SHAPE-SAT IMPLEMENTATION REPORT

Architecture and Realization of the SHAPE-SAT IoT Satellite Demonstrator

Project Implementation Report

Prepared by

SAHIL KUMAR SHARMA

Abstract

This document presents the practical implementation of **SHAPE-SAT (Student High-Altitude Payload Experiment / Satellite Demonstrator)**, a compact, solar-powered Internet of Things (IoT) satellite demonstrator based on the ESP32 microcontroller. The system is implemented as a realistic small-satellite platform with a *flight segment* running on the ESP32 and a *ground segment* implemented as a Python-based ground station application. The flight segment integrates a full set of sensors and modules: BH1750 light sensor, BMP280 pressure and temperature sensor, MPU6050 IMU, DS3231 real-time clock on a HW-111 module, INA219 current and voltage sensor, DHT11 temperature and humidity sensor, CJMCU-GUVA-S12SD UV sensor, MQ-2 gas sensor, PCA9548A I^2C multiplexer, micro SD card interface, RGB status LED, buzzer, solar panels, CN3791 solar charging module, and MT3608 DC–DC boost converter, with provision for an ESP32-CAM module for future image capture.

Dedicated firmware sketches (`Maincpu.ino`, `SensorTest.ino`, and `GPSSetup.ino`) implement system modes, health monitoring, calibration, GPS visualization, and wireless dashboards. Telemetry is provided simultaneously to a web-based dashboard (viewed on a mobile phone) and to a desktop ground station that supports both USB serial and Wi-Fi (HTTP/JSON) connections. This report focuses on *what was actually built and tested*: the sensor and module integration, operating modes, software features, and the observed results illustrated with real hardware and software screenshots.

Contents

Chapter 1

Introduction

SHAPE-SAT (Student High-Altitude Payload Experiment / Satellite Demonstrator) was conceived as a complete, end-to-end platform that mimics the architecture and behavior of a small satellite while remaining accessible as an educational and research project. The goal was not only to design the system on paper, but to *build, wire, program, test, and operate* a working prototype.

The implemented system can be logically divided into:

- A flight segment running on an ESP32 microcontroller with multiple sensors, power management, and communication interfaces.
- A ground segment consisting of a Python-based ground station GUI and a phone-friendly web dashboard.
- Supporting diagnostic sketches and tools for sensor connection testing and GPS visualization.

The following sections document:

1. The actual sensors and modules used and how they are wired.
2. The main firmware features, operating modes, and self-test routines.
3. The ground station and phone dashboard views.
4. What was achieved during implementation and testing.

Where appropriate, photographs from the assembled hardware and screenshots from the dashboards are included to show the real system.

Chapter 2

Overall System Overview

2.1 Flight Segment

The flight segment is built around an ESP32 development board that acts as:

- Central controller for all sensors and peripherals.
- Telemetry source for the ground station and phone dashboard.
- Power-aware node that monitors bus voltage and current.

The ESP32 communicates over:

- I^2C via a PCA9548A multiplexer for BH1750, BMP280, MPU6050, DS3231 RTC, INA219 and (optionally) OLED and other devices.
- GPIO / one-wire for DHT11.
- ADC inputs for the analog UV sensor (GUVA-S12SD) and the MQ-2 gas sensor.
- UART2 for the NEO-6M GPS module (in the dedicated GPS setup sketch).
- SPI for the micro SD card adapter and optional wireless modules such as NRF24L01.
- Wi-Fi for the built-in web dashboard, JSON API, and (in future) ESP32-CAM image transfer.

Power is supplied by a pair of solar panels feeding a CN3791-based charger that charges a Li-ion/LiPo battery bank. An MT3608 boost converter then generates a stable supply for the ESP32 and the sensor stack. An INA219 module continuously measures bus voltage and current to profile the power subsystem. Provision is also made for an ESP32-CAM module acting as an imaging payload, triggered by the main ESP32 and powered from the same regulated rail when the power budget permits.

2.2 Ground Segment

The ground segment currently consists of two main elements:

- A **Python desktop ground station** (`ground_station.py`) with a Tkinter GUI, Matplotlib plots, and an attitude visualization using IMU data.
- A **phone-friendly Wi-Fi dashboard** hosted directly on the ESP32, accessible via a standard browser and auto-refreshing every few seconds.

The ground station can connect either:

- Over a USB serial connection (original mode), or
- Over Wi-Fi, by polling the ESP32 `/data` HTTP endpoint using JSON (updated mode with dual-connection support).

This dual-connection approach allows easy local debugging (serial) and convenient remote monitoring (Wi-Fi) without changing the firmware.

Chapter 3

Hardware: Sensors and Modules Used

This chapter lists the actual sensors and modules used in the SHAPE-SAT implementation and briefly describes their function and role. The descriptions are aligned with the pin assignments and code in `Maincpu.ino`, `SensorTest.ino`, and `SENSOR_CONNECTIONS.md`.

3.1 ESP32 Main Controller

The ESP32 development board serves as the main microcontroller:

- Dual-core 32-bit MCU with integrated Wi-Fi and Bluetooth.
- Provides GPIO, ADC, I^2C, SPI, UART, PWM for connecting all peripherals.
- Hosts the main SHAPE-SAT firmware in `Maincpu.ino`.

It is responsible for:

- Initializing all sensors and the I^2C multiplexer.
- Running self-test and calibration sequences.
- Managing operating modes and mission statistics.
- Serving the web dashboard and JSON API over Wi-Fi.

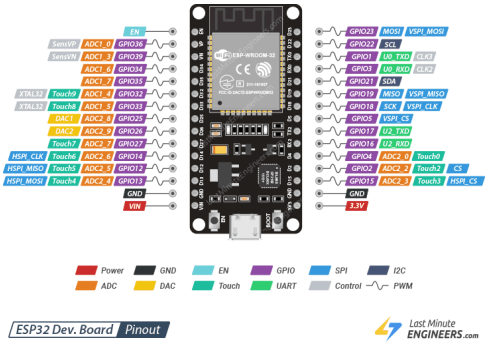


Figure 3.1: ESP32 development board used as the main SHAPE-SAT controller.

3.2 PCA9548A I^{2}C Multiplexer

The PCA9548A module is used to connect multiple I^{2}C devices that may have overlapping addresses. Different groups of devices are placed on different channels (0–7), and the ESP32 selects a channel before accessing the corresponding devices.

In the current design:

- Channel 0: BH1750 light sensor.
- Channel 1: BMP280 pressure/temperature sensor.
- Channel 2: MPU6050 IMU.
- Channel 3: DS3231 RTC (HW-111).
- Channel 4: INA219 current/voltage sensor.
- Remaining channels can be used for OLED or additional expansions.

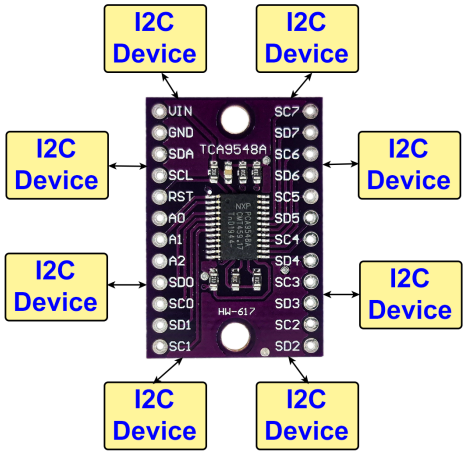


Figure 3.2: PCA9548A I^{2}C multiplexer module used to fan out the ESP32 I^{2}C bus.

