Summer Internship Report

on

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

Submitted by

Manas Krishna Neigapula

B.Tech in Electronics and Communication Engineering,

VIT BHOPAL UNIVERSITY, Bhopal

Under the Guidance of

Prof. Prabhu Ramachandran

Principal Investigator, FOSSEE Project

Department of Aerospace Engineering,

Indian Institute of Technology Bombay

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.

Manas Krishna Neigapula
VIT BHOPAL UNIVERSITY

Declaration

This internship has been an enriching learning experience, allowing me to contribute to the development of simulated hardware components and implement enhancements and bug fixes in **OpenHW Studio**. It provided me with valuable exposure to open-source software development, embedded systems simulation, and collaborative engineering workflows. The knowledge and practical experience gained during this internship will undoubtedly support my future academic and professional endeavors.

I would also like to thank the entire **FOSSEE** team for their coordination, guidance, and continuous support throughout the internship. Their collective efforts, timely assistance, and encouragement created an excellent learning environment and played a significant role in the successful completion of my assigned tasks.

.

Manas Krishna Neigapula
VIT BHOPAL UNIVERSITY

Index

Chapter 1

Introduction

OpenHW Studio is an open-source, web-based hardware simulation platform developed under the FOSSEE (Free and Open Source Software for Education) project at IIT Bombay. It allows students, educators, hobbyists, and researchers to design, wire, and simulate Arduino-based electronic circuits entirely inside a browser — without needing physical hardware. By providing a virtual canvas of microcontrollers, sensors, displays, and actuators, OpenHW Studio lowers the entry barrier to embedded systems education and accelerates rapid prototyping.

The platform offers a drag-and-drop component library, live wiring, real-time serial monitoring, and instant code execution. It is particularly well suited for STEM classrooms, online laboratories, and remote learning environments where access to physical Arduino kits may be limited. Components in OpenHW Studio are modeled with realistic pin-outs, electrical behavior, and configurable runtime attributes, enabling users to build and test projects ranging from simple LED blinks to advanced sensor-driven robotics systems.

By combining the flexibility of the Arduino ecosystem with a fully simulated hardware environment, OpenHW Studio fosters creativity, makes electronics prototyping more approachable, and supports India's wider push toward open, accessible, and high-quality engineering education.

1.1 Project Overview

As part of my summer internship with the FOSSEE OpenHW Studio team, I worked as a member of the Emulator Team. My primary responsibility involved designing and integrating new simulated hardware components into the platform, ensuring that each component exposed accurate pin definitions, configurable runtime attributes, realistic electrical behavior, and clean integration with the Arduino code-execution engine.

In addition to building new components, I worked on debugging and fixing issues with existing components — particularly around telemetry, the live-data HUD, and serial output reliability. The goal was to make the simulator a robust, classroom-ready tool whose components behaved

consistently with their physical counterparts and whose runtime data could be trusted by students learning embedded programming.

1.2 Role and Responsibilities

Across the course of the internship I contributed to nine simulated components and a set of supporting fixes. My responsibilities can be grouped into four broad streams:

- Designing component manifests (type, dimensions, attributes, pin layout) for new sensors and modules.
- Implementing the runtime behavior of each component so that its outputs respond correctly to user-configurable attributes (e.g. temperature, pressure, acceleration, time).
- Authoring component documentation pages — pin reference tables, attribute tables, wiring diagrams, and end-to-end Arduino example sketches.
- Debugging existing components and fixing issues with telemetry data flow and Serial Monitor output.

Chapter 2

Component Development

This chapter describes each of the nine components I contributed to the OpenHW Studio emulator. For every component, I detail the physical device it models, the pin layout exposed in the simulator, the configurable runtime attributes, and the way the component integrates with Arduino sketches inside the platform.

2.1 BMP180 Pressure Sensor Breakout

The BMP180 Pressure and Temperature Sensor component is a virtual representation of the Bosch BMP180 digital barometric pressure sensor used within the circuit simulation environment. It is designed to accurately emulate the behavior of the physical sensor, enabling users to develop, test, and validate embedded applications without requiring actual hardware. The component communicates using the Inter-Integrated Circuit (I^2C) protocol and provides real-time atmospheric pressure and temperature measurements. In addition to these primary measurements, the simulator also computes altitude using the standard atmospheric pressure equation, making the component suitable for weather monitoring, environmental sensing, navigation, and altitude estimation applications.

Component Design

The BMP180 component has been developed using a modular software architecture consisting of five independent modules: component registration, simulation logic, graphical user interface, validation module, and component manifest. This modular approach improves maintainability, readability, scalability, and allows future enhancements without affecting other parts of the simulator.

The **component registration module** integrates all required files and registers the BMP180 component with the simulation engine. It links the simulation logic, user interface, context menu, manifest, and validation rules into a single executable component. This structure enables the simulator to dynamically instantiate the sensor whenever it is added to the workspace.

The **manifest module** defines the physical characteristics of the BMP180 breakout board, including its dimensions, default environmental parameters, and electrical pin configuration. Four

connection pins are exposed to the simulator: VIN for power input, GND for ground reference, SCL for the I²C clock line, and SDA for the I²C data line. Default operating conditions are initialized with a pressure of 1013.25 hPa and a temperature of 25°C, providing realistic startup values for most simulations.

Simulation Logic

The core functionality of the BMP180 component is implemented through a dedicated simulation logic module that accurately models the behavior of the physical Bosch BMP180 sensor. The simulator maintains internal variables representing atmospheric pressure, ambient temperature, altitude, power status, register pointers, and measurement commands. These variables are continuously updated during simulation to ensure realistic sensor behavior.

Whenever the simulation is running, the component monitors changes in pressure and temperature and synchronizes the updated values with the simulator state. The calculated altitude is automatically refreshed whenever atmospheric pressure changes, ensuring that all sensor outputs remain consistent with real-world physical relationships.

Power Management

To simulate real hardware operation, the BMP180 continuously monitors its supply voltage. The component becomes operational only when the input voltage at the VIN pin exceeds the minimum operating voltage of approximately 1.8 V. If sufficient power is unavailable, the sensor remains inactive, accurately representing the operating characteristics of the actual device. This feature enables users to identify power-related circuit issues during simulation before hardware implementation.

I²C Communication Protocol

Communication between the microcontroller and the BMP180 sensor is performed using the I²C serial communication protocol. The simulated sensor operates at the fixed 7-bit device address **0x77**, identical to the actual BMP180 hardware.

The component fully supports the standard I²C communication sequence, including:

- Start condition detection
- Address recognition
- Register selection
- Data transmission
- Data reception
- Stop condition detection

By implementing these communication stages, the simulator remains compatible with commonly used Arduino libraries such as the Adafruit BMP085/BMP180 library, allowing existing embedded applications to execute without modification.

Register-Level Sensor Emulation

To achieve realistic behavior, the component simulates the internal register structure of the BMP180. Several important registers are implemented, including the chip identification register, measurement control register, pressure registers, temperature registers, and factory calibration registers.

The simulated calibration coefficients are carefully selected to simplify the mathematical compensation process while maintaining compatibility with standard BMP180 driver libraries. This enables Arduino applications to retrieve meaningful pressure and temperature values exactly as they would from a physical sensor.

Measurement Generation

Instead of storing static output values, the simulator dynamically generates uncompensated temperature (UT) and uncompensated pressure (UP) values. These raw measurements are calculated from the user-defined environmental parameters and returned through the simulated I^2C interface.

This approach closely mimics the operation of the physical BMP180, where the sensor internally measures raw values and the software library performs compensation using calibration coefficients. Consequently, the simulator accurately reproduces the complete software interaction between the microcontroller and the sensor.

Altitude Calculation

In addition to pressure and temperature sensing, the component automatically estimates altitude using the internationally accepted barometric pressure equation. The calculated altitude is continuously updated whenever atmospheric pressure changes, allowing users to simulate different elevations and environmental conditions.

This functionality is particularly useful for applications involving weather stations, drone navigation, environmental monitoring systems, and portable altimeters.

Graphical User Interface

The BMP180 component includes a detailed graphical representation of a commercial breakout board. The user interface has been designed using scalable vector graphics (SVG) and includes visual elements such as the printed circuit board, mounting holes, voltage regulator, passive electronic components, pressure sensor package, copper traces, gold-plated pin headers, and silkscreen labels.

The realistic appearance improves user experience by closely resembling commercially available BMP180 modules commonly used in embedded system projects.

Interactive User Controls

The component provides an interactive context menu that enables users to modify the simulated environmental conditions during runtime. Users can manually adjust atmospheric pressure and ambient temperature through numeric input fields, allowing the behavior of embedded programs to be tested under different operating conditions without altering the program code.

Changes made through the interface are immediately reflected in the sensor outputs, making the simulator highly interactive and suitable for educational as well as development purposes.

Validation Mechanism

To improve circuit reliability, the BMP180 component incorporates an automated validation system that checks the correctness of electrical connections before simulation begins. The validation module verifies that:

- VIN is connected to a valid power supply.
- GND is connected properly.
- SDA is connected to the correct I²C data pin.
- SCL is connected to the correct I²C clock pin.
- Appropriate Arduino I²C pin mapping is followed.

Whenever an error is detected, descriptive warning messages are displayed, enabling users to identify and correct wiring mistakes before executing the simulation. This significantly reduces debugging time and enhances the educational value of the simulator.

Features of the BMP180 Component

The implemented BMP180 component provides the following features:

- Realistic simulation of the Bosch BMP180 digital pressure sensor.
- Complete implementation of the I²C communication protocol.
- Register-level emulation compatible with standard Arduino libraries.
- Dynamic simulation of atmospheric pressure and ambient temperature.
- Automatic altitude estimation using the barometric pressure equation.
- Real-time synchronization of environmental parameters.
- Interactive graphical interface representing an actual breakout board.
- User-configurable pressure and temperature values through the context menu.
- Automatic power-state monitoring.
- Built-in circuit validation and wiring verification.
- Modular software architecture for easy maintenance and future expansion.

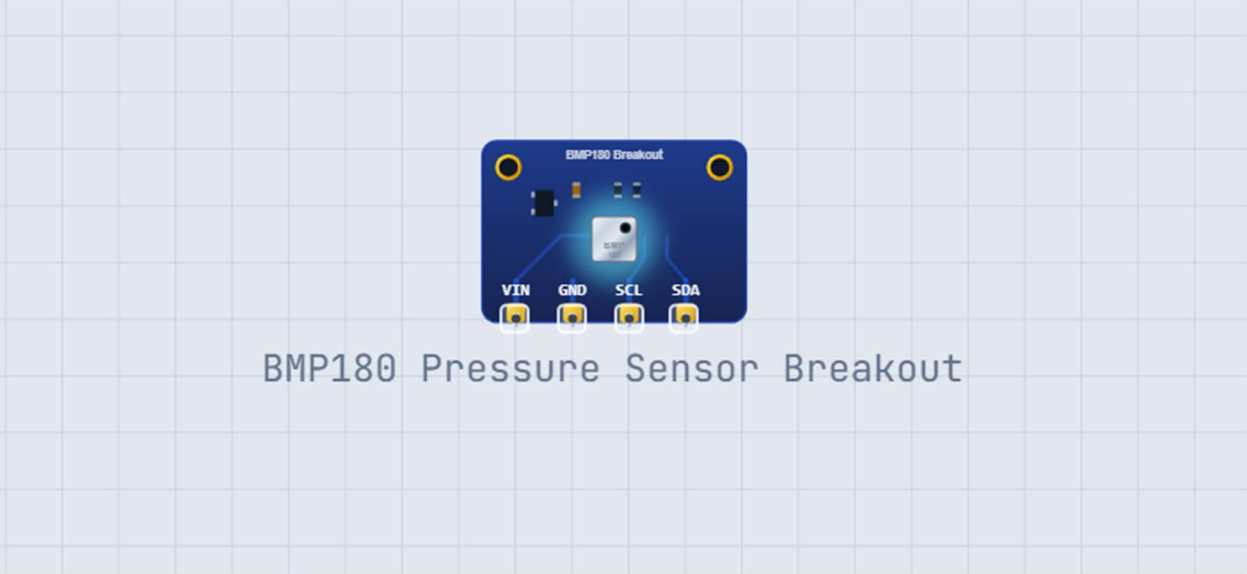

Fig 2.1

2.2 DS1307 Real-Time Clock Module

The DS1307 is an industry-standard I2C real-time clock chip (address 0x68) that keeps accurate time using a 32.768 kHz crystal and a CR2032 backup battery. I modeled a Tiny RTC breakout board variant of the DS1307 with dual-side pin headers (matching the physical board), enabling realistic wiring exercises in OpenHW Studio The DS1307 Real-Time Clock (RTC) module is a digital timekeeping component designed to maintain accurate date and time information independently of the main microcontroller. Unlike software-based timers that stop when power is removed, the DS1307 continues to operate using a dedicated backup battery, allowing it to preserve time information even during system shutdowns or power failures. In the simulation environment, the DS1307 component accurately reproduces the functionality of the physical RTC module by implementing real-time clock progression, I²C communication, register-level operations, battery-backed timekeeping, and an interactive graphical interface. This enables users to develop and test time-dependent embedded applications without requiring physical hardware.

