzer, and the frequency-dependent tone() behaviour expected for a passive buzzer, with a note that supplying a constant digitalWrite() HIGH to a passive buzzer will not reliably produce sound.

Wiring guidance documented: the signal pin connects to a digital or PWM-capable output pin, and the ground pin connects to GND, with no additional resistor required for typical 5 V Arduino-compatible buzzer modules.

The accompanying example sketch (Appendix A.4) demonstrates a tone() frequency sweep, chosen specifically to make the frequency-dependence of the passive buzzer behaviour visible to a student reading the documentation.

Rotary Potentiometer Documentation

Pin	Function
Terminal 1	Connects to 5V
Wiper (middle terminal)	Connects to an analog input pin (e.g., A0)
Terminal 2	Connects to GND

Table 2.3.6: Rotary potentiometer pin reference, as published in the documentation page.

The rotary potentiometer documentation page introduces the component as a three-terminal variable resistor used in a voltage-divider configuration, and lists the pin functions shown above.

Electrical characteristics documented: the expected 0–1023 analogRead() range on a 10-bit ADC, and the linear relationship between wiper position and output voltage that was specifically verified during testing in Section 2.2.5.

Wiring guidance documented: the two outer terminals connect to 5 V and GND (interchangeably, since this only reverses the direction of the sweep), while the wiper (middle terminal) connects to an analog input pin — with an explicit warning, informed by the edge-case test in Section 2.2.5, against mistakenly wiring the analog input to one of the outer terminals instead of the wiper.

The accompanying example sketch (Appendix A.5) reads and prints the potentiometer value to the Serial Monitor, kept intentionally simple as the canonical first analogRead() example.

OLED Display Documentation

Pin	Function
VCC	Power supply input (3.3V–5V)
GND	Ground reference
SCL	I2C clock line
SDA	I2C data line

Table 2.3.7: OLED display pin reference, as published in the documentation page.

The OLED display documentation page identifies the component as a standard SSD1306-driven I2C display and lists the four-pin interface shown above, along with the library dependency (Adafruit_SSD1306, plus the underlying Wire library for I2C) required to use it.

Electrical characteristics documented: the default I2C address (0x3C) and a note on how to recognize an address-mismatch failure (the display simply fails to initialize), informed directly by the negative test case in Section 2.2.6.

Wiring guidance documented: VCC and GND to power, SCL and SDA to the microcontroller's I2C clock and data lines respectively, with an explicit warning against swapping SDA and SCL — again informed by the corresponding edge-case test in Section 2.2.6.

The accompanying example sketch (Appendix A.6) initializes the display and prints a short text string, providing the minimal “hello world” starting point for students building more complex display-driven projects.

2.4 Cross-Component Summary

The table below summarizes the communication/interfacing style and the primary characteristic verified for each of the six tested components, as a quick-reference complement to the detailed subsections above.

Component	Interface Type	Primary Characteristic Verified
LED	Digital / PWM	Forward-voltage conduction and PWM brightness scaling
Li-ion Battery	Power source	Discharge-curve voltage and minimum-operating-voltage cutoff
NPN Transistor	Digital (switch)	Cutoff/saturation switching and base-current dependency
Buzzer	Digital / PWM	Clean on/off response and frequency-dependent tone() behaviour
Rotary Potentiometer	Analog	Linearity of analogRead() across the full wiper sweep
OLED Display	I2C	Correct address recognition, initialization, and live rendering

Chapter 3

TECHNOLOGIES USED AND DEVELOPMENT ENVIRONMENT

3.1 Introduction

The successful development and maintenance of a modern open-source software project require the integration of multiple technologies, development tools, documentation frameworks, and collaborative platforms. During my internship with the FOSSEE Open Source Hardware (OSHW) Project, I worked primarily within the backend documentation ecosystem of OpenHW Studio.

Although my contributions focused on documentation engineering and component validation, they required interaction with several technologies spanning web development, documentation frameworks, version control systems, automation tools, and collaborative development environments.

This chapter presents the software tools, technologies, programming languages, development environment, and workflow adopted during the internship.