3.3 BH1750 Light Sensor

The BH1750 (GY-302) module provides digital ambient light measurements (in lux) over I^{2}C. In SHAPE-SAT it is used to:

- Measure environmental light level for context.
- Detect basic day/night conditions.
- Demonstrate I^{2}C sensor integration through the multiplexer.

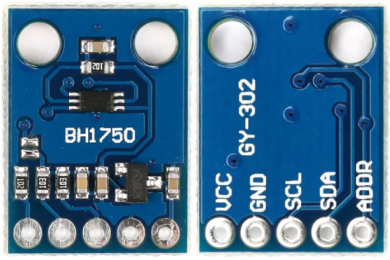


Figure 3.3: BH1750 (GY-302) ambient light sensor module.

3.4 BMP280 Pressure and Temperature Sensor

The BMP280 provides barometric pressure and temperature readings. In the firmware:

- Pressure is converted into altitude using a configurable reference pressure.
- Temperature and pressure are checked for validity and used in mission statistics (min/max).
- The altitude calibration routine computes a more accurate sea-level pressure reference.

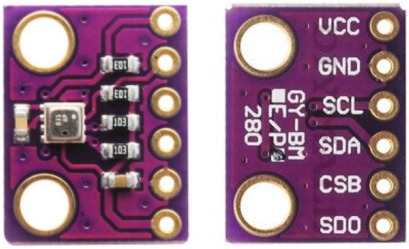


Figure 3.4: BMP280 barometric pressure and temperature sensor module.

3.5 MPU6050 Inertial Measurement Unit

The MPU6050 sensor integrates a 3-axis accelerometer and 3-axis gyroscope. The ESP32 reads acceleration and angular velocity in satellite mission mode and computes:

- Instantaneous acceleration magnitude.
- Maximum acceleration experienced during the mission.

The ground station software uses the accelerometer and gyroscope readings to display a simple 3D visualization of SHAPE-SAT's orientation as a virtual cube.

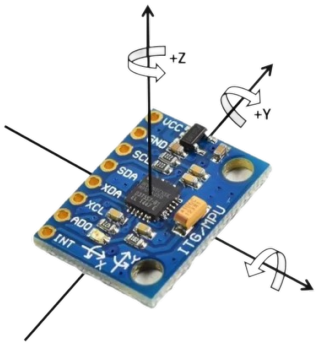


Figure 3.5: MPU6050 6-axis IMU module used for motion and attitude sensing.

3.6 DS3231 RTC (HW-111 Module)

The HW-111 RTC module (DS3231-based) provides:

- Stable date and time across resets.
- Timestamps for telemetry and mission statistics.

The main firmware validates the RTC by checking that the date is in a valid range (2000–2100) and warns if the time appears to be the default (not yet set).

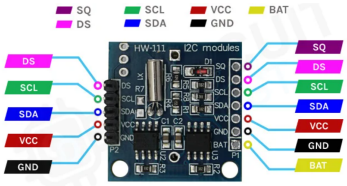


Figure 3.6: HW-111 DS3231 real-time clock module providing accurate timekeeping.

3.7 INA219 Current and Voltage Sensor

The INA219 monitors:

- Bus voltage feeding the ESP32 and sensor stack.
- Current through a shunt resistor.
- Instantaneous power consumption.

These values are:

- Displayed in the serial telemetry output.

- Shown on the Wi-Fi dashboard with an *OK/FAIL* indicator.
- Used in the ground station to plot power-related graphs.

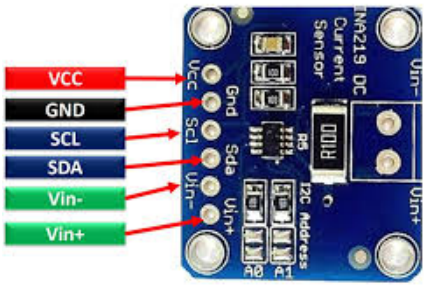


Figure 3.7: INA219 current and voltage monitoring module in the SHAPE-SAT power rail.

3.8 DHT11 Temperature and Humidity Sensor

The DHT11 is wired to GPIO 4 and read using the Adafruit DHT library. Completed integration steps include:

- Periodic reading every 2 seconds (as required by DHT11 timing).
- Validity checks on temperature and humidity ranges.
- Fault detection flag `dht11OK`.
- Display of DHT temperature and humidity on serial telemetry, Wi-Fi dashboard, and ground station.

This provides an independent measurement channel in addition to the BMP280 temperature sensor.

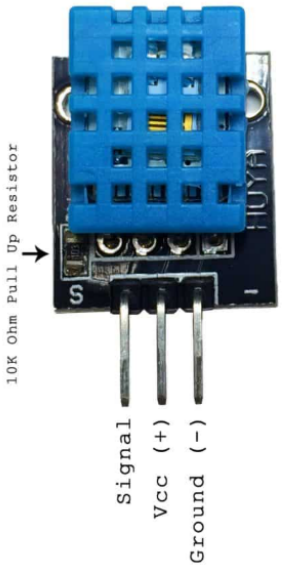


Figure 3.8: DHT11 temperature and humidity sensor used as an additional environmental channel.

3.9 CJMCU-GUVA-S12SD UV Sensor

The GUVA-S12SD UV sensor is connected to an ESP32 ADC pin (GPIO 39 in the current design). The firmware:

- Reads the ADC value and converts it to voltage using the 12-bit resolution and 3.3 V reference.
- Converts voltage into a UV index using an approximate linear relationship (UV index $\approx V \times 11$).
- Caps the UV index at 11 to match the typical scale.
- Reports UV voltage and derived UV index to the serial console, Wi-Fi dashboard, and ground station.

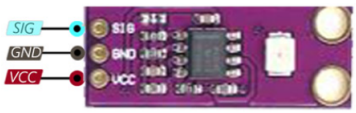


Figure 3.9: CJMCU-GUVA-S12SD UV sensor module for estimating UV index.

3.10 MQ-2 Gas Sensor

The MQ-2 is a gas and smoke sensor module based on a heated SnO_2 (tin dioxide) sensing element whose resistance changes in the presence of combustible gases such as LPG, propane, methane, hydrogen and smoke. The breakout board provides an analog output that can be connected to an ESP32 ADC pin.

In the SHAPE-SAT prototype the MQ-2 is used as a general-purpose *air-quality and gas-leak indicator*. The ESP32:

- Samples the analog voltage from the MQ-2 over an ADC channel.
- Applies basic scaling and filtering to obtain a relative gas concentration level.
- Reports this level in the serial telemetry stream and, when enabled, on the Wi-Fi dashboard and ground station.

Because MQ-series sensors require an initial burn-in and careful calibration for absolute measurements, the current implementation focuses on relative trends (clean vs. polluted / leaky environments) rather than exact ppm values.

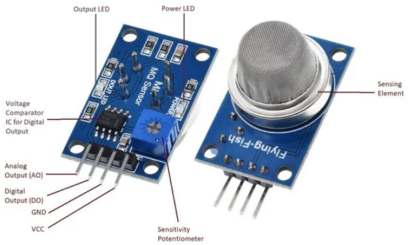


Figure 3.10: MQ-2 gas and smoke sensor module used as a relative air-quality indicator.

3.11 Solar Panels, CN3791 Charger, MT3608 Boost Converter

Two solar panels feed a CN3791-based charger, which:

- Charges a Li-ion/LiPo battery bank from the panel output.
- Implements CC/CV charging and basic protection features.

An MT3608 DC–DC boost module:

- Raises the battery voltage to a constant level suitable for the ESP32 and sensors.
- Is monitored indirectly by the INA219 module.