Component Architecture

The DS1307 RTC component is implemented using a modular architecture consisting of five independent software modules: component registration, simulation logic, graphical user

interface, validation module, and manifest configuration. This architecture improves software maintainability by separating the functional behavior, user interface, hardware configuration, and validation mechanisms into dedicated files.

The component registration module integrates all necessary modules, including the simulation logic, user interface, context menu, validation rules, component manifest, and user documentation. This centralized registration enables the simulator to dynamically load and instantiate the DS1307 RTC whenever it is selected by the user.

Component Configuration

The manifest file defines the physical characteristics and hardware configuration of the DS1307 RTC module. It specifies the component dimensions, default simulation attributes, and available input/output pins. The module initializes with a default date and time of **1 January 2024, 00:00:00**, which serves as the starting point for the simulation clock.

The simulated RTC module exposes multiple connection pins including VCC, GND, SDA, SCL, battery output (BAT), square-wave output (SQ), and DS18B20 interface pins. These pin definitions accurately reflect commercially available Tiny RTC modules and allow realistic circuit construction within the simulator.

Simulation Logic

The core functionality of the DS1307 component is implemented through the **DS1307RTCLogic** class, which emulates the internal operation of the physical RTC integrated circuit. The simulator maintains internal variables such as the base date, simulation start time, power status, register pointer, and current display state.

When the simulation begins, the configured date and time are stored as the base reference. As the simulation executes, elapsed system time is continuously added to this reference, allowing the simulated clock to advance in real time. The updated date and time are automatically synchronized with the simulator state, ensuring that all displayed and transmitted values accurately represent the current simulation time.

Real-Time Clock Operation

The primary function of the DS1307 is to provide continuous timekeeping. The simulator accurately reproduces this behavior by maintaining an internal software clock that advances

automatically during execution. The displayed time includes the current year, month, day, hour, minute, and second.

Unlike conventional timers, the RTC continues to maintain a consistent timeline regardless of application execution, closely matching the behavior of the physical DS1307 integrated circuit that uses a 32.768 kHz crystal oscillator and backup battery for continuous operation. This implementation enables realistic testing of scheduling, timestamping, data logging, and clock-based automation applications.

Power Management

To simulate actual hardware behavior, the DS1307 continuously monitors its supply voltage. The RTC becomes operational only when the supply voltage exceeds approximately **2.5 V**. If the required voltage is not available, the component is marked as unpowered and its display is updated accordingly.

This feature allows users to verify proper power connections during circuit design and provides realistic feedback when the RTC is incorrectly powered or disconnected.

I²C Communication

The DS1307 communicates with the microcontroller using the **Inter-Integrated Circuit (I²C)** communication protocol. The simulator fully implements this communication mechanism using the standard 7-bit device address **0x68**, identical to the physical RTC module.

The communication process includes:

- Start condition detection
- Slave address recognition
- Register pointer selection
- Sequential register reading
- Register writing
- Stop condition handling

The implementation ensures compatibility with popular Arduino libraries such as **RTClib**, allowing existing embedded applications to interact with the simulated RTC without modification.

Register-Level Emulation

To accurately reproduce the internal behavior of the DS1307, the simulator implements the device's register map. The component stores and returns time information through dedicated registers corresponding to seconds, minutes, hours, day of the week, date, month, year, and control settings.

All time values are represented using **Binary-Coded Decimal (BCD)** format, matching the data storage method of the actual DS1307 integrated circuit. During each I²C read operation, the simulator converts the current date and time into BCD format before transmitting the register values to the microcontroller. This enables standard RTC libraries to interpret the data exactly as they would when communicating with physical hardware.

User Interface

The graphical representation of the DS1307 module has been designed to resemble a commercially available Tiny RTC breakout board. The interface includes a realistic printed circuit board (PCB), crystal oscillator, DS1307 integrated circuit, EEPROM chip, battery connector, electronic components, mounting holes, and clearly labelled pin headers.

A digital display positioned on the module dynamically presents the current simulated date and time. When the module is powered, the display updates in real time, providing immediate visual feedback to the user throughout the simulation.

Interactive Configuration

The component provides an interactive context menu through which users can configure the initial date and time before or during simulation. The interface utilizes a date-time selector, allowing users to specify the starting timestamp without modifying the program source code.

Once configured, the selected date and time become the base reference for the internal simulation clock. This functionality enables convenient testing of applications involving alarms, event scheduling, data logging, and calendar-based operations.

Validation Mechanism

The DS1307 RTC module incorporates an automated validation system to verify correct circuit connections before simulation begins. The validation module performs several hardware integrity checks, including verification of the power supply, ground connection, SDA line, and SCL line.

If any required connection is missing, descriptive warning messages are generated to guide the user toward correcting the wiring configuration. These validation routines significantly improve usability by preventing common connection errors and reducing debugging time during circuit development.

Features of the DS1307 RTC Component

The developed DS1307 RTC component provides the following features:

- Accurate real-time clock simulation with continuous time progression.
- Complete implementation of the I²C communication protocol using device address **0x68**.
- Register-level emulation compatible with standard Arduino RTC libraries.
- Binary-Coded Decimal (BCD) representation of date and time registers.
- Automatic synchronization of simulation time with the graphical interface.
- Interactive configuration of the initial date and time through the context menu.
- Realistic graphical representation of the Tiny RTC breakout board.
- Dynamic digital display showing the current simulated date and time.
- Automatic power-state detection based on supply voltage.
- Built-in validation rules for power and I²C wiring verification.
- Modular software architecture that separates logic, interface, configuration, and validation for improved maintainability and future extensibility.

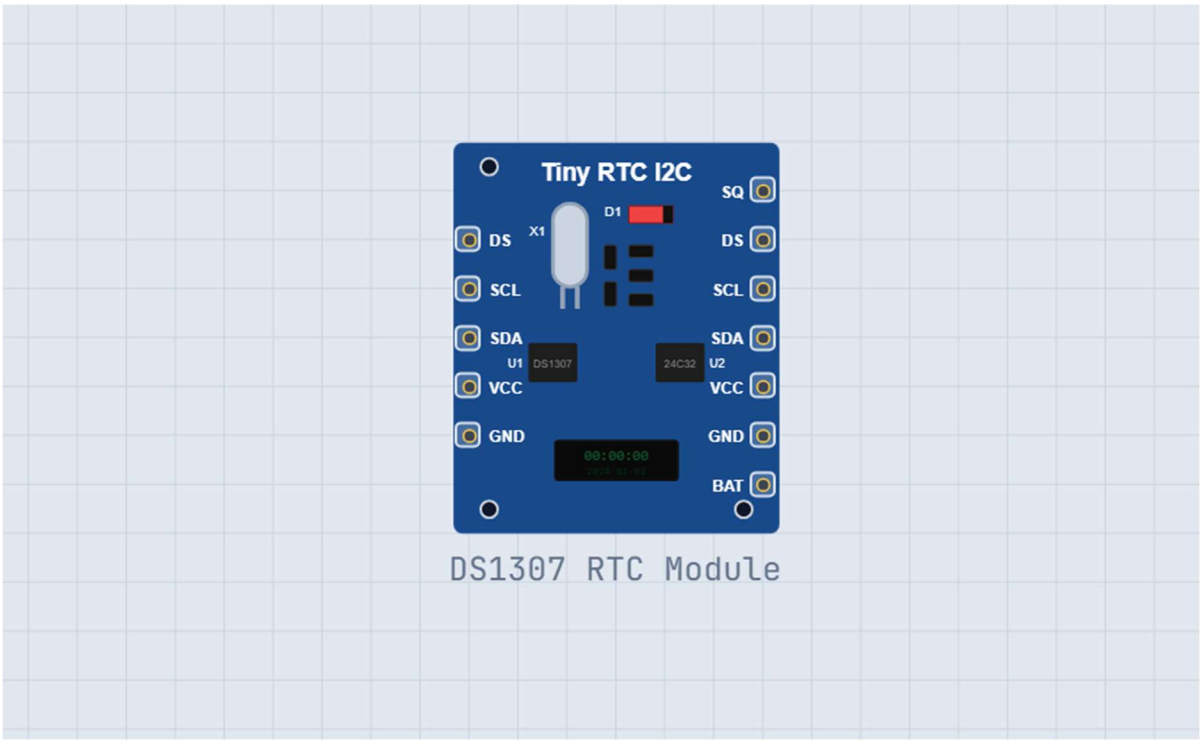

Fig 2.2

2.3 MPU6050 IMU Sensor

The MPU6050 is a six Degrees of Freedom (6-DOF) Inertial Measurement Unit (IMU) that integrates a three-axis accelerometer and a three-axis gyroscope within a single integrated circuit. It is widely used in embedded systems for measuring linear acceleration, angular velocity, orientation, motion, and vibration. The sensor communicates with the microcontroller through the Inter-Integrated Circuit (I^2C) communication protocol, enabling efficient data transfer with minimal wiring requirements. In the simulation environment, the MPU6050 component accurately emulates the behavior of the physical sensor by supporting register-level communication, configurable motion parameters, real-time sensor data generation, and an interactive graphical interface. This implementation allows users to develop, test, and validate motion sensing applications without requiring actual hardware.

Component Architecture

The MPU6050 component is implemented using a modular architecture consisting of component registration, simulation logic, graphical user interface, manifest configuration, validation rules, and integrated documentation. This modular design separates the functionality of

each module, making the component easier to maintain, debug, and extend. The registration module connects all component resources, including the simulation logic, user interface, context menu, validation functions, and documentation, allowing the simulator to dynamically instantiate and execute the component during runtime.

Component Configuration

The manifest file defines the physical characteristics and hardware interface of the MPU6050 module. It specifies the board dimensions, default sensor parameters, and available connection pins. The simulator initializes the sensor with zero acceleration on the X and Y axes, 1 g acceleration along the Z-axis to represent gravitational force, zero angular velocity on all three rotational axes, and a default operating temperature of 25°C.

The module exposes eight connection pins including VCC, GND, SDA, SCL, XDA, XCL, ADO, and INT. While SDA and SCL provide the primary I²C communication interface, the XDA and XCL pins are available for connecting auxiliary I²C devices such as external magnetometers. The ADO pin allows selection of the I²C device address, while the INT pin provides interrupt signaling for motion detection and data-ready events.

Simulation Logic

The simulation logic is implemented through the **MPU6050Logic** class, which accurately models the behavior of the physical MPU6050 sensor. The component maintains internal variables representing acceleration along the X, Y, and Z axes, angular velocity along the three rotational axes, operating temperature, power status, register pointer, sleep mode, and device configuration registers.