3.2 Technology Stack

The OpenHW Studio documentation ecosystem is built using modern web technologies that facilitate collaborative development and scalable documentation management. Throughout the internship, the following technologies were utilized.

Table 3.1 Technology Stack

Category	Technologies
Programming Languages	JavaScript, HTML5, CSS3, Markdown
Runtime Environment	Node.js
Documentation Framework	VitePress
Version Control	Git
Repository Hosting	GitHub
IDE	Antigravity IDE
AI Development Assistant	Gemini (Integrated within Antigravity IDE)
Graphics Format	SVG
Markup Language	Markdown

Build Tools	npm, Node.js Scripts

3.3 Programming Languages JavaScript

JavaScript was extensively used within the OpenHW Studio project for backend utilities, documentation automation scripts, validation logic, and build-related processes.

During the internship, JavaScript-based scripts were analyzed and utilized for automating documentation tasks, maintaining documentation consistency, and simplifying repetitive maintenance operations.

Markdown

Markdown served as the primary documentation language for OpenHW Studio.

Every simulator component documentation page was maintained as an independent Markdown file.

Using Markdown provided several advantages:

- Easy version control
- Human-readable syntax
- Maintainability
- VitePress compatibility
- Structured documentation generation

A significant portion of my internship involved restructuring, validating, and improving Markdown documentation files.

HTML

The original documentation of several components existed in HTML format.

These HTML files served as reference implementations during the documentation migration and standardization process.

Their structure, SVG illustrations, wiring diagrams, and component descriptions were carefully analyzed before being incorporated into the Markdown documentation system.

CSS

Although frontend styling was not the primary focus of my internship, an understanding of CSS was required while ensuring that documentation pages rendered correctly within the VitePress environment.

Knowledge of CSS also assisted in understanding layout issues affecting documentation presentation.

3.4 Documentation Framework – VitePress

OpenHW Studio utilizes VitePress, a modern static site generator powered by Vue.js and Vite, for managing its technical documentation.

VitePress offers several features that make it suitable for documentation-driven projects:

- Fast documentation builds
- Markdown support
- Automatic sidebar generation
- Responsive layouts
- Built-in search functionality
- Dark mode support
- Navigation support
- Component embedding

Throughout the internship, documentation pages were continuously validated against VitePress build requirements to ensure successful compilation without parser errors.

3.5 Version Control Using Git and GitHub

The OpenHW Studio project follows a collaborative development workflow using Git and GitHub.

Version control played a crucial role during the internship by enabling:

- Feature development
- Branch management
- Pull request creation
- Code reviews
- Merge conflict resolution
- Documentation updates
- Collaborative development

During the internship, I regularly synchronized my local repository with the central repository, managed development branches, resolved merge conflicts, and prepared changes for pull request submission.

Working within a real-world Git workflow significantly enhanced my understanding of collaborative software development

3.6 Development Environment

The primary development environment used throughout the internship was Antigravity IDE, which provided an integrated workspace for project development and documentation maintenance.

The IDE facilitated:

- Project navigation
- Source code management
- Documentation editing
- Git integration
- Terminal access
- AI-assisted development

Its integrated tooling enabled efficient management of documentation files while simplifying interaction with the large OpenHW Studio repository.

3.7 AI-Assisted Development Workflow

During the internship, AI-assisted development tools were used as productivity aids to accelerate repetitive tasks such as documentation drafting, structural improvements, code explanation, and validation.

These tools assisted in:

- Drafting technical documentation
- Improving Markdown structure
- Reviewing documentation consistency
- Suggesting documentation improvements
- Explaining unfamiliar code sections

Chapter 4

Summary

My **six-month** FOSSEE internship with **the OpenHW-Studio team (under IIT Bombay**) involved three interconnected streams of contribution: gamification design, component testing and validation, and component documentation.