Together, these components provide a fully autonomous, solar-powered platform for SHAPE-SAT.

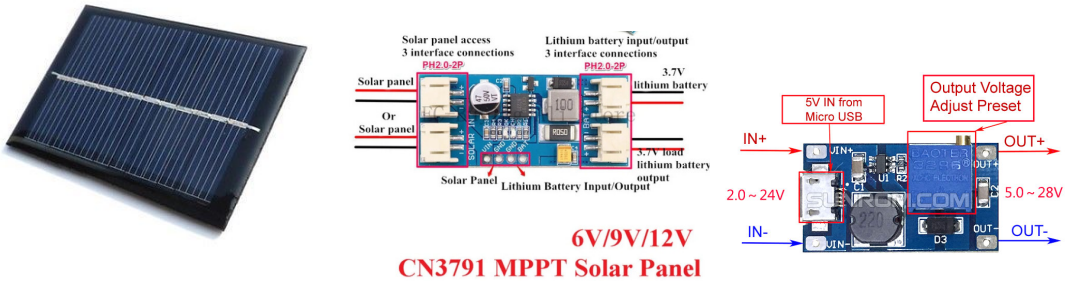


Figure 3.11: Solar panels, CN3791 solar charging module, and MT3608 boost converter forming the SHAPE-SAT power subsystem.

3.12 ESP32-CAM Camera Module (Planned Integration)

An ESP32-CAM module is reserved for future integration into the SHAPE-SAT platform as an imaging payload. The ESP32-CAM combines an ESP32 with an OV2640 camera sensor and an SD card slot, allowing it to capture JPEG images or short video streams.

In the planned configuration:

- The ESP32-CAM will operate as a companion node triggered by the main ESP32 controller.
- Captured images can be stored locally on the ESP32-CAM SD card and/or transmitted over Wi-Fi when bandwidth and power budgets permit.
- Images taken "from above" can be time-correlated with SHAPE-SAT sensor data to provide visual context for environmental measurements.

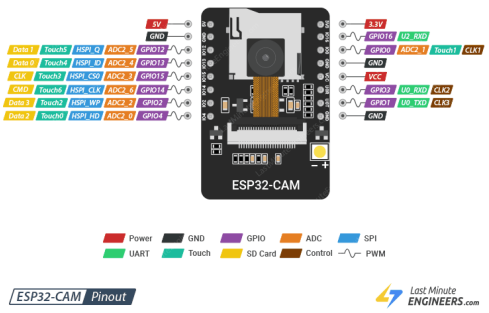


Figure 3.12: ESP32-CAM module planned as a future imaging payload for SHAPE-SAT.

3.13 Buzzer and RGB LED

An active buzzer on GPIO 25 and an RGB LED (red, green, blue channels) provide immediate status feedback:

- Red LED and buzzer pulses to indicate faults or abort conditions.
- Blue LED to indicate general system activity.
- Green + blue LEDs to indicate satellite mission mode is active.
- Short buzzer beeps as user feedback on commands such as *arm* and *calibrate*.

Firmware commands over serial or HTTP can enable or disable the buzzer dynamically.

3.14 NEO-6M GPS Module (Dedicated GPS Setup Sketch)

The NEO-6M GPS module is used together with a dedicated sketch `GPSSetup.ino`:

- GPS TX is connected to ESP32 RX2 (GPIO 16), GPS RX to TX2 (GPIO 17).
- The ESP32 reads NMEA sentences at 9600 baud and extracts latitude, longitude, altitude, satellite count, and fix status.
- The firmware starts a Wi-Fi Access Point (`GPS-Setup`) and a small web server.
- A phone or laptop can connect directly to the ESP32 at `http://192.168.4.1` to view live GPS status.

This sketch was used to verify GPS wiring, antenna placement, and NMEA parsing independently from the main SHAPE-SAT firmware.

Chapter 4

Firmware Features and Operating Modes

The main SHAPE-SAT firmware is implemented in `Maincpu.ino`. It provides a structured set of operating modes, self-test and calibration procedures, and telemetry outputs.

4.1 System Modes

The firmware defines four logical modes:

Ground mode Default safe mode after boot. Sensors are read and telemetry is produced, but mission statistics are not accumulated and safety checks are less strict.

Self-test mode Activated by a short press of Button 2. Runs a comprehensive health check on all connected sensors and prints a human-readable *SENSOR FAULT SUMMARY*.

Calibration mode Activated by a long press of Button 1. Calibrates the MPU6050 (acceleration offset) and BMP280 reference pressure for altitude calculations.

Satellite / mission mode Activated by a 3-second press of both buttons together. Marks the system as "armed" and starts tracking mission statistics such as maximum altitude and maximum acceleration.

Mode transitions are implemented with simple button timing logic and are reflected in both serial telemetry and the Wi-Fi dashboard.

4.2 Self-Test Implementation

The self-test routine:

- Selects each PCA9548A channel in turn.
- Tries to initialize and read from BH1750, BMP280, MPU6050, RTC, INA219, and DHT11.

- Checks that each reading lies within realistic limits.
- Sets individual flags (`bh1750OK`, `bmp280OK`, etc.) and the global `sensorFault` flag.
- Prints a detailed summary indicating which sensors passed and which failed.
- Uses LEDs and buzzer to give quick visual and audible feedback.

The same status flags are used by the web dashboard and ground station to color-code sensor health indicators.

4.3 Calibration Implementation

The calibration procedure consists of:

1. A visual and audible indication (blinking LEDs and buzzer) for several seconds to signal that calibration is in progress and the board should remain still.
2. **MPU6050 calibration:** multiple readings of acceleration on the Z-axis are averaged, subtracting 9.81 m/s^2 to compute an offset. Only physically reasonable readings are included.
3. **Altitude calibration:** multiple BMP280 pressure readings are averaged to compute a local reference pressure for altitude calculations.
4. Setting the `calibrationComplete` flag if both sub-calibrations succeed.

The firmware stores calibration offsets in RAM for use during mission mode.

4.4 Telemetry and JSON API

Telemetry is generated every second and sent to the serial console in a structured format with blocks for:

- System status and time.
- Environmental sensors.
- IMU readings.
- Power monitor values.
- Additional sensors (DHT11 and UV).
- Sensor fault summary.

- Mission statistics when armed.

In addition, a JSON API endpoint `/data` exposes the same fields over HTTP:

- Includes scalar values such as temperature, pressure, altitude, acceleration, bus voltage, current, and power.
- Exposes boolean flags for each sensor's health and overall fault status.
- Provides mission statistics like maximum altitude and maximum acceleration when available.

This JSON endpoint is used by the ground station when operating in Wi-Fi mode.

4.5 Key Algorithms, Mathematical Models, and Code Snippets

Several core computations in `Maincpu.ino` implement the "intelligence" of SHAPE-SAT using well-defined mathematical relations. In this section, the main formulas are summarised and linked directly to representative code fragments.

Altitude Estimation from BMP280

Atmospheric altitude is derived from the barometric pressure reading using the standard barometric formula with a configurable reference pressure P_0:

$$h = 44330 \left(1 - \left(\frac{P}{P_0}\right)^{0.1903}\right)$$

where:

- h is altitude in metres,
- P is the measured pressure in hPa,
- P_0 is the reference sea-level pressure in hPa obtained during calibration.

During altitude calibration, multiple BMP280 samples are averaged to obtain an accurate P_0. Listing 4.1 shows how this formula is applied in code when updating the BMP280 readings.