During simulation, these parameters are continuously updated and synchronized with the simulator state. Whenever the user modifies acceleration, gyroscope readings, or temperature through the simulator interface, the internal sensor state is immediately updated and made available through the I²C communication interface. This enables real-time testing of motion sensing algorithms and embedded applications under varying environmental conditions.

Power Management

The MPU6050 simulation incorporates realistic power management by continuously monitoring the supply voltage connected to the VCC pin. The component becomes operational

only when the applied voltage exceeds approximately **2.375 V**, closely matching the operating requirements of the physical device.

The simulator also reproduces the sensor's default startup behavior, where the MPU6050 powers up in **sleep mode**. To begin normal operation, the microcontroller must write an appropriate value to the **PWR_MGMT_1** register, thereby waking the device from sleep. This behavior closely follows the initialization procedure required by actual MPU6050 hardware and ensures compatibility with existing embedded software libraries.

I²C Communication

Communication between the microcontroller and the MPU6050 is performed using the I²C serial communication protocol. The simulator fully implements this protocol, including support for start and stop conditions, register addressing, sequential data transfer, and automatic register pointer incrementing.

The component supports two possible I²C addresses:

- **0x68** (default when ADO is LOW)
- **0x69** (when ADO is HIGH)

The simulator automatically determines the active device address based on the voltage applied to the ADO pin, accurately reproducing the hardware address selection mechanism of the physical sensor. This implementation ensures compatibility with standard Arduino libraries such as **Adafruit_MPU6050** and other MPU6050 driver libraries.

Register-Level Emulation

To achieve realistic hardware behavior, the component implements a register-level simulation of the MPU6050. Configuration registers, sensor output registers, power management registers, interrupt configuration registers, and device identification registers are all supported.

The simulator emulates important registers such as:

- Accelerometer output registers
- Gyroscope output registers
- Temperature output registers

- Power Management Register (PWR_MGMT_1)
- Accelerometer and Gyroscope configuration registers
- WHO_AM_I identification register

The **WHO_AM_I** register consistently returns the value **0x68**, allowing software drivers to correctly identify the simulated device. Additionally, configuration registers preserve values written by the microcontroller, enabling realistic interaction between embedded applications and the virtual sensor.

Sensor Data Generation

Instead of returning constant values, the simulator dynamically generates raw accelerometer, gyroscope, and temperature measurements based on the user-defined sensor parameters. The acceleration values are converted into raw digital counts using the ±2 g sensitivity scale, while gyroscope readings are generated using the ±250°/s sensitivity range. Similarly, temperature values are converted according to the conversion formula used by the actual MPU6050 sensor.

These generated raw values are stored within the corresponding output registers, allowing embedded software to retrieve and process them exactly as it would with physical hardware. This approach enables accurate testing of sensor fusion algorithms, orientation estimation, balancing systems, and motion-controlled applications.

Interactive User Interface

The MPU6050 component includes a detailed graphical representation of a commercial breakout board. The SVG-based interface depicts the printed circuit board, mounting holes, integrated sensor package, voltage regulator, passive electronic components, labelled connection pins, axis orientation indicators, and a digital information display.

The graphical interface also includes a live sensor status display showing acceleration, angular velocity, and temperature values in real time. These values are automatically updated during simulation, providing immediate visual feedback regarding the current operating state of the sensor.

Interactive Configuration

A dedicated context menu allows users to modify simulated sensor values during runtime. Through this interface, users can independently adjust acceleration along the X, Y, and Z axes, angular velocity around each rotational axis, and operating temperature.

Changes made through the context menu are immediately reflected within both the graphical interface and the internal sensor registers, enabling realistic testing of motion detection algorithms, robotic control systems, balancing applications, gesture recognition, and inertial navigation systems without modifying the embedded software.

Validation Mechanism

The MPU6050 component incorporates an automated validation system to verify correct circuit configuration before simulation begins. The validation module checks whether the power supply, ground, SDA, and SCL connections have been established correctly.

Additionally, the simulator reminds users that the MPU6050 initially starts in sleep mode and must be awakened by writing **0x00** to the **PWR_MGMT_1 (0x6B)** register during system initialization. This validation mechanism improves usability by preventing common configuration errors and ensuring proper communication between the microcontroller and the sensor.

Features of the MPU6050 Component

The implemented MPU6050 component provides the following features:

- Accurate simulation of a 6-DOF Inertial Measurement Unit combining a three-axis accelerometer and three-axis gyroscope.
- Complete implementation of the I²C communication protocol with support for both **0x68** and **0x69** device addresses.
- Register-level emulation compatible with standard Arduino MPU6050 libraries.
- Real-time generation of acceleration, angular velocity, and temperature measurements.
- Support for configurable accelerometer, gyroscope, and temperature values through an interactive context menu.
- Automatic power-state detection and realistic startup behavior with sleep mode support.

- Persistent configuration registers and accurate device identification through the **WHO_AM_I** register.
- Interactive graphical interface with a live sensor data display.
- Built-in validation rules for power supply, I²C communication, and device initialization.
- Modular software architecture that separates simulation logic, graphical interface, configuration, validation, and documentation, improving maintainability and facilitating future enhancements.

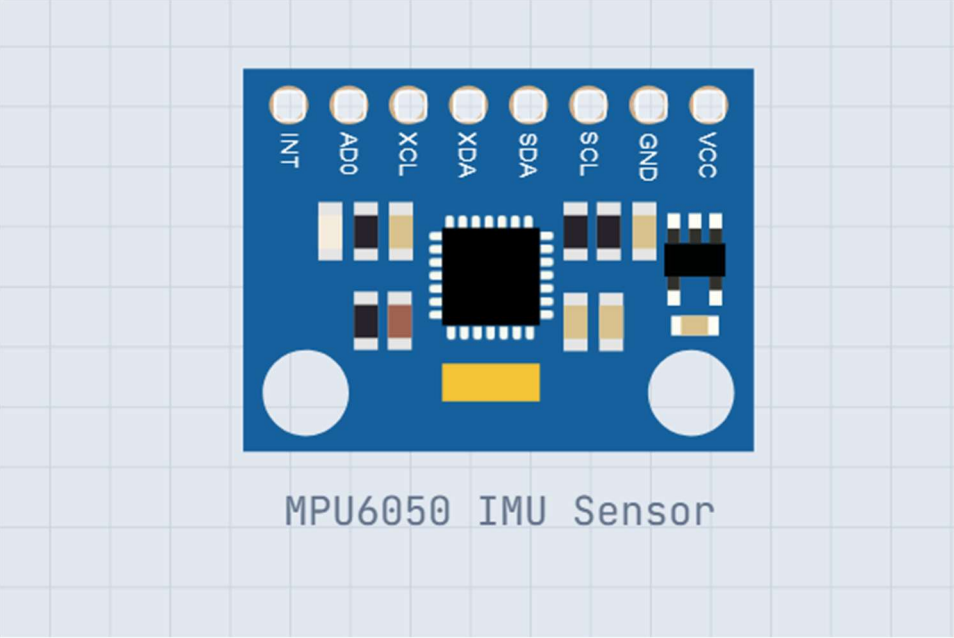

Fig 2.3

2.4 NTC Thermistor Module

The NTC Thermistor Module is a low-cost analog/digital temperature sensor commonly used in entry-level Arduino projects. It pairs an NTC thermistor with an LM393 comparator to produce both a raw analog voltage and a thresholded digital trigger. I implemented this module so that students can practice both analogRead() and digitalRead() workflows on the same component. The NTC Thermistor Temperature Sensor Module is a temperature sensing component that combines a Negative Temperature Coefficient (NTC) thermistor with an LM393 voltage comparator to provide both analog and digital temperature outputs. The resistance of the NTC thermistor decreases as temperature increases, producing a variable voltage through a voltage divider circuit. This voltage is processed by the LM393 comparator to generate a digital output

whenever the measured temperature crosses a user-defined threshold. In the simulation environment, the component accurately reproduces the behavior of the physical module by implementing thermistor resistance calculations, voltage divider operation, comparator logic, configurable threshold adjustment, and an interactive graphical interface. This enables users to design, test, and validate temperature monitoring and control applications without requiring physical hardware.

Component Architecture

The NTC Thermistor Module is implemented using a modular software architecture consisting of component registration, simulation logic, graphical user interface, manifest configuration, and validation modules. Each module performs a dedicated function, improving software maintainability and allowing future enhancements without affecting other parts of the simulator.

The component registration module integrates the simulation logic, graphical interface, context menu, validation rules, and manifest configuration into a single component that can be dynamically loaded by the simulation engine. This modular structure simplifies debugging, code maintenance, and future feature development.

Component Configuration

The manifest file defines the physical properties and electrical interface of the NTC Thermistor Module. The module is initialized with a default temperature of **25°C** and a comparator threshold value of **512**, representing the midpoint of the 10-bit analog range (0–1023).

The component exposes four output pins:

- **A0** – Analog voltage output from the thermistor voltage divider.
- **D0** – Digital comparator output.
- **GND** – Ground connection.
- **VCC** – Power supply input (3.3 V–5 V).

These pin definitions accurately represent commercially available LM393-based NTC temperature sensor modules commonly used in Arduino-based applications.

Simulation Logic

The core functionality of the NTC Thermistor Module is implemented through the **NTCThermistorLogic** class, which emulates the electrical characteristics of an NTC thermistor and the behavior of the LM393 comparator.

The simulation maintains internal variables including:

- Ambient temperature
- Nominal thermistor resistance
- Nominal reference temperature
- Beta coefficient
- Comparator threshold
- Power status
- Digital output state

Whenever the simulated temperature or threshold value changes, the component recalculates the thermistor resistance, updates the analog voltage, determines the digital comparator output, and synchronizes the updated values with the simulator state. This enables realistic testing of temperature-dependent embedded applications under varying environmental conditions.

Temperature Measurement

The module measures temperature using a Negative Temperature Coefficient (NTC) thermistor whose resistance decreases as temperature increases. The simulator calculates the thermistor resistance using the **Beta Parameter Equation**, which models the nonlinear relationship between resistance and temperature.

The calculated resistance is then used within a voltage divider consisting of a fixed 10 kΩ resistor and the NTC thermistor. As the surrounding temperature changes, the output voltage at the analog pin changes proportionally, allowing the microcontroller to determine the corresponding temperature through analog-to-digital conversion. This implementation closely reproduces the behavior of a physical NTC temperature sensing circuit.

Voltage Divider and Comparator Operation

The simulated module generates an analog output voltage through a voltage divider circuit formed by the fixed resistor and the thermistor. The analog voltage appearing at the A0 pin is continuously updated according to the current thermistor resistance and supply voltage.

The LM393 comparator compares this analog voltage against a threshold voltage determined by the adjustable potentiometer. If the measured voltage falls below the threshold, indicating that the temperature has exceeded the configured limit, the comparator drives the digital output (D0) LOW. Otherwise, the digital output remains HIGH. This behavior accurately reproduces the operation of commercially available NTC comparator modules used in temperature alarm and automatic control applications.

Power Management

The component continuously monitors the voltage applied to the VCC pin. The module becomes operational only when the supply voltage exceeds approximately **2.0 V**. When adequate power is available, the analog and digital outputs are generated according to the measured temperature and comparator threshold. If the module is not powered, both outputs are disabled and the component enters an inactive state.

This realistic power management enables users to identify wiring and power supply issues during circuit development before deploying the design to physical hardware.

Interactive User Interface

The graphical user interface provides a realistic representation of an LM393-based NTC Thermistor Module. The SVG-based design includes the printed circuit board, NTC thermistor, LM393 comparator integrated circuit, adjustable potentiometer, indicator LEDs, mounting hole, pin headers, PCB traces, and connection labels.

The interface also displays the operational state of the module through two status LEDs. The power LED indicates whether the module is receiving sufficient supply voltage, while the digital output LED reflects the current comparator output, allowing users to observe the sensor response during simulation.