Gamification. **OpenHW-Studio's** progression model unlocks advanced components based on demonstrated competence rather than passive time spent on the platform. My contribution here was design-stage: I evaluated candidate mini-game and checkpoint formats against three criteria — educational relevance to the underlying component, feasibility of implementation within the existing Blockly/simulation canvas, and how cleanly each format could feed into the platform's points-badges-levels structure. From this, four candidate formats were shortlisted: wiring-accuracy challenges (reproducing a target circuit correctly), signal-reading challenges (interpreting sensor output via the Serial Monitor/plotter), predict-the-output challenges (reasoning about circuit/code behaviour before running it), and debug-the-circuit challenges (finding and fixing a deliberately broken circuit). These were mapped onto progression tiers — simpler formats for early components like LEDs and buzzers, more advanced formats for protocol-driven components like I2C/SPI sensors and displays — so that each unlocked badge corresponds to a real, demonstrable skill.

Testing and Validation. I performed structured, three-layer testing (electrical correctness, interface correctness, and integration correctness inside representative Arduino sketches) on six simulated components: LED, Li-ion battery, NPN transistor, buzzer, rotary potentiometer, and OLED display. For each, I verified specific real-world electrical characteristics in the simulator — for example, the LED's forward-voltage conduction threshold and correct auto-assist resistor behaviour, the Li-ion battery's discharge-curve

voltage profile, the NPN transistor's cutoff/saturation switching behaviour, the buzzer's frequency-dependent tone() response, the potentiometer's analogRead() linearity across its full sweep, and the OLED display's I2C initialization and live-rendering behaviour. Beyond nominal test cases, I also ran edge-case and negative tests (e.g., reverse polarity, incorrect I2C address, swapped potentiometer wiring) to confirm the simulator correctly flags or reflects these faults rather than masking them.

Documentation. For the same six components, I authored documentation following a consistent five-part structure: a component overview, a pin reference table, electrical characteristics, wiring guidance, and a minimal example Arduino sketch. This was meant to keep the open, contributor-extensible component library consistent and usable both for new contributors and for teachers building classroom assignments.

Together, these three streams supported OpenHW-Studio's broader goals — instruction-level simulation accuracy, classroom-grade tooling, and an open, modular component ecosystem maintained collaboratively by an eighteen-member contributor team.

Conclusion

Working on **OpenHW-Studio** as part of the **FOSSEE Internship** has been a valuable learning experience, combining hands-on electronics validation with platform design thinking. Testing components such as the LED, Li-ion battery, NPN transistor, buzzer, rotary potentiometer, and OLED display required me to think carefully about how simulated behaviour should map onto real electrical characteristics, while contributing to the gamification design pushed me to think about component access from an educational and engagement perspective rather than a purely technical one.

I am grateful to my mentors, **Rajesh Kushalkar, and Pratik Bhosale**, and to the wider eighteen-member contributor team, for their guidance and collaboration throughout this project. I **believe OpenHW-Studio** has strong potential as a classroom-ready embedded

systems learning tool, and I am glad to have contributed to its testing, validation, documentation, and gamification design.

IEEE Reference

You can directly place these in the References section of your report.

[1] Bosch Sensortec, BMP180 Digital Pressure Sensor Datasheet, Rev. 2.8, Bosch Sensortec GmbH, 2013.

[2] Maxim Integrated, DS1307 64 × 8 Serial Real-Time Clock Datasheet, Maxim Integrated Products Inc.

[3] InvenSense, MPU-6000 and MPU-6050 Product Specification, Revision 3.4.

[4] Texas Instruments, LM393 Dual Differential Comparator Datasheet, Texas Instruments Inc.

[5] Songle Relay, SRD-05VDC-SL-C Relay Datasheet.

[6] Analog Devices, ADXL345 Three-Axis Digital Accelerometer Datasheet, Analog Devices Inc.

[7] Maxim Integrated, DS18B20 Programmable Resolution 1-Wire Digital Thermometer Datasheet.

[8] Vishay Semiconductors, TSOP38238 IR Receiver Modules for Remote Control Systems Datasheet.

[9] NXP Semiconductors, MFRC522 Standard Performance MIFARE and NTAG Frontend Datasheet, Rev. 3.9.

[10] Adafruit Industries, BMP180 Barometric Pressure Sensor Guide.

[11] Adafruit Industries, ADXL345 Triple-Axis Accelerometer Guide.

[12] Miguel Balboa, MFRC522 Arduino Library Documentation, GitHub