Acceleration Magnitude from IMU

From the MPU6050 accelerometer readings (a_x, a_y, a_z), the firmware computes the total acceleration magnitude:

$$a_{\text{mag}} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

This value is used to:

- detect abnormal dynamics beyond a safety threshold,
- track the maximum acceleration experienced during the mission.

Prior to use, the Z-axis is calibrated so that static gravity is removed and small offsets are compensated; the resulting magnitude is then used in the safety checks illustrated in Listing 4.2.

Power Monitoring with INA219

The INA219 provides bus voltage V_{bus}, shunt voltage V_{shunt} across a known shunt resistor R_{shunt}, and internally computed current and power. Conceptually:

$$I = \frac{V_{\text{shunt}}}{R_{\text{shunt}}}, \qquad P = V_{\text{bus}} \times I$$

The firmware logs V_{bus}, I and P to characterise energy usage and solar-charging performance over time, as implemented in Listing 4.2.

UV Index Computation

The CJMCU-GUVA-S12SD UV sensor produces an analog voltage V_{UV} that is read by the ESP32 ADC and normalised to $[0, 3.3]$ V. An approximate linear mapping is used to estimate the UV index:

$$\text{UV Index} \approx \alpha\, V_{\text{UV}}, \qquad \alpha \approx 11$$

with the result clipped to the interval $[0, 11]$. This provides a simple but useful indication of UV exposure level that is then reported in the telemetry stream (see Listing 4.3).

Mission Time and Statistics

Mission time is computed from the difference between the current millisecond counter and the millisecond value at arming:

$$t_{\text{mission}} = \frac{t_{\text{now}} - t_{\text{arm}}}{1000}$$

and is formatted into hours, minutes, and seconds for display. During mission mode, the firmware continuously updates:

- maximum and minimum altitude,
- maximum and minimum temperature,
- maximum acceleration magnitude.

These statistics provide a compact numerical summary of the entire mission profile and are printed in the structured telemetry output (Listing 4.3).

Representative Implementation Snippets

To complement the above mathematical description, this subsection presents selected excerpts from `Maincpu.ino` that show how the algorithms are implemented in practice.

I²C Multiplexing and Sensor Updates. Listing 4.1 shows the helper used to select a PCA9548A channel and part of the periodic sensor update logic using this helper.

```
// PCA9548A I2C multiplexer helper
#define PCA_ADDR 0x70

void selectChannel(uint8_t ch) {
  Wire.beginTransmission(PCA_ADDR);
  Wire.write(1 << ch);
  Wire.endTransmission();
  delay(2);  // Small delay for channel switch
}

void updateSensors() {
  // Light sensor on channel 0
  if (millis() - lastLight > 500) {
    lastLight = millis();
    selectChannel(0);
    float newLux = lightMeter.readLightLevel();
    if (newLux >= 0 && newLux < 100000) {
      luxValue = newLux;
    }
  }

  // BMP280 on channel 1
  if (millis() - lastBMP > 200) {
    lastBMP = millis();
    selectChannel(1);
    float newTemp  = bmp.readTemperature();
```

```
    float newPress = bmp.readPressure() / 100.0;  // hPa
    if (newTemp > -50 && newTemp < 100 &&
        newPress > 300 && newPress < 1200) {
      bmpTemp     = newTemp;
      bmpPressure = newPress;
      altitude = 44330.0 * (1 - pow(bmpPressure / referencePressure,
          0.1903));
      validateSensorData();
    }
  }
}
```

Listing 4.1: PCA9548A channel selection and periodic sensor updates in `Maincpu.ino`.

Power Computation and Safety Checks. Listing 4.2 illustrates how INA219 measurements are used to compute power and how acceleration and altitude limits are enforced in mission mode.

```
// INA219 power monitor on channel 4
if (millis() - lastINA219 > 500) {
  lastINA219 = millis();
  selectChannel(4);
  if (ina219.begin()) {
    float newVoltage = ina219.getBusVoltage_V();
    float newCurrent = ina219.getCurrent_mA();
    float newPower   = ina219.getPower_mW();

    if (!isnan(newVoltage) && !isnan(newCurrent)) {
      busVoltage = newVoltage;
      current    = newCurrent;
      power      = newPower;
    }
  }
}

// Periodic safety checks in mission mode
if (satelliteArmed && millis() - lastHealthCheck >
   HEALTH_CHECK_INTERVAL) {
  lastHealthCheck = millis();
  checkSensorHealth();

  if (altitude > MAX_ALTITUDE) {
```

```
    Serial.println("[SAFETY] WARNING: Altitude exceeds safety limit!
        ");
  }

  float accMag = sqrt(accX*accX + accY*accY + accZ*accZ);
  if (accMag > MAX_ACCELERATION) {
    Serial.println("[SAFETY] WARNING: Acceleration exceeds safety
        limit!");
  }
}
```

Listing 4.2: Power monitoring and safety checks for altitude and acceleration.

Structured Telemetry Output. Finally, Listing 4.3 extracts part of the main telemetry routine that formats sensor data and mission statistics into a human-readable report for the serial console.

```
void telemetry() {
  if (millis() - lastPrint < TELEMETRY_INTERVAL) return;
  lastPrint = millis();
  telemetryCount++;

  selectChannel(3);
  DateTime now = rtc.now();

  unsigned long missionTime = satelliteArmed
                              ? (millis() - missionStartTime) / 1000
                              : 0;

  Serial.println("\n========== SATELLITE TELEMETRY ==========");
  Serial.printf("MODE         : %s\n",
                satelliteArmed ? "SATELLITE" : "GROUND");
  Serial.printf("UTC TIME     : %04d-%02d-%02d %02d:%02d:%02d\n",
                now.year(), now.month(), now.day(),
                now.hour(), now.minute(), now.second());

  Serial.printf("\n--- ENVIRONMENTAL SENSORS ---\n");
  Serial.printf("LIGHT        : %.2f lx\n", luxValue);
  Serial.printf("TEMP         : %.2f C\n", bmpTemp);
  Serial.printf("PRESSURE     : %.2f hPa\n", bmpPressure);
  Serial.printf("ALTITUDE     : %.2f m\n", altitude);

```

```
26   Serial.printf("\n--- POWER MONITOR (INA219) ---\n");
27   Serial.printf("BUS VOLTAGE   : %.3f V\n", busVoltage);
28   Serial.printf("CURRENT      : %.2f mA\n", current);
29   Serial.printf("POWER        : %.2f mW\n", power);
30 }
```

Listing 4.3: Excerpt from the structured telemetry function.

These examples demonstrate that the SHAPE-SAT firmware is not a simple "black box": it is built on explicit mathematical models and carefully structured C++ code that implements sensing, safety logic, and rich telemetry.

4.6 Wi-Fi Dashboard

The firmware starts the ESP32 in station mode and connects to a user-configured Wi-Fi network. If successful, it launches an HTTP server with:

- Root page (`/`) providing a mobile-friendly HTML dashboard with live sensor values and health indicators.
- `/data` endpoint returning JSON telemetry.
- `/buzzer` endpoint for remote buzzer control.

The dashboard is styled with a dark background and green text and refreshes automatically every two seconds. Users can open the dashboard on any phone or computer on the same Wi-Fi network by typing the IP address displayed in the serial monitor.

Chapter 5

Ground Station Software

The `ground_station.py` application was extended to support both serial and Wi-Fi connections.

5.1 Connection Modes

On startup, the user can choose:

- **Serial Port**: read raw telemetry text over USB.
- **IP Address (Wi-Fi)**: poll the ESP32 `/data` endpoint over HTTP.

The connection status is clearly shown as:

- `DISCONNECTED` (red).
- `CONNECTED (Serial)` (green).
- `CONNECTED (WiFi: <IP>)` (green).