Interactive Configuration

The component includes an interactive context menu through which users can modify the simulated ambient temperature and comparator threshold during runtime. Temperature values can be adjusted over a practical operating range, while the comparator threshold can be configured using values between **0 and 1023**, corresponding to the full-scale range of a 10-bit analog-to-digital converter.

The simulator also allows users to interact with the graphical potentiometer. Rotating the potentiometer modifies the comparator threshold in discrete steps, providing a realistic representation of threshold adjustment on the physical module. Any modifications are immediately reflected in both the analog output voltage and digital comparator state.

Validation Mechanism

The NTC Thermistor Module incorporates an automated validation system that verifies proper circuit configuration before simulation begins. The validation module checks that the sensor is correctly connected, that one terminal is linked to an analog input, and that a fixed resistor is included to form the required voltage divider circuit.

If any of these conditions are not satisfied, informative warning messages are displayed to assist users in correcting the circuit. This validation process minimizes common wiring mistakes and improves the reliability of simulated temperature measurements.

Features of the NTC Thermistor Module

The implemented NTC Thermistor Module provides the following features:

- Accurate simulation of an LM393-based NTC temperature sensing module.
- Realistic thermistor resistance calculation using the Beta parameter equation.
- Dynamic analog voltage generation through a simulated voltage divider circuit.
- Digital comparator output with adjustable temperature threshold.
- Interactive adjustment of ambient temperature and comparator threshold through the context menu.
- Graphical potentiometer simulation for threshold configuration.

- Real-time power monitoring with visual status indicators.
- Detailed graphical representation of the physical sensor module.
- Built-in validation for circuit connections, analog interface, and voltage divider configuration.
- Modular software architecture separating simulation logic, user interface, configuration, and validation, ensuring scalability and ease of maintenance.

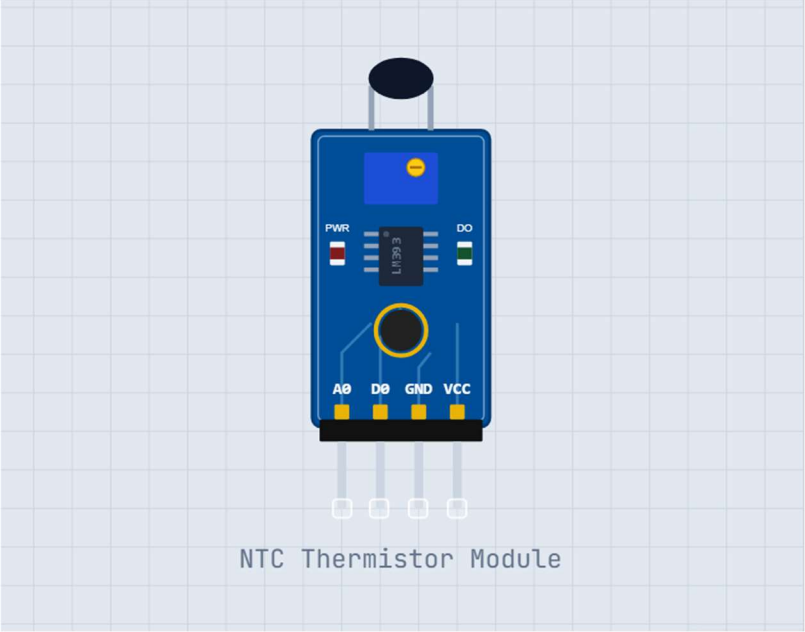

Fig 2.4

2.5 Relay Module

The Relay Module is an electromechanical switching device designed to enable low-voltage microcontrollers to safely control high-voltage or high-current electrical loads. It provides electrical isolation between the control circuit and the load circuit, making it suitable for applications such as home automation, industrial control, motor switching, lighting systems, and power management. The simulated Relay Module accurately reproduces the behavior of a single-channel 5 V relay by implementing trigger-based switching, configurable trigger modes, relay state management, power monitoring, and a realistic graphical representation. The module supports both Active LOW and Active HIGH triggering, allowing users to simulate different commercially available relay modules within the circuit simulator.

Component Architecture

The Relay Module has been implemented using a modular software architecture consisting of component registration, simulation logic, graphical user interface, validation module, manifest configuration, and integrated documentation. Each module performs an independent task, improving maintainability, scalability, and ease of future enhancements.

The component registration module combines the simulation logic, graphical interface, context menu, validation rules, and manifest into a single component that can be dynamically loaded by the simulator. This modular structure ensures that modifications to one part of the component do not affect the remaining implementation.

Component Configuration

The manifest file defines the physical dimensions, default attributes, and electrical interface of the Relay Module. By default, the relay is initialized in the **OFF** state and configured to operate using **Active LOW** triggering, which is the most common configuration found in commercial Arduino-compatible relay modules.

The module provides six external connection terminals:

- **NO (Normally Open)** – Connected to COM only when the relay is energized.
- **COM (Common)** – Common switching terminal connected to the external load.
- **NC (Normally Closed)** – Connected to COM while the relay remains de-energized.
- **IN** – Digital control input from the microcontroller.
- **GND** – Ground connection.
- **VCC** – 5 V supply required to energize the relay coil.

These connections accurately represent the electrical interface of a standard single-channel relay module used in embedded system applications.

Simulation Logic

The operational behavior of the Relay Module is implemented through the **RelayModuleLogic** class, which simulates the electromechanical switching characteristics of a physical relay. The simulation maintains internal variables representing the relay energization state and configurable trigger level.

During simulation, the component continuously monitors both the supply voltage and the input control signal. Based on the configured trigger mode, the simulator determines whether the relay coil should be energized or de-energized. The internal state is then synchronized with the graphical interface, allowing users to observe relay operation in real time.

The simulation accurately models relay switching behavior by responding dynamically to changes in the control signal while ensuring that relay operation occurs only when sufficient supply voltage is available.

Relay Switching Operation

The primary function of the Relay Module is to electrically connect or disconnect external loads through its switching contacts. When the relay coil is energized, the internal movable contact switches from the **Normally Closed (NC)** terminal to the **Normally Open (NO)** terminal, thereby connecting the **Common (COM)** terminal to **NO**. When the relay is de-energized, the contact automatically returns to its default position, reconnecting COM to NC.

The simulator reproduces this switching mechanism by maintaining an internal energization state and updating the relay status whenever the input control signal changes. This enables realistic simulation of switching applications such as motor control, lighting automation, and appliance control.

Trigger Configuration

Unlike fixed relay simulations, this component supports both **Active LOW** and **Active HIGH** trigger modes. In the default Active LOW configuration, the relay is energized whenever the control input voltage falls below approximately **1 V**. In Active HIGH mode, the relay is energized whenever the input voltage exceeds approximately **2.5 V**.

The trigger mode can be modified during simulation using the context menu, allowing users to evaluate embedded programs with different relay configurations without altering the program source code. This flexibility improves compatibility with a wide range of commercially available relay modules.

Power Management

To accurately emulate physical hardware, the Relay Module continuously monitors the voltage applied to the VCC terminal. The relay operates only when the supply voltage exceeds approximately **2.5 V**. If the supply voltage falls below this threshold, the relay is automatically de-energized and the switching contacts return to their default position.

This behavior reflects the operating characteristics of actual relay modules and enables users to identify incorrect power connections during circuit development before implementing the design on physical hardware.

Graphical User Interface

The Relay Module includes a detailed graphical representation closely resembling a commercial single-channel relay board. The interface displays the printed circuit board, screw terminal block, relay package, pin headers, indicator LEDs, and clearly labelled input and output terminals.

Two visual indicators are incorporated into the interface. A red LED represents the power status of the module, while a green LED indicates whether the relay coil is currently energized. These indicators update dynamically throughout the simulation, providing immediate visual feedback regarding the operating condition of the relay.

Interactive Configuration

The simulator provides an interactive context menu that allows users to modify the relay trigger mode during execution. Users can switch between **Active LOW** and **Active HIGH** configurations without restarting the simulation or modifying the embedded program.

Any changes made through the context menu immediately affect the internal switching logic, allowing developers to evaluate application behavior under different relay configurations and improve compatibility with multiple hardware variants.

Validation Mechanism

The Relay Module incorporates an automated validation system that verifies proper electrical connections before simulation begins. The validation module checks that the relay has been correctly connected to a power supply, ground, and a digital control output from the microcontroller.

If any required connection is missing, the simulator displays descriptive warning messages that assist users in correcting wiring errors. This validation mechanism improves circuit reliability, minimizes debugging time, and helps ensure successful relay operation during simulation.

Features of the Relay Module

The implemented Relay Module provides the following features:

- Accurate simulation of a single-channel 5 V electromechanical relay.
- Support for both **Active LOW** and **Active HIGH** trigger modes.
- Realistic switching between **Normally Open (NO)**, **Normally Closed (NC)**, and **Common (COM)** contacts.
- Dynamic monitoring of relay energization state based on the digital input signal.
- Automatic power-state detection to ensure realistic operation.
- Interactive trigger-level selection through the context menu.
- Real-time visual indication of relay status using power and activity LEDs.
- Detailed graphical representation closely matching commercially available relay modules.
- Built-in validation for power supply and digital control connections.

- Modular software architecture separating simulation logic, graphical interface, configuration, validation, and documentation, ensuring maintainability and future extensibility.

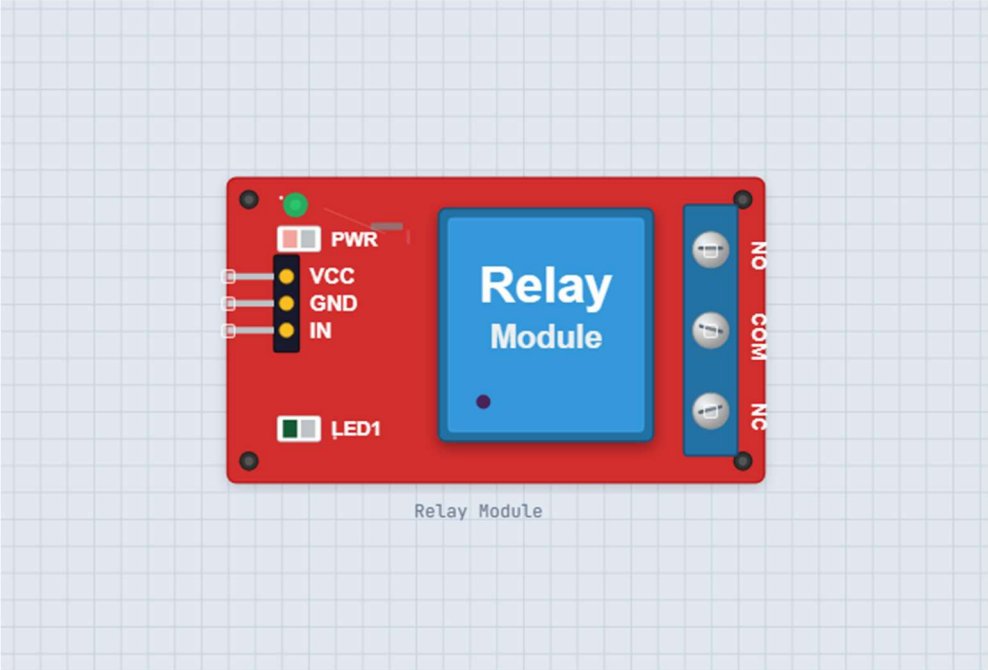

Fig 2.5

2.6 ADXL345 Accelerometer

The ADXL345 is a low-power, high-resolution, three-axis digital accelerometer capable of measuring acceleration along the X, Y, and Z axes. It is widely used in embedded systems for motion sensing, tilt detection, vibration monitoring, free-fall detection, orientation measurement, and activity recognition. The sensor communicates with microcontrollers through the Inter-Integrated Circuit (I^2C) communication protocol and provides high-resolution digital acceleration data suitable for a wide range of industrial, consumer, and IoT applications. In the simulation environment, the ADXL345 component accurately emulates the behavior of the physical sensor by implementing register-level communication, configurable acceleration values, dynamic data generation, power management, and an interactive graphical interface. This enables users to develop, test, and validate motion-based applications without requiring physical hardware.