5.2 User Interface and Visualizations

The ground station GUI provides:

- Numerical displays for all key telemetry values.
- Time-series plots (Matplotlib) of altitude, temperature, acceleration, and power metrics.
- A sensor health panel with green/red indicators for each sensor.
- A 3D orientation view that uses IMU data to render a virtual cube representing SHAPE-SAT's attitude.

- A login window to restrict access.

The additional DHT11 and UV sensor values are fully integrated:

- Stored in the internal telemetry structure.
- Displayed in a dedicated “Additional Sensors” panel.
- Parsed from both serial text and JSON IP data.

Chapter 6

Diagnostic Sketches and Tools

6.1 `SensorTest.ino`

The `SensorTest.ino` sketch is a dedicated sensor connection test program. Its purpose is to verify wiring and module health before running the main firmware.

Key features:

- Tests PCA9548A connectivity and all channels.
- Individually tests BH1750, BMP280, MPU6050, DS3231 RTC, INA219, DHT11, and UV sensor.
- Prints a detailed *TEST SUMMARY* with PASS/FAIL status for each sensor.
- Provides continuous monitoring mode that prints a compact line with the latest readings from each sensor every few seconds.

This sketch is especially useful during initial hardware bring-up or when debugging wiring problems.

6.2 `GPSSetup.ino`

The `GPSSetup.ino` sketch is a simple GPS viewer that:

- Reads NEO-6M NMEA sentences over UART2.
- Parses RMC and GGA sentences for time, fix status, satellite count, position, and altitude.
- Starts a Wi-Fi Access Point called `GPS-Setup`.
- Hosts a small HTML page showing whether GPS data is being received, whether a fix is present, the number of satellites, and the decoded position.

This allows quick testing of the GPS module using only a phone and the ESP32, even without the full SHAPE-SAT firmware.

Chapter 7

Results and Achievements

The main achievements of the SHAPE-SAT implementation can be summarized as follows:

7.1 Multi-Sensor Integration

- Successful integration of BH1750, BMP280, MPU6050, DS3231 RTC, INA219, DHT11, UV sensor, and MQ-2 gas sensor through a PCA9548A multiplexer and direct GPIO/ADC.
- Robust self-test routines that detect missing or misbehaving sensors and clearly report faults.
- Additional sensors (DHT11 and UV) fully integrated into the telemetry stream, web dashboard, and ground station.

7.2 Operational Modes and Safety

- Clear separation of ground, self-test, calibration, and satellite mission modes.
- Safety checks in mission mode for altitude and acceleration limits.
- Mission statistics including maximum altitude, temperature extremes, and maximum acceleration.

7.3 Power and Telemetry

- Solar-powered operation using CN3791 charger and MT3608 boost converter.
- Real-time bus voltage, current, and power measurements using INA219.
- Reliable serial telemetry and JSON telemetry over Wi-Fi.

7.4 Ground Station and Phone Dashboard

- Desktop ground station extended to support both serial and Wi-Fi connections.
- Real-time plots and 3D attitude visualization from IMU data.
- Phone-friendly Wi-Fi dashboard showing all key sensor values and sensor health indicators.
- Wi-Fi troubleshooting guide and improved reconnection handling to make phone access more reliable.

7.5 GPS and Diagnostic Tools

- GPS setup sketch verifying NEO-6M performance and antenna placement using a phone-based web page.
- Dedicated sensor test sketch for quickly validating wiring and module health before running the main firmware.

Overall, the project moved from concept to a working prototype with real hardware, sensors, power management, and multi-platform visualization.

Chapter 8

Circuit Schematic

The complete wiring of the SHAPE-SAT prototype is captured in the circuit schematic shown in Figure 8.1, stored as `sat.pdf` in the project repository. This schematic brings together the power subsystem, ESP32 controller, PCA9548A I^{2}C multiplexer, and all connected sensors and indicators on a single page.

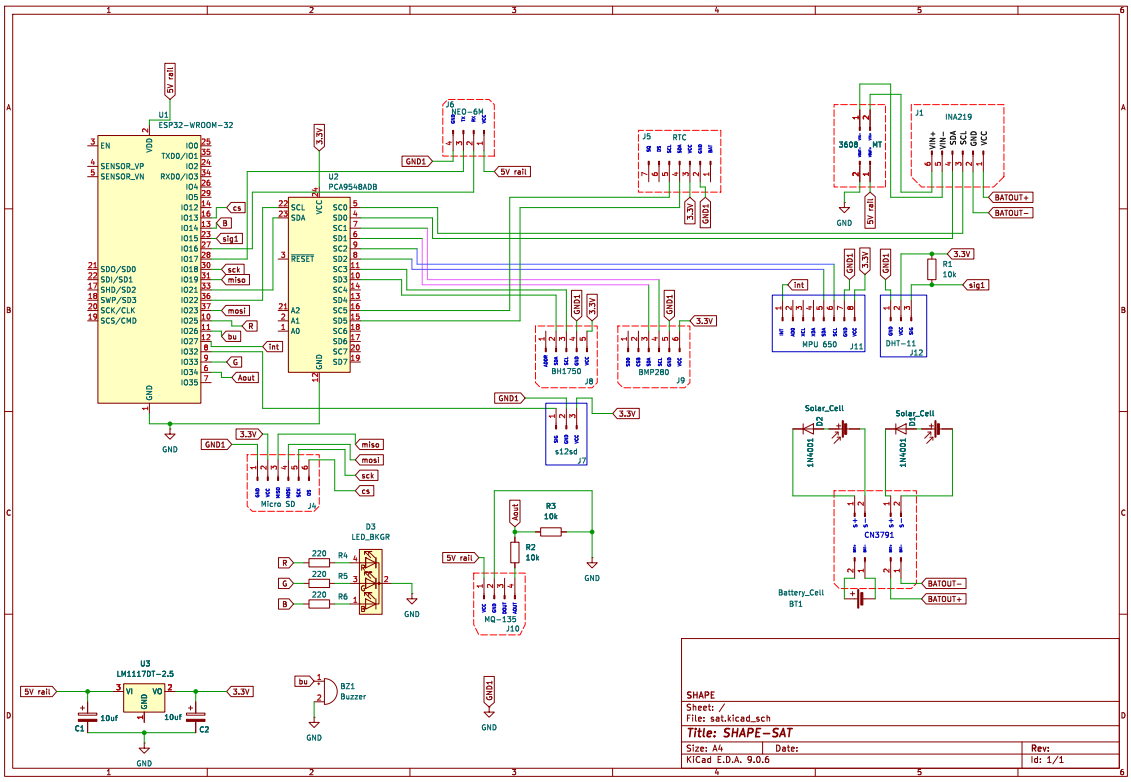


Figure 8.1: Full SHAPE-SAT circuit schematic (`sat.pdf`) showing solar power, ESP32, PCA9548A, and all sensors/modules.

At the top of the schematic, two solar panels feed the CN3791 solar charging module, which in turn charges a Li-ion/LiPo battery pack. The battery output is connected to the MT3608 DC–

DC boost converter, providing a regulated supply rail for the ESP32 and the sensor stack. An INA219 current and voltage sensor is inserted in series with the main supply, allowing real-time monitoring of bus voltage, current, and power.

The ESP32 module is shown at the centre of the schematic, with:

- I^2C lines (SDA on GPIO 21 and SCL on GPIO 22) routed to the PCA9548A multiplexer.
- Digital pins reserved for the buzzer and RGB LED indicators.
- GPIO 4 assigned to the DHT11 temperature and humidity sensor.
- ADC input pins connected to the analog output of the CJMCU-GUVA-S12SD UV sensor and to the MQ-2 gas sensor (and/or MQ-135, depending on configuration).
- UART2 pins (GPIO 16 and GPIO 17) reserved for the NEO-6M GPS module in the dedicated GPS sketch.

Downstream from the PCA9548A, individual channels connect to the I^2C sensors and modules:

- Channel 0: BH1750 light sensor (GY-302).
- Channel 1: BMP280 pressure and temperature sensor.
- Channel 2: MPU6050/650 IMU.
- Channel 3: HW-111 DS3231 RTC module.
- Channel 4: INA219 power monitor.
- Additional channels available for the OLED display or future I^2C expansions.

The schematic also shows the micro SD card adapter connected over SPI to the ESP32, as well as headers for the ESP32-CAM module and NRF24L01 transceivers when used. Together, this diagram confirms how the logical architecture described in earlier chapters is realised as an actual hardware wiring layout for the SHAPE-SAT prototype.

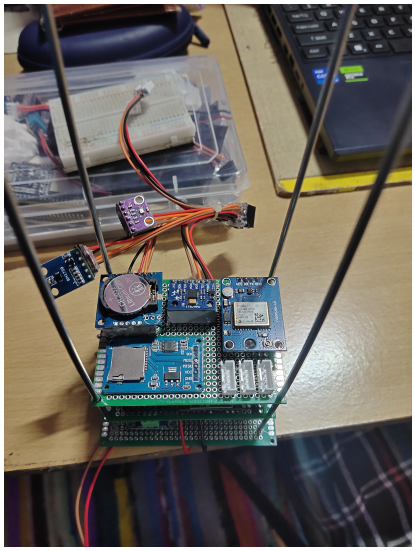
Chapter 9

Hardware and Software Photos

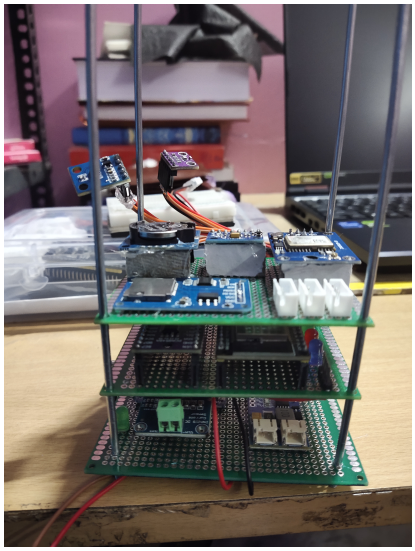
This chapter shows representative images from the assembled hardware and the main software interfaces. All images are sourced from the `photo` folder of the project.

9.1 Assembled Hardware

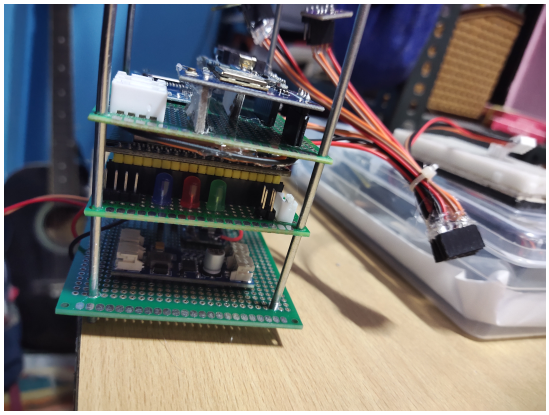
Figure 9.1: Initial assembled SHAPE-SAT hardware stack and closer view of the sensor and power management modules around the ESP32.



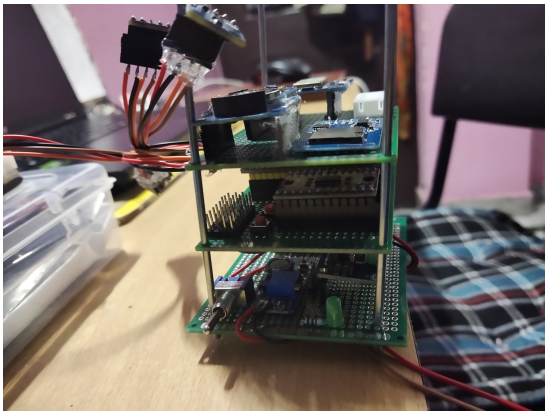
Top view



Front view



Left-side view



Right-side view

Figure 9.2: Final SHAPE-SAT assembly: top and front views (top row) and left- and right-side views (bottom row) after complete integration and wiring.

9.2 ESP32 Web Dashboard Modes

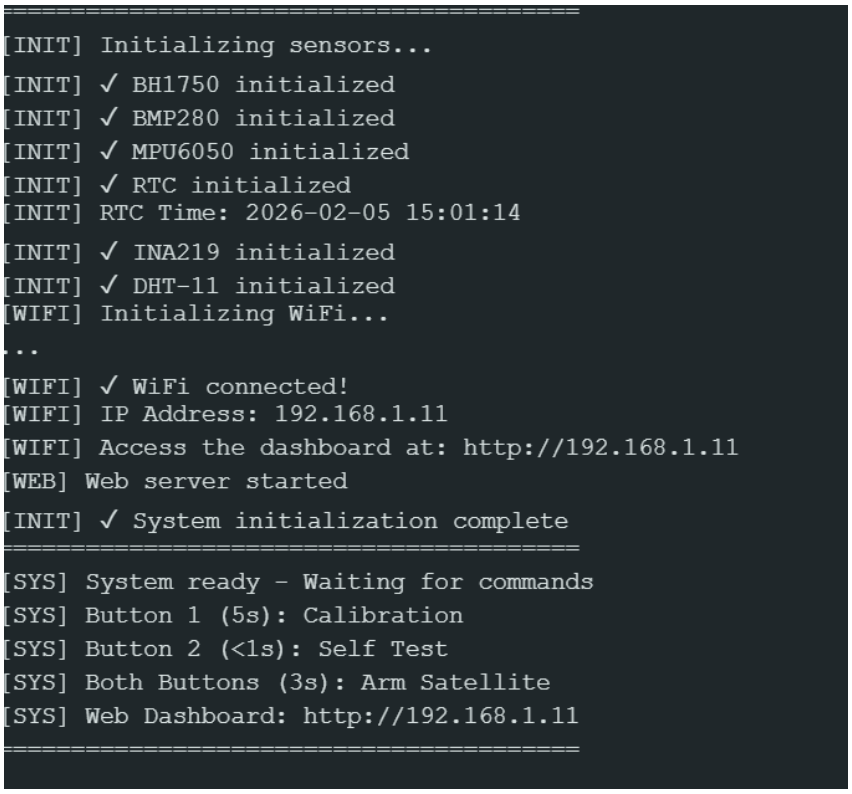


Figure 9.3: Initialization system screen showing boot messages and early sensor status on the web dashboard.