Component Architecture

The ADXL345 component has been implemented using a modular software architecture consisting of component registration, simulation logic, graphical user interface, manifest configuration, and validation modules. Each module performs an independent function, improving software maintainability, scalability, and code readability.

The component registration module integrates the simulation logic, graphical interface, context menu, validation rules, and manifest configuration into a single component that can be dynamically loaded by the simulation engine. This modular design simplifies future enhancements while ensuring that updates to one module do not affect the remaining implementation.

Component Configuration

The manifest file defines the physical characteristics, default operating parameters, and electrical interface of the ADXL345 module. By default, the simulator initializes the sensor with **0 g** acceleration along the X-axis, **0 g** along the Y-axis, and **1 g** along the Z-axis to represent the effect of Earth's gravitational field on a stationary sensor.

The module exposes eight connection pins:

- **GND** – Ground reference.
- **VCC** – Power supply input.
- **CS** – Chip Select pin used for SPI/I²C mode selection.
- **INT1** – Interrupt Output 1.
- **INT2** – Interrupt Output 2.
- **SDO** – Serial Data Output and I²C address selection.
- **SDA** – Serial Data line.
- **SCL** – Serial Clock line.

These pin definitions accurately represent commercially available ADXL345 breakout boards used in embedded system development.

Simulation Logic

The operational behavior of the ADXL345 is implemented through the **ADXL345Logic** class, which models the functionality of the physical digital accelerometer. The simulator

maintains internal variables representing acceleration along the three orthogonal axes, power status, register pointer, communication state, and internal configuration registers.

During simulation, the component continuously monitors changes in acceleration values and synchronizes the updated measurements with the simulator state. Whenever users modify acceleration values through the graphical interface, the simulator immediately recalculates the corresponding register contents and makes the updated data available through the I²C communication interface. This enables realistic testing of motion sensing and orientation detection applications.

Power Management

The ADXL345 simulation continuously monitors the voltage supplied to the VCC pin. The component becomes operational only when the supply voltage exceeds approximately **2.0 V**, corresponding to the operating characteristics of commercially available ADXL345 breakout modules.

The simulator also reproduces the standby behavior of the physical sensor. After initialization, the ADXL345 remains in standby mode until the microcontroller sets the **Measure** bit in the **POWER_CTL (0x2D)** register. Until measurement mode is enabled, the sensor returns zero acceleration values, accurately representing the behavior of the actual hardware.

I²C Communication

Communication between the microcontroller and the ADXL345 is performed using the I²C serial communication protocol. The simulator fully implements standard I²C communication, including start and stop conditions, device addressing, register selection, sequential register access, register writing, and automatic register pointer incrementing.

The ADXL345 supports two possible I²C addresses:

- **0x53** (default when the SDO pin is LOW or left floating).
- **0x1D** (when the SDO pin is connected HIGH).

The simulator automatically determines the active I²C address according to the logic level applied to the SDO pin, enabling compatibility with different hardware configurations and standard Arduino ADXL345 libraries.

Register-Level Emulation

To accurately reproduce hardware behavior, the simulator implements the internal register map of the ADXL345. The component supports device identification registers, configuration registers, interrupt registers, power management registers, bandwidth configuration registers, offset registers, and acceleration output registers.

The **DEVID** register always returns the value **0xE5**, allowing embedded software to verify successful communication with the sensor. Configuration registers retain values written by the microcontroller, while the acceleration registers are dynamically updated according to the simulated motion data. This register-level emulation ensures compatibility with existing ADXL345 software libraries and embedded applications.

Acceleration Data Generation

The simulator dynamically generates acceleration measurements for the X, Y, and Z axes based on user-defined input values. These acceleration values are converted into raw digital data using the ADXL345's default full-resolution sensitivity of **256 LSB/g** and stored in the corresponding output registers.

The sensor outputs are organized using a little-endian format, where the least significant byte (LSB) is transmitted before the most significant byte (MSB). When measurement mode is disabled, all acceleration registers return zero, accurately reflecting the standby operation of the physical device. This implementation enables realistic testing of motion detection, inclination measurement, vibration analysis, and orientation sensing algorithms.

Interactive User Interface

The graphical user interface provides a realistic representation of a commercial ADXL345 breakout board. The SVG-based interface includes the printed circuit board, mounting holes, integrated accelerometer package, voltage regulator, passive electronic components, labelled connection pins, axis orientation indicators, and a digital telemetry display.

The telemetry display continuously presents the simulated acceleration values along the X, Y, and Z axes. A status indicator also reflects the current power state of the sensor, providing immediate visual feedback during simulation.

Interactive Configuration

The simulator provides an interactive context menu that allows users to modify acceleration values independently along each of the three axes. Slider controls and numerical input fields support acceleration values ranging from **−16 g to +16 g**, enabling users to simulate a wide variety of motion conditions.

Any changes made through the interface are immediately reflected within the internal sensor registers and graphical telemetry display, allowing embedded applications to receive updated motion data in real time without requiring changes to the application code.

Validation Mechanism

The ADXL345 component incorporates an automated validation system that verifies proper hardware configuration before simulation begins. The validation module checks whether the power supply, ground connection, SDA line, and SCL line have been correctly connected.

Additionally, the validation mechanism verifies the configuration of the **CS** pin. If the CS pin is connected to ground, the simulator warns the user that the sensor will operate in SPI mode rather than I²C mode, helping to prevent communication errors during circuit development. These validation routines significantly improve simulator usability while reducing debugging effort.

Features of the ADXL345 Component

The implemented ADXL345 component provides the following features:

- Accurate simulation of a three-axis digital accelerometer.
- Complete implementation of the I²C communication protocol with support for **0x53** and **0x1D** device addresses.
- Register-level emulation compatible with standard Arduino ADXL345 libraries.
- Dynamic generation of acceleration measurements along the X, Y, and Z axes.
- Automatic conversion of acceleration values into raw digital sensor data using the device's full-resolution sensitivity.
- Support for standby and measurement operating modes through the **POWER_CTL** register.
- Interactive adjustment of acceleration values through sliders and numeric input controls.

- Real-time telemetry display showing acceleration measurements and sensor status.
- Automatic power monitoring and I²C configuration validation.
- Built-in validation for power supply, communication lines, and SPI/I²C mode selection.
- Modular software architecture separating simulation logic, user interface, configuration, and validation, ensuring maintainability and future extensibility.

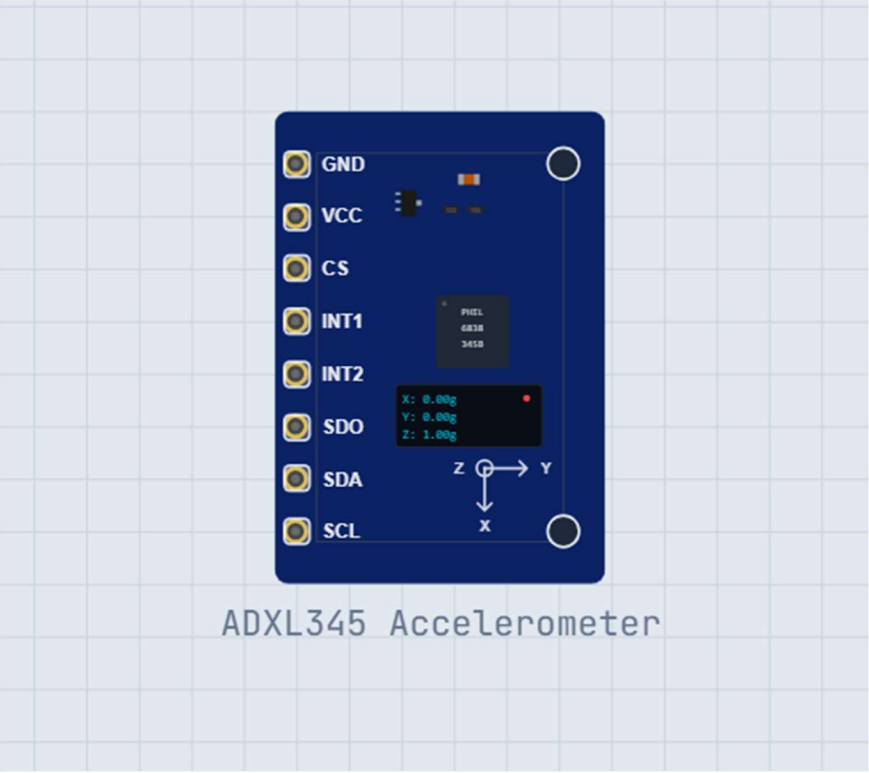

Fig 2.6

2.7 DS18B20 Digital Temperature Sensor Module

The DS18B20 Digital Temperature Sensor Module is a high-precision digital temperature sensing device that communicates using the 1-Wire communication protocol, requiring only a single data line for communication with the microcontroller. Unlike analog temperature sensors, the DS18B20 performs internal analog-to-digital conversion and transmits calibrated digital temperature data, eliminating the need for external analog signal conditioning. The sensor provides accurate temperature measurements over a wide operating range and is widely used in weather

monitoring systems, industrial automation, HVAC control, smart agriculture, medical equipment, and IoT-based environmental monitoring applications. In the simulation environment, the DS18B20 component accurately emulates the behavior of the physical sensor by implementing temperature simulation, digital data conversion, 1-Wire communication abstraction, configurable measurement parameters, and an interactive graphical interface. This enables users to develop and validate temperature monitoring applications without requiring physical hardware.

Component Architecture

The DS18B20 component is developed using a modular software architecture consisting of component registration, simulation logic, graphical user interface, manifest configuration, validation module, and integrated documentation. Each module performs a dedicated function, improving software maintainability, readability, scalability, and ease of future enhancement.

The component registration module integrates the simulation logic, graphical interface, context menu, validation rules, manifest configuration, and documentation into a single executable component. This modular approach enables the simulation engine to dynamically instantiate and manage the DS18B20 sensor while maintaining clear separation between functional and visual components.

Component Configuration

The manifest file defines the physical characteristics, default operating parameters, and electrical interface of the DS18B20 module. By default, the sensor initializes with a temperature value of **25°C** and a measurement resolution of **12 bits**, providing high-precision temperature readings at the start of simulation.

The module exposes three external connection pins:

- **GND** – Ground reference.
- **DQ** – Bidirectional 1-Wire digital communication pin.
- **VCC** – Power supply input supporting 3 V to 5.5 V operation.

These connections accurately represent commercially available DS18B20 breakout modules commonly used in embedded systems and IoT applications.

Simulation Logic

The core functionality of the DS18B20 component is implemented through the **DS18B20Logic** class, which models the operating characteristics of the physical digital temperature sensor. The simulator maintains internal variables representing the current temperature, raw digital temperature value, and connection status.

Whenever the simulated temperature is modified through the user interface, the component automatically recalculates the corresponding raw digital value and synchronizes the updated information with the simulator state. The simulator continuously monitors the power supply and updates the sensor status accordingly, ensuring realistic behavior throughout the simulation.

Temperature Measurement

The DS18B20 measures temperature digitally, eliminating the inaccuracies associated with analog voltage conversion. Within the simulator, users can configure the ambient temperature over the sensor's supported operating range of **−55°C to +125°C**.

The measured temperature is internally converted into the same digital format used by the physical DS18B20. Specifically, temperature values are represented as **16-bit signed integers** with a resolution of **1/16°C (0.0625°C per bit)**. For example, a temperature of **25°C** is stored internally as the hexadecimal value **0x0190 (400 decimal)**. This implementation accurately reproduces the digital measurement format expected by standard DS18B20 software libraries.