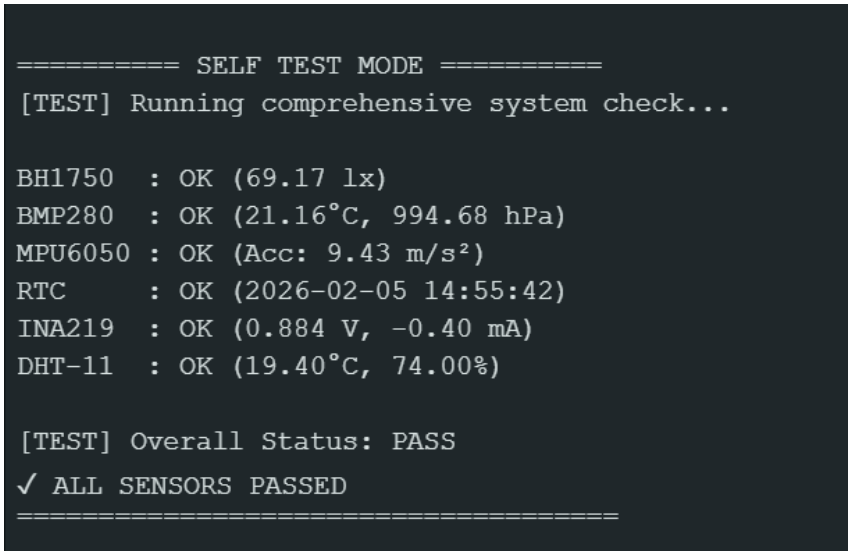


Figure 9.4: Self-test mode screen on the phone, summarizing the status of all sensors after running the comprehensive health check.

```
========== CALIBRATION MODE ==========
[CAL] Starting full system calibration...
[CAL] Starting MPU6050 calibration...
[CAL] MPU6050 calibrated: Offset = -0.4449 m/s²
[CAL] Starting altitude calibration...
[CAL] Altitude calibrated: Reference = 994.67 hPa
[CAL] ✓ Calibration COMPLETE
======================================
```

Figure 9.5: Calibration mode screen indicating that IMU and altitude calibration are in progress and showing final offsets.

```
========== ARMING SEQUENCE ==========
[ARM] Running pre-flight checks...
[ARM] ✓ Satellite ARMED - Mission mode ACTIVE
======================================


========== SATELLITE TELEMETRY ==========
MODE         : SATELLITE
SYSTEM MODE  : SATELLITE
UTC TIME     : 2026-02-05 15:06:19
UPTIME       : 51 s
MISSION TIME : 00:00:02

--- ENVIRONMENTAL SENSORS ---
LIGHT        : 65.83 lx (OK)
TEMP         : 21.31 °C
PRESSURE     : 994.58 hPa
ALTITUDE     : 0.34 m (OK)

--- IMU SENSORS ---
ACC X        : 1.044 m/s²
ACC Y        : 0.809 m/s²
ACC Z        : 9.784 m/s² (OK)
GYRO X       : -0.058 rad/s
GYRO Y       : 0.008 rad/s
GYRO Z       : -0.037 rad/s

--- POWER MONITOR (INA219) ---
BUS VOLTAGE  : 0.888 V (OK)
SHUNT VOLTAGE: -0.040 mV
CURRENT      : -0.40 mA
POWER        : 0.00 mW

--- ADDITIONAL SENSORS ---
DHT-11 TEMP  : 19.80 °C (OK)
DHT-11 HUMID : 70.00 % (OK)

--- SENSOR FAULT SUMMARY ---
✓ ALL SENSORS OK

--- SYSTEM STATUS ---
SENSOR FAULT : NO
CALIBRATED   : YES
TELEMETRY #  : 31
ERROR COUNT  : 0
READ ERRORS  : 0

--- MISSION STATISTICS ---
MAX ALTITUDE : 156.69 m
MIN ALTITUDE : -0.31 m
MAX TEMP     : 21.31 °C
MIN TEMP     : 21.17 °C
MAX ACCEL    : 9.87 m/s²
======================================
[SYS] Calibration blocked - Satellite is ARMED
```

Figure 9.6: Satellite (mission) mode display, where mission statistics such as maximum altitude and acceleration are actively tracked.

9.3 Ground Station User Interface

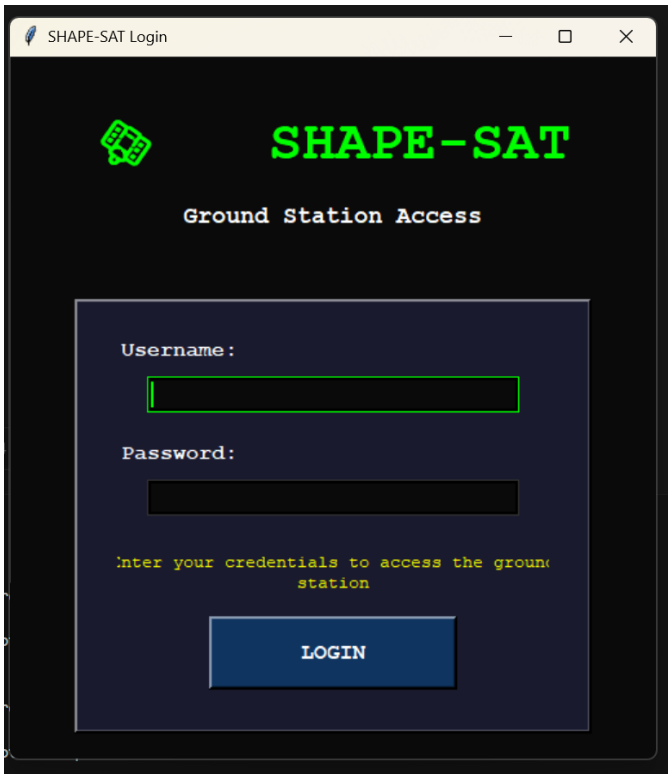


Figure 9.7: Ground station login window used to restrict access to authorized users.

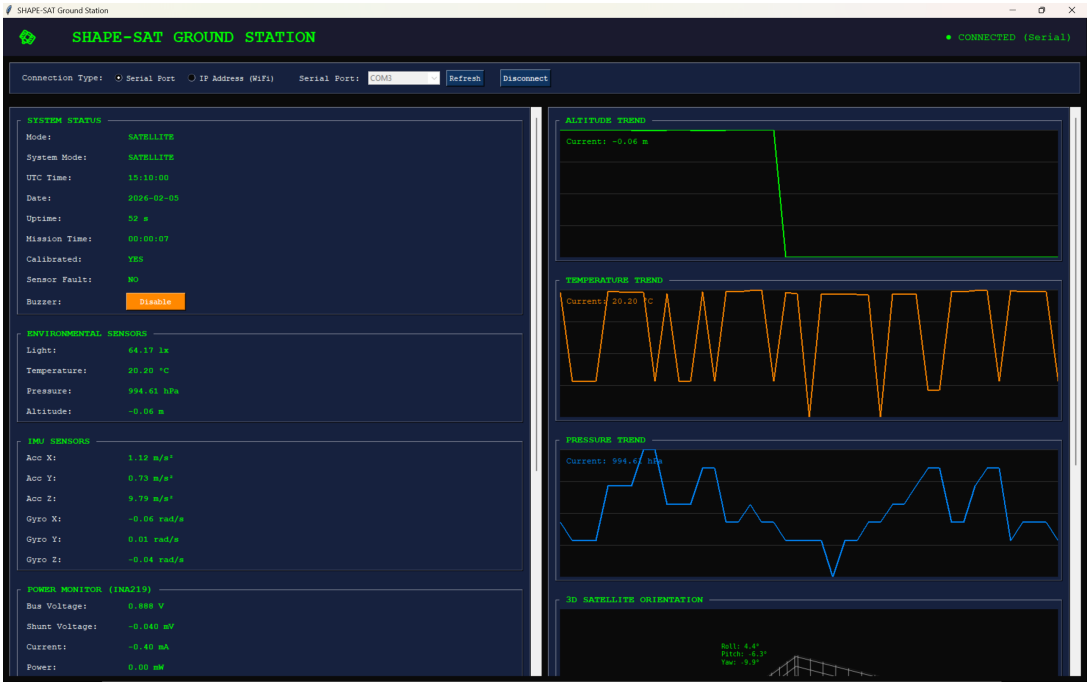


Figure 9.8: Main ground station window showing numerical telemetry, sensor health indicators, and mission status.