1-Wire Communication

Communication between the microcontroller and the DS18B20 is performed using the **1-Wire communication protocol**, which requires only a single bidirectional data line in addition to power and ground connections. This communication method significantly reduces wiring complexity compared to conventional serial interfaces.

Rather than simulating the complete bit-level timing of the 1-Wire protocol, the simulator implements communication at the library abstraction level. The simulation engine directly provides the configured temperature value to applications using the **OneWire** and **DallasTemperature** libraries, ensuring compatibility while maintaining efficient simulation performance. The DQ line is also maintained in its idle HIGH state whenever the sensor is powered, accurately reflecting the behavior of a properly pulled-up 1-Wire bus.

Power Management

The DS18B20 component continuously monitors the voltage supplied to the VCC terminal. The sensor becomes operational only when the supply voltage exceeds approximately **2.5 V**. Once powered, the simulator places the DQ communication line in its idle HIGH state, representing the effect of the required pull-up resistor on the 1-Wire bus.

If the module is not adequately powered, communication is disabled and the sensor reports a disconnected status. This realistic power management enables users to detect power-related circuit issues during simulation before hardware implementation.

Temperature Data Generation

The simulator dynamically generates digital temperature data based on the user-defined temperature value. Whenever the temperature changes, the simulator converts the new measurement into the corresponding 16-bit raw value used internally by the DS18B20 and updates the component state accordingly.

This approach accurately models the behavior of the physical sensor, allowing embedded applications to receive realistic digital temperature measurements without implementing the complete low-level timing requirements of the 1-Wire protocol. As a result, applications developed using standard Arduino temperature libraries execute correctly within the simulation environment.

Interactive User Interface

The DS18B20 component includes a detailed graphical representation of a commercial blue breakout module. The SVG-based interface depicts the printed circuit board, TO-92 sensor package, passive electronic components, indicator LED, PCB traces, mounting hardware, and clearly labelled connection pins.

The realistic graphical design closely resembles commercially available DS18B20 modules, enabling users to easily recognize the component and construct practical embedded circuits within the simulator.

Interactive Configuration

The simulator provides an interactive context menu that allows users to modify the simulated temperature and sensor resolution during execution. Users can directly specify the desired temperature value as well as select the measurement resolution from **9-bit**, **10-bit**, **11-bit**, or **12-bit** operation.

Changes made through the context menu are immediately reflected in the simulated sensor output, enabling real-time testing of embedded applications under different environmental conditions without modifying the program source code.

Validation Mechanism

To improve circuit reliability, the DS18B20 component incorporates an automated validation system that verifies correct hardware connections before simulation begins. The validation module checks that the power supply, ground, and DQ communication line are properly connected.

Additionally, the simulator verifies the presence of the required **4.7 kΩ pull-up resistor** between the DQ pin and VCC. If the resistor or any required connection is missing, informative warning messages are displayed to assist users in correcting the circuit configuration. These validation routines reduce wiring errors and improve the accuracy of simulated 1-Wire communication.

Features of the DS18B20 Component

The implemented DS18B20 component provides the following features:

- Accurate simulation of a digital temperature sensor using the **1-Wire communication protocol**.
- Dynamic temperature measurement over a range of **−55°C to +125°C**.
- Automatic conversion of temperature into the 16-bit digital format used by the physical DS18B20.
- Support for configurable measurement resolutions of **9-bit**, **10-bit**, **11-bit**, and **12-bit**.
- Library-level compatibility with **OneWire** and **DallasTemperature** Arduino libraries.
- Automatic monitoring of sensor power status and communication availability.
- Interactive adjustment of temperature and resolution through the context menu.
- Realistic graphical representation of a commercial DS18B20 breakout module.
- Built-in validation for power supply, data communication, and mandatory **4.7 kΩ pull-up resistor** connections.

- Modular software architecture separating simulation logic, graphical interface, configuration, validation, and documentation, ensuring maintainability and future extensibility.

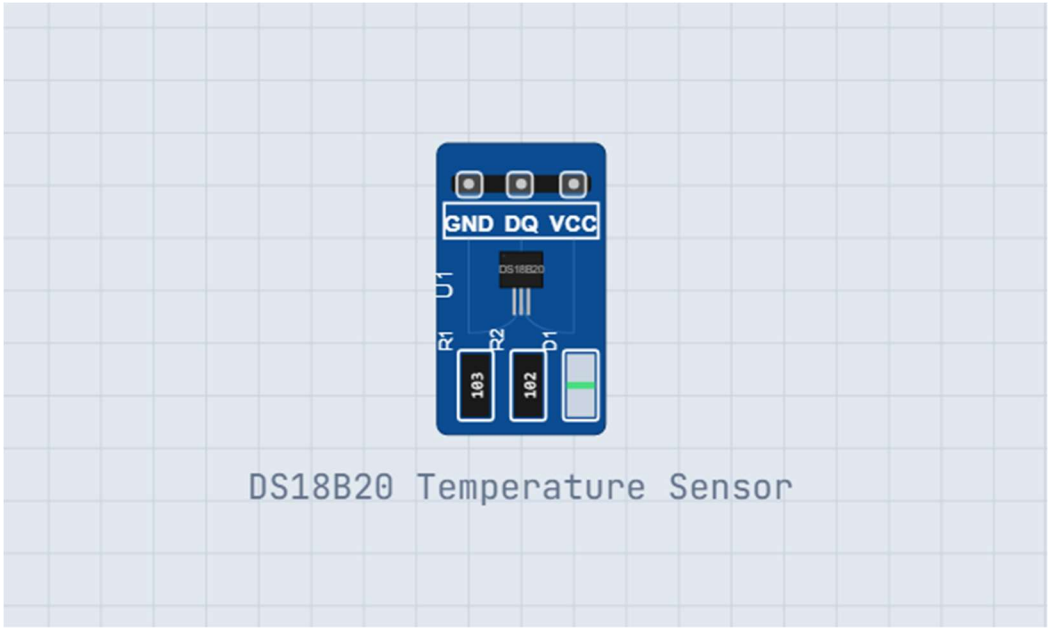

Fig 2.7

2.8 IR Receiver Module

The IR Receiver Module is a digital infrared sensing device designed to receive and demodulate infrared signals transmitted from remote controls operating at a carrier frequency of 38 kHz. The module converts modulated infrared light into digital pulses that can be interpreted by a microcontroller using standard infrared communication protocols such as the NEC protocol. It is widely used in television remote controls, home automation systems, consumer electronics, robotics, wireless control systems, and embedded applications requiring contactless user interaction. In the simulation environment, the IR Receiver Module accurately reproduces the behavior of a physical TSOP38238/VS1838B-style receiver by implementing NEC protocol transmission, active-LOW digital output, configurable carrier frequency, power management, virtual remote control interaction, and a realistic graphical user interface. This enables users to develop and validate infrared communication applications without requiring physical hardware.

Component Architecture

The IR Receiver Module is implemented using a modular software architecture consisting of component registration, simulation logic, graphical user interface, manifest configuration, and validation modules. Each module performs a dedicated function, improving software maintainability, scalability, readability, and ease of future development.

The component registration module integrates the simulation logic, graphical interface, context menu, validation rules, and manifest configuration into a single executable component that can be dynamically loaded by the simulator. This modular implementation ensures that future improvements to the communication protocol or user interface can be incorporated without affecting the remaining functionality.

Component Configuration

The manifest file defines the physical characteristics, default operating parameters, and electrical interface of the IR Receiver Module. The module is configured by default to receive infrared signals with a 38 kHz carrier frequency, which is the standard frequency used by most commercial infrared remote controls.

The module exposes three external connection pins:

- GND – Ground reference.
- VCC – Power supply input (3.3 V–5 V).
- OUT – Active-LOW digital signal output.

These pin definitions accurately represent commercially available TSOP38238 and VS1838B infrared receiver modules commonly used with Arduino and other embedded development platforms.

Simulation Logic

The operational behavior of the IR Receiver Module is implemented through the IRReceiverLogic class, which emulates the functionality of a physical infrared receiver. The

simulator maintains internal variables representing power status, transmission state, pulse queue, current pulse, received button, and decoded infrared value.

Whenever a user presses a button on the virtual remote control, the simulator retrieves the corresponding NEC command, generates the complete infrared pulse sequence, and transmits the resulting digital waveform through the OUT pin. Throughout the transmission, the internal state is continuously synchronized with the graphical interface, providing real-time visual feedback during simulation.

Infrared Signal Reception

The primary function of the IR Receiver Module is to detect infrared signals emitted by remote transmitters and convert them into digital pulse sequences suitable for decoding by a microcontroller. Commercial IR receivers internally filter ambient light, amplify the received infrared signal, demodulate the 38 kHz carrier, and produce a clean digital output.

Within the simulator, this behavior is reproduced by generating an active-LOW digital waveform whenever a virtual remote button is pressed. The generated signal follows the timing characteristics of the NEC infrared communication protocol, allowing Arduino programs using the IRremote library to operate without modification.

NEC Protocol Implementation

The simulator fully implements the transmission sequence of the widely used NEC infrared protocol. Every virtual button corresponds to a predefined 32-bit NEC command that is transmitted using precise pulse timing.

The generated transmission includes:

- A 9 ms leader (AGC) pulse.
- A 4.5 ms synchronization space.
- Thirty-two data bits representing the command.
- Individual bit encoding using a 562 µs mark followed by either a 562 µs space (logic 0) or a 1687 µs space (logic 1).

- A final stop pulse completing the transmission.

This implementation accurately reproduces the timing characteristics of commercial infrared remote controls, enabling realistic testing of infrared decoding algorithms within the simulation environment.

Digital Output Operation

The IR Receiver Module produces an active-LOW digital output, matching the behavior of physical TSOP-series infrared receivers. During normal operation, when no infrared signal is present, the OUT pin remains at a HIGH logic level.

Whenever an infrared transmission is received, the output alternates between LOW and HIGH according to the generated NEC pulse sequence. LOW levels represent infrared bursts (marks), while HIGH levels represent the spaces between bursts. This behavior enables direct compatibility with Arduino digital input pins and commonly used infrared software libraries.

Power Management

The simulator continuously monitors the voltage supplied to the VCC terminal. The IR Receiver Module becomes operational only when the supply voltage reaches approximately 2.5 V or higher.

If sufficient power is unavailable, the module disables infrared reception, forces the output inactive, and prevents virtual remote transmissions from being processed. Once adequate power is restored, the receiver immediately resumes normal operation, accurately reproducing the behavior of a physical infrared receiver module.

Interactive User Interface

The graphical user interface provides a detailed representation of a commercial VS1838B/TSOP38238 infrared receiver module. The SVG-based design includes the printed circuit board, infrared receiver package, passive electronic components, indicator LED, mounting holes, connection headers, silkscreen markings, and labelled input/output pins.

The interface also incorporates dynamic visual indicators. The power LED illuminates whenever the module is powered, while the infrared receiver dome changes appearance during active signal reception, allowing users to observe infrared communication in real time.

Interactive Configuration

The simulator provides an interactive virtual remote control that allows users to transmit infrared commands during simulation. By clicking the IR Receiver Module, users can open the virtual remote interface containing commonly used buttons such as POWER, OK, Volume, Channel, directional keys, and numeric buttons.

Each button transmits its corresponding NEC command immediately, allowing embedded applications to receive realistic infrared signals without additional hardware. The component also provides a configuration menu through which users can select carrier frequencies of 36 kHz, 38 kHz, 40 kHz, or 56 kHz, enabling simulation of different infrared receiver variants.

Validation Mechanism

The IR Receiver Module incorporates an automated validation system that verifies correct circuit configuration before simulation begins. The validation module checks that the power supply, ground connection, and digital output pin are correctly connected.

Additionally, the simulator verifies that the OUT pin is connected to a valid digital input rather than an analog-only input. If any required connection is missing or incorrectly configured, descriptive warning messages are displayed to assist users in correcting wiring errors before simulation begins.

Features of the IR Receiver Module

The implemented IR Receiver Module provides the following features:

- Accurate simulation of a 38 kHz infrared receiver compatible with TSOP38238 and VS1838B modules.
- Complete implementation of the NEC infrared communication protocol with realistic pulse timing.

- Support for a wide range of predefined remote control commands.
- Active-LOW digital output compatible with the Arduino IRremote library.
- Interactive virtual remote control for real-time infrared signal transmission.
- Configurable carrier frequency selection (36 kHz, 38 kHz, 40 kHz, and 56 kHz).
- Automatic power monitoring with realistic receiver activation and deactivation.
- Dynamic graphical interface with visual indicators for power and infrared reception.
- Built-in validation for power supply, digital output connections, and correct pin usage.
- Modular software architecture separating simulation logic, graphical interface, configuration, validation, and documentation, ensuring maintainability and future extensibility.

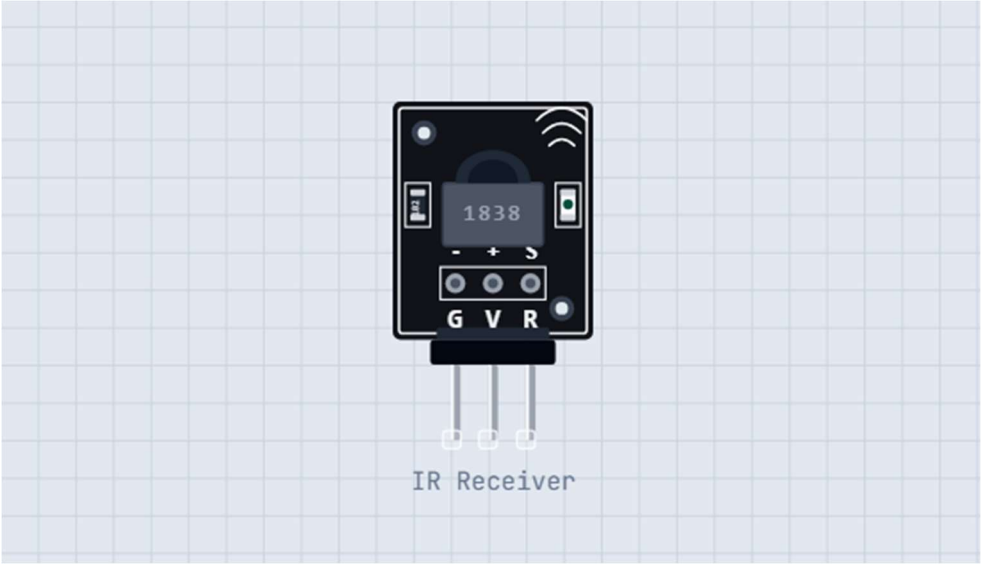

Fig 2.8

2.9 MFRC522 RFID Reader Module

The MFRC522 RFID Reader Module is a low-cost, high-performance radio frequency identification (RFID) reader operating at a carrier frequency of **13.56 MHz**. It communicates with microcontrollers through the **Serial Peripheral Interface (SPI)** protocol and is capable of reading

the unique identification (UID) stored in MIFARE Classic RFID cards and key fobs. The module is extensively used in access control systems, attendance monitoring, smart payment systems, inventory management, asset tracking, library automation, and IoT-based authentication applications. In the simulation environment, the MFRC522 component accurately emulates the behavior of a physical RFID reader by implementing SPI communication, configurable RFID card presence, dynamic UID generation, power monitoring, and an interactive graphical interface. This enables developers to design, test, and validate RFID-based embedded applications without requiring physical RFID hardware.

Component Architecture

The MFRC522 RFID Reader Module is implemented using a modular software architecture consisting of component registration, simulation logic, graphical user interface, manifest configuration, validation module, and integrated documentation. Each module performs an independent function, improving maintainability, scalability, readability, and future extensibility.

The component registration module combines the simulation logic, graphical interface, context menu, validation rules, manifest configuration, and documentation into a single executable component that can be dynamically loaded by the simulator. This modular organization enables independent enhancement of the communication protocol, user interface, and simulation engine without affecting the remaining modules.

Component Configuration

The manifest file defines the physical dimensions, default operating parameters, and electrical interface of the MFRC522 RFID Reader Module. By default, the simulator initializes the module with **no RFID card present** and uses a default card UID of **DE AD BE EF** whenever a virtual RFID card is enabled.

The module exposes eight external connection pins:

- **3V3** – 3.3 V power supply input.

- **RST** – Hardware reset input.
- **GND** – Ground reference.
- **IRQ** – Interrupt output.
- **MISO** – SPI Master-In Slave-Out data line.
- **MOSI** – SPI Master-Out Slave-In data line.
- **SCK** – SPI serial clock.
- **SDA (SS)** – SPI chip select input.

These connections accurately represent commercially available MFRC522 RFID reader modules used with Arduino and other embedded development platforms.

Simulation Logic

The operational behavior of the RFID reader is implemented through the **MFRC522Logic** class, which models the essential operating characteristics of the physical MFRC522 integrated circuit. The simulator maintains internal variables representing the power state, RFID card presence, configured card UID, and communication status.

During simulation, the module continuously monitors changes in the power supply and user interactions. Whenever a virtual RFID card is presented, the simulator updates its internal state and makes the configured UID available through the SPI communication interface. Removal of the card immediately terminates data availability, accurately reproducing the behavior of a physical RFID reader.

RFID Card Detection

The primary function of the MFRC522 module is to detect RFID tags operating at **13.56 MHz** and retrieve their unique identification number (UID). In practical systems, the reader

continuously generates an electromagnetic field that powers passive RFID tags when they enter the operating range. The tag then transmits its UID back to the reader through inductive coupling.

Within the simulator, RFID detection is reproduced using a configurable **Card Present** state. When enabled, the simulator behaves as though a valid RFID card has entered the reader's operating field, allowing Arduino programs to detect the card and read its UID. When the card is removed, communication immediately ceases, accurately reflecting the behavior of real RFID hardware.

SPI Communication

Communication between the microcontroller and the MFRC522 module is performed using the **Serial Peripheral Interface (SPI)** protocol. The module exchanges data using the **MOSI**, **MISO**, **SCK**, and **SDA (Chip Select)** pins, while the **RST** pin is used to initialize the reader.

The simulator implements a simplified SPI communication model that provides the configured UID whenever a valid RFID card is present. If the reader is not powered or no card is available, the simulator returns **0xFF**, matching the inactive communication behavior expected by many embedded applications. This implementation enables compatibility with commonly used Arduino MFRC522 libraries while maintaining efficient simulation performance.

UID Data Generation

Each RFID tag contains a unique identification number that distinguishes it from every other tag. The simulator allows users to configure this UID manually, enabling testing of authentication systems using different RFID credentials.

Internally, the configured hexadecimal UID string is converted into an array of bytes. During SPI communication, the simulator sequentially returns these UID bytes whenever the microcontroller requests RFID data. This implementation allows developers to verify access-control logic, user

authentication routines, and RFID-based identification algorithms without requiring multiple physical RFID cards.

Power Management

The MFRC522 RFID Reader Module continuously monitors the voltage supplied to the **3V3** input. Unlike many other embedded modules, the MFRC522 operates exclusively within a supply voltage range of approximately **1.8 V to 3.6 V**, with **3.3 V** being the recommended operating voltage.

If the module receives a valid supply voltage, RFID communication becomes available. If the supply voltage is outside the supported operating range or the module is unpowered, RFID communication is disabled and no UID data is transmitted. This behavior closely reproduces the operating characteristics of the physical MFRC522 integrated circuit.

Interactive User Interface

The graphical user interface provides a realistic representation of a commercial MFRC522 RFID reader module. The SVG-based interface includes the printed circuit board, RFID antenna coil, MFRC522 integrated circuit, crystal oscillator, passive electronic components, mounting holes, labelled pin headers, and indicator LED.

Whenever an RFID card is virtually presented, animated ripple effects appear around the antenna region, visually representing the electromagnetic detection field and providing immediate feedback that a card has entered the reader's operating range.

Interactive Configuration

The simulator provides an interactive context menu that allows users to control RFID operation during simulation. Users can enable or disable **Card Present** mode to simulate placing or removing an RFID tag from the reader.

The simulation also supports configurable RFID card UIDs, allowing developers to test authentication algorithms using different RFID identities without requiring multiple physical cards. Any configuration changes are immediately reflected within the simulation, enabling real-time testing of RFID applications.

Validation Mechanism

To improve simulation reliability, the MFRC522 component incorporates an automated validation system that verifies proper circuit configuration before simulation begins. The validation module checks that the reader is connected to a valid **3.3 V** power supply and ensures that all required SPI communication lines (**MISO**, **MOSI**, **SCK**, and **SDA**) are correctly connected.

Additionally, the simulator warns users if the module appears to be connected to a **5 V** supply, since applying 5 V to a physical MFRC522 module can permanently damage the hardware. These validation routines significantly reduce wiring errors while improving the accuracy of RFID communication during simulation.

Features of the MFRC522 RFID Reader Module

The implemented MFRC522 RFID Reader Module provides the following features:

- Accurate simulation of a **13.56 MHz MFRC522 RFID reader**.
- Complete support for **SPI-based communication** with Arduino-compatible applications.
- Simulation of RFID card detection using configurable **Card Present** mode.
- Dynamic generation of user-defined RFID **UID** values for authentication testing.
- Automatic power monitoring supporting the module's **3.3 V operating requirement**.
- Simplified SPI response mechanism compatible with standard **MFRC522 Arduino libraries**.
- Interactive graphical interface with animated RFID detection visualization.
- Real-time configuration of RFID card presence and UID values during simulation.

- Built-in validation for power supply voltage and SPI communication wiring.
- Modular software architecture separating simulation logic, graphical interface, configuration, validation, and documentation, ensuring maintainability and future extensibility.

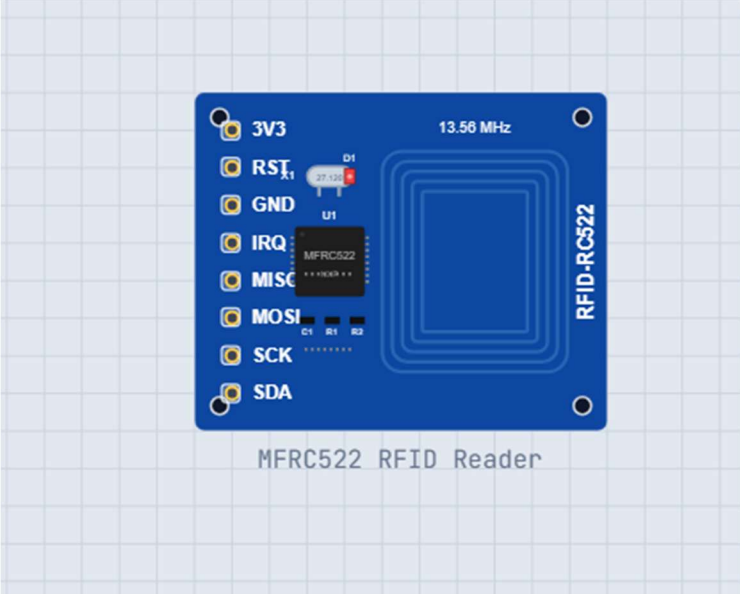

Fig 2.9

2.10 Graphical User Interface (SVG-Based Component Design)

A major part of the internship involved designing graphical user interfaces (GUIs) for each electronic component to provide a realistic and interactive simulation experience. The interfaces were developed using **Scalable Vector Graphics (SVG)**, allowing components to be rendered with high visual quality while remaining resolution-independent. Each component was carefully modeled to resemble its physical counterpart, including the printed circuit board (PCB), integrated circuits, passive components, connectors, pin headers, mounting holes, silkscreen labels, and indicator LEDs.