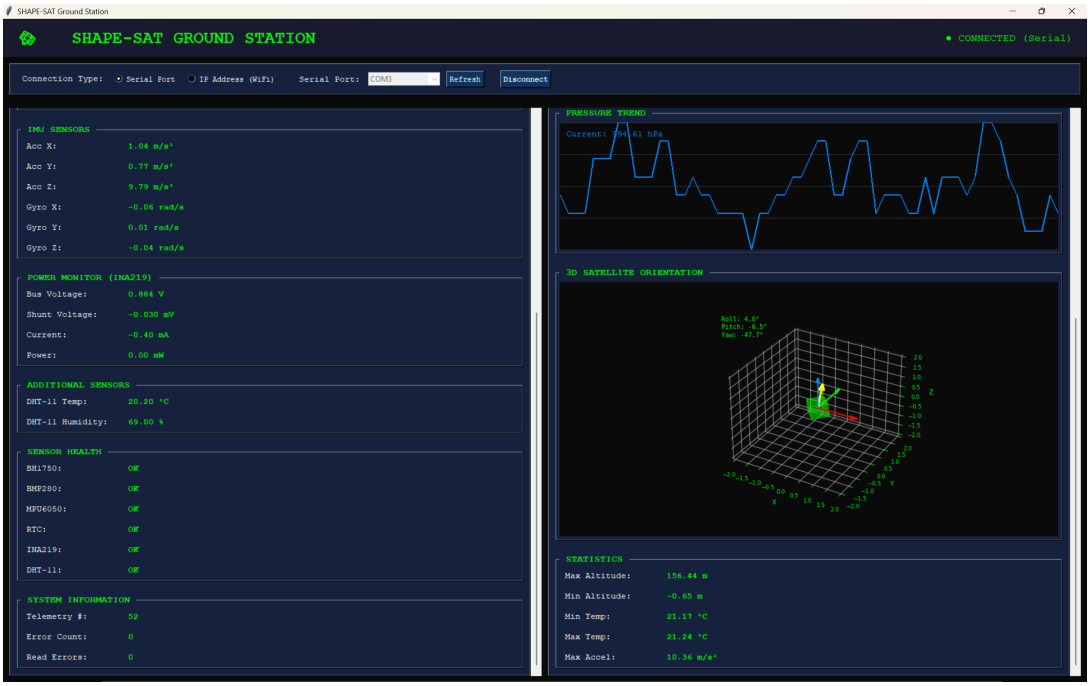


Figure 9.9: Ground station views with plots and 3D attitude visualization driven by IMU data from SHAPE-SAT.

9.4 Mobile Phone Dashboards

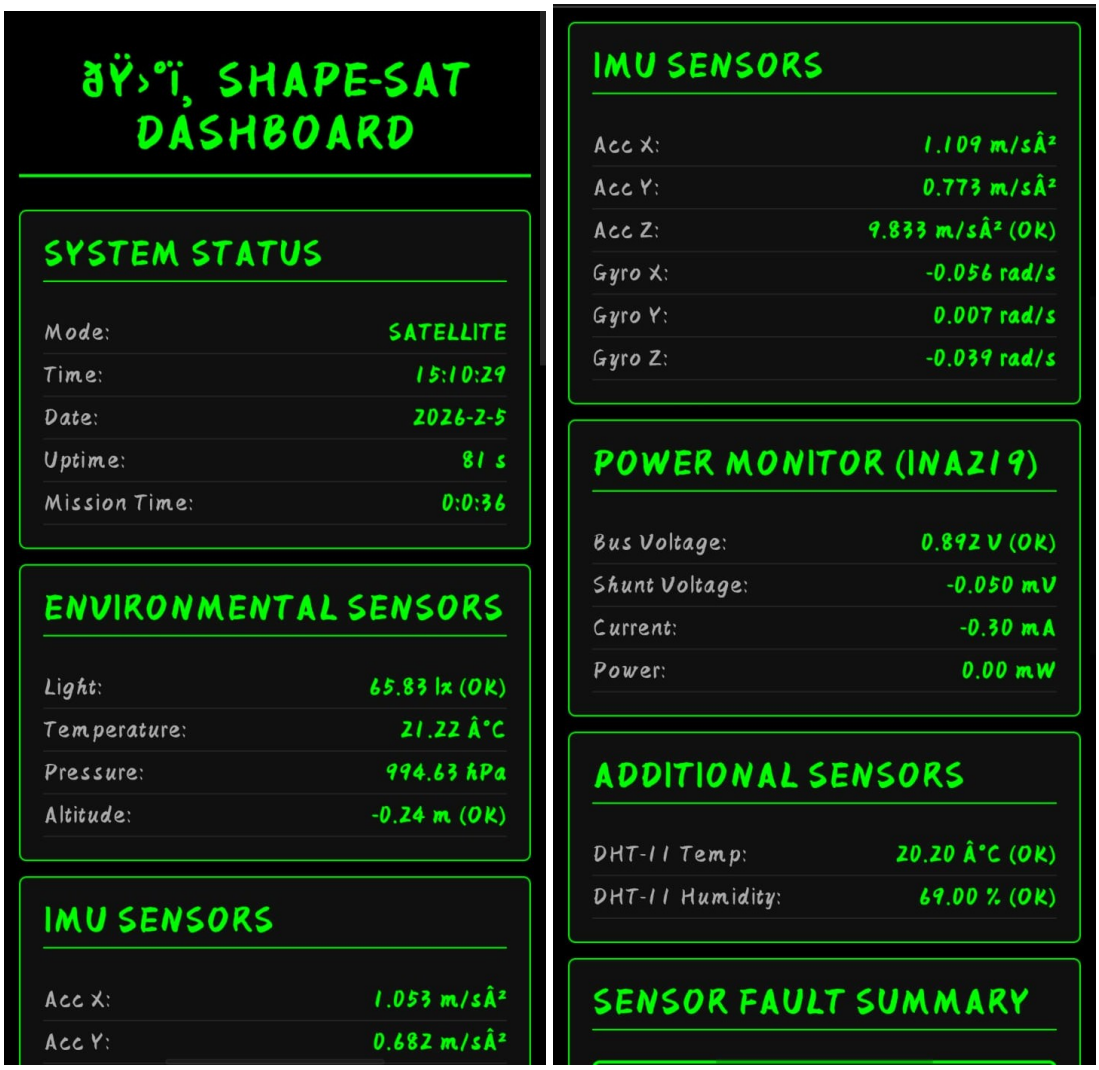


Figure 9.10: Examples of the Wi-Fi dashboard as viewed on a mobile phone, showing sensor values and system status.

Figure 9.11: Additional mobile dashboard screens highlighting sensor health indicators and mission-related values.

Chapter 10

Conclusion

The SHAPE-SAT project successfully delivers a working prototype of a solar-powered, multi-sensor IoT satellite demonstrator based on the ESP32 platform. All major subsystems—sensing, power, communication, and visualization—have been implemented and tested using real hardware, as documented in this report.

The system provides a realistic environment for experimenting with sensor fusion, power management, telemetry, and ground station design. It can be extended further with additional sensors, more advanced power control, and deeper integration with cloud services, but already functions as a complete educational and experimental platform.

During development, modern AI-based coding assistance tools were also used to help debug firmware issues, refine sensor integration, and structure this technical documentation, allowing faster iteration and more reliable code.