The graphical interface was designed not only for visual representation but also to provide real-time feedback during simulation. Dynamic visual elements such as power indicators, sensor readings, relay states, RFID detection animations, and communication status were incorporated to

improve usability and help users understand the operational state of each component. SVG was selected because it enables lightweight rendering, easy customization, and efficient integration within the web-based simulator. The developed interfaces significantly enhanced the overall user experience by making the simulated components visually consistent with commercially available hardware modules.

2.11 Interactive Context Menu Implementation

To improve the flexibility of the simulator, interactive context menus were implemented for every developed component. These context menus allow users to modify component parameters directly during simulation without changing the Arduino program or restarting the simulation environment.

Different components expose different configurable parameters based on their functionality. For example, environmental sensors allow modification of temperature, pressure, and acceleration values, while communication modules enable changing RFID card presence, card UID, relay trigger modes, or date and time values. User inputs are immediately synchronized with the internal simulation logic, enabling real-time testing under different operating conditions.

The implementation of context menus provides a highly interactive simulation environment where users can easily emulate changing environmental conditions and hardware states. This feature improves the educational value of the simulator by allowing developers to observe how embedded systems respond dynamically to external inputs.

2.12 Component Documentation Development

Comprehensive technical documentation was prepared for every component developed during the internship. The documentation was designed to assist users in understanding the purpose, operation, electrical characteristics, and practical usage of each component within the simulator.

Each documentation page includes a detailed description of the component, pin configuration tables, communication protocols, operating principles, wiring instructions, example Arduino programs, expected outputs, and simulation-specific notes. The documentation also explains any

special configuration requirements such as power supply limitations, pull-up resistors, or initialization procedures.

This documentation was integrated directly into the simulator, allowing users to access component information without referring to external resources. The integrated documentation significantly improves the usability of the simulator, particularly for beginners and students learning embedded systems.

2.13 Component Validation Rules

To improve simulation reliability and reduce user errors, validation mechanisms were developed for each component. These validation rules automatically examine circuit connections before simulation begins and verify that all required electrical connections have been correctly established.

The validation system performs checks such as verifying power supply connections, ground connections, communication interfaces (I^2C, SPI, and 1-Wire), voltage compatibility, mandatory pull-up resistors, and correct pin assignments. Whenever an invalid configuration is detected, descriptive warning messages are generated to guide users in correcting the circuit.

These validation mechanisms help prevent common wiring mistakes that frequently occur during embedded system development. By detecting configuration errors before execution, the simulator provides a more realistic and user-friendly development environment while reducing debugging time.

2.14 Simulation Logic and Hardware Protocol Implementation

The core functionality of every component was implemented through dedicated simulation logic that accurately models the behavior of the corresponding physical hardware. Each component maintains internal states, processes user interactions, updates sensor values, and communicates with the simulator through well-defined interfaces.

Multiple communication protocols were implemented to ensure compatibility with standard Arduino libraries. Sensor modules such as the BMP180, DS1307, MPU6050, and ADXL345

utilize the **I²C communication protocol**, while the MFRC522 RFID Reader communicates through the **SPI protocol**, and the DS18B20 Digital Temperature Sensor operates using the **1-Wire protocol**. The simulation logic also incorporates power management, register-level emulation, communication state handling, and real-time data synchronization.

By accurately reproducing hardware communication protocols and internal device behavior, the simulator allows existing Arduino applications to execute without modification, providing a realistic environment for software development, debugging, and educational demonstrations.

2.15 Component Testing and Verification

Comprehensive testing and verification were performed throughout the development process to ensure the correctness and reliability of each simulated component. Individual components were tested independently to verify graphical rendering, parameter updates, communication interfaces, validation rules, and overall functionality.

Integration testing was then conducted by connecting the simulated components with Arduino boards and executing example programs using standard Arduino libraries. Sensor outputs, communication responses, graphical updates, and Serial Monitor outputs were compared with the expected behavior of physical hardware to ensure accurate simulation.

Various operating conditions, including incorrect wiring, insufficient power supply, invalid communication settings, and dynamic parameter changes, were also evaluated. These tests confirmed that the developed components responded correctly under both normal and abnormal operating conditions, thereby improving the robustness, reliability, and educational effectiveness of the simulator.

Chapter 3

Telemetry and Serial Output Fixes

Beyond building new components, a significant portion of my internship was spent investigating and fixing issues with the telemetry pipeline and the Serial Monitor output of existing components. These issues primarily affected the reliability of the live-data HUD and the correctness of Serial.print() / Serial.println() streams during simulation — both of which are central to how students debug their sketches.

3.1 Telemetry Pipeline Issues

Several existing sensor components were producing stale or partially-updated telemetry values: the live HUD would show the last-known value instead of the current simulated reading, or attribute changes made through the context menu would not propagate to the running Arduino sketch until the simulation was restarted.

- Audited the telemetry event-emission paths in the affected components and identified missing update notifications when context-menu attribute changes occurred mid-simulation.
- Patched the affected components so that every attribute mutation now pushes a fresh telemetry frame to the HUD.
- Verified the fix by exercising each component with rapid attribute changes and confirming that the HUD, the I2C/analog reads inside the sketch, and the Serial Monitor remained consistent.

3.2 Serial Output Reliability

A second class of issues affected Serial Monitor output. Some components were emitting truncated lines, dropping characters at high baud rates, or interleaving output from different timer callbacks. These problems were particularly visible when sketches printed sensor readings inside tight loops.

- Reproduced the truncation by writing minimal test sketches that printed at 9600, 57600, and 115200 baud.

- Identified buffering races between the component's data-ready callback and the simulator's serial-emission queue.
- Adjusted the affected components to serialize their writes through the platform's serial queue, eliminating the interleaving and truncation.
- Confirmed clean, ordered output across all three baud rates after the fix.

3.3 Outcome

After these fixes, the affected components produce telemetry and serial output that students and educators can trust as a true reflection of their sketch's behavior. This is a small but high-leverage improvement: incorrect Serial Monitor output is one of the most demoralising experiences for a beginner debugging embedded code, and the fix removes a class of distractions that previously masked real bugs in user sketches.

Chapter 4

Summary

My summer internship with the **FOSSEE OpenHW Studio** team resulted in significant contributions to the enhancement of the OpenHW Studio simulator. The major deliverables completed during the internship are summarized below:

- **Development of nine new simulated hardware components**—BMP180 Pressure Sensor, DS1307 Real-Time Clock (RTC), MPU6050 Inertial Measurement Unit (IMU), NTC Thermistor Module, Relay Module, ADXL345 Accelerometer, DS18B20 Digital Temperature Sensor, IR Receiver Module, and MFRC522 RFID Reader. Each component was implemented with accurate pin configurations, configurable attributes, runtime simulation logic, realistic graphical user interfaces, interactive context menus, validation rules, and integrated technical documentation.

- **Enhancement of component simulation and user interaction** by designing SVG-based graphical interfaces, implementing interactive runtime configuration menus, developing validation mechanisms to detect incorrect wiring and configuration errors, and creating comprehensive documentation to improve the usability and educational value of the simulator.

- **Fixes to the telemetry pipeline** of existing simulator components by resolving stale-value synchronization issues. These improvements ensured that real-time telemetry displayed in the Heads-Up Display (HUD), context menu values, and Arduino sketch outputs remained synchronized throughout the simulation.

- **Improvements to Serial Monitor reliability** by resolving issues related to incomplete data transmission, output truncation, and interleaved serial messages across multiple components and baud rates. These fixes provided consistent and accurate serial communication, making debugging and monitoring of embedded applications more reliable.

Collectively, these contributions expanded the OpenHW Studio component library, enhanced the accuracy and reliability of component simulation, improved runtime telemetry and serial communication, and significantly strengthened the platform's effectiveness as an educational and prototyping tool for embedded systems and electronics.

Chapter 5

Conclusion

This internship has been a deeply enriching and rewarding experience — both technically and creatively. Contributing to the OpenHW Studio platform by building new simulated components and improving the reliability of the telemetry and serial-output infrastructure not only challenged me but also gave me the opportunity to grow as a developer of embedded-systems software.

The process of designing manifests, implementing runtime behavior, authoring documentation, and then debugging and refining each component felt like crafting a small piece of educational infrastructure that real students will use. I am sincerely grateful to my mentors for their constant guidance, support, and encouragement throughout the internship. Their insights played a vital role in shaping the direction and outcome of my work.

I believe OpenHW Studio — strengthened with these nine new components and a more reliable telemetry pipeline — holds great potential as both an educational and a research tool. By enabling a more intuitive learning experience through realistic, browser-based hardware simulation, it can empower students, educators, and researchers alike. The combination of technical depth and accessibility makes it especially useful for those venturing into electronics and embedded systems for the first time.

Chapter 6

Future Work

The component additions and telemetry/serial fixes I delivered during this internship represent a meaningful step forward for OpenHW Studio. Several promising directions remain open for future work to further improve the simulator's usability, scalability, and educational value.

6.1 Expanding the Component Library

- Add more I2C sensor modules (BME280, SHT31, VL53L0X) to cover environmental and time-of-flight sensing.
- Add common displays such as the SSD1306 OLED and ILI9341 TFT for richer UI-driven projects.
- Add motor-driver modules (L298N, TB6612FNG) and servo banks for full robotics curricula.

6.2 Telemetry and Live-Data Improvements

- Surface a unified, dockable Live-Data panel that aggregates telemetry from every component on the canvas.
- Add recordable telemetry traces with export to CSV, so educators can build lab assignments around captured data.
- Add scriptable scenarios that drive component attributes through time, enabling automated demonstrations and tests.

6.3 Serial Monitor Enhancements

- Add multi-tab Serial Monitor support so that projects with multiple virtual ports can be debugged side-by-side.
- Add a Serial Plotter mode to visualise streamed numeric values directly from print statements.
- Provide configurable line-ending and timestamping options to match the conventions used in physical Arduino IDE workflows.

References

Component	Primary Reference	Additional Reference
BMP180 Pressure Sensor	Bosch Sensortec. *BMP180 Digital Pressure Sensor Datasheet*, Rev. 2.8, 2013.	Adafruit Industries. *BMP085/BMP180 Barometric Pressure Sensor Guide*.
DS1307 RTC Module	Maxim Integrated. *DS1307 64 × 8 Serial Real-Time Clock Datasheet*.	Adafruit Industries. *RTClib Library Documentation*.
MPU6050 IMU Sensor	InvenSense. *MPU-6000/MPU-6050 Product Specification*, Rev. 3.4.	Electronic Cats. *MPU6050 Arduino Library Documentation*.
NTC Thermistor Module (LM393)	Texas Instruments. *LM393 Dual Differential Comparator Datasheet*.	Vishay. *NTC Thermistors Technical Guide*.
Relay Module	Songle Relay. *SRD-05VDC-SL-C Relay Datasheet*.	Arduino Documentation – Digital Output.
ADXL345 Accelerometer	Analog Devices. *ADXL345 Digital Accelerometer Datasheet*.	Adafruit. *ADXL345 Triple-Axis Accelerometer Guide*.
DS18B20 Temperature Sensor	Maxim Integrated. *DS18B20 Programmable Resolution 1-Wire Digital Thermometer Datasheet*.	Miles Burton. *DallasTemperature Arduino Library*.
IR Receiver (VS1838B / TSOP38238)	Vishay Semiconductors. *TSOP38238 IR Receiver Module Datasheet*.	Arduino-IRremote Library Documentation.
MFRC522 RFID Reader	NXP Semiconductors. *MFRC522 Standard Performance MIFARE and NTAG Frontend Datasheet*.	Miguel Balboa. *MFRC522 Arduino Library Documentation*.

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.

[13] Arduino-IRremote Project, *IRremote Library Documentation*, GitHub.

[14] Adafruit Industries, *RTClib Arduino Library Documentation*.

[15] Miles Burton, *DallasTemperature Arduino Library Documentation*.