DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2022

MiniMIST- A Model Satellite For Teaching STEM Skills

MiniMIST- En modellsatellit för att lära ut STEM-färdigheter

Kocin Sabareeswaran Bama

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Authors

Kocin Sabareeswaran Bama kocin@kth.se
Information and Communication Technology
KTH Royal Institute of Technology

Place for Project

Kista, Sweden

Examiner

Carl-Mikael Zetterling Kista, Sweden
KTH Royal Institute of Technology

Supervisor

Matthias Becker

Kista, Sweden

KTH Royal Institute of Technology

Abstract

Satellites are evolving around the globe, expanding our possibility to explore space and many other applications. This thesis project is a model satellite used for teaching purposes in the course IE120V Electronics and Programming for Space Applications provided at KTH Royal Institute of Technology. The model can be used to teach Science, technology, engineering, and mathematics (STEM) in high school. The satellite is inspired by the MIniature Student saTellite (MIST) project carried out by the KTH Royal Institute of Technology, and hence this project will be called ”MiniMIST.”

The MiniMIST aims to perform functions carried out on practical satellites. The onboard computer performs real-time image capturing, reads temperature from the surroundings, and analyzes power from solar panels and the power supply (batteries). The ground station setup is used to communicate with the satellite (MiniMIST) and receive data to be viewed and analyzed.

The ESP32-Wrover was used as the main computer to run the MiniMIST. It was programmed with MicroPython and C++. Client-server communication using HTTP protocol emulated the communication between the ground station and MiniMIST with Web-Server communication. The ground station is a web page programmed using HTML, CSS, and JavaScript along with the AJAX framework.

The thesis aims to build a working prototype of MiniMIST by dividing it into three sections: Hardware design, onboard computer software, and ground station software.

Keywords

MIST, Satellite, ESP32, MicroPython, C++, AJAX, Photovoltaic cells

Sammanfattning

Satelliter utvecklas runt om i världen, vilket utökar vår möjlighet att utforska rymden och många andra applikationer. Detta examensarbete är en modellsatellit som används för undervisningsändamål i kursen IE120V Elektronik och programmering för rymdtillämpningar som ges vid Kungliga Tekniska Högskolan. Modellen kan användas för att lära ut naturvetenskap, teknik, ingenjörskap och matematik (STEM) på gymnasiet. Satelliten är inspirerad av projektet MIniature Student saTellite (MIST) som genomförts av Kungliga Tekniska Högskolan, och därför kommer detta projekt att kallas ”MiniMIST”.

MiniMIST syftar till att utföra funktioner som utförs på praktiska satelliter. Datorn ombord utför bildtagning i realtid, läser av temperatur från omgivningen och analyserar ström från solpaneler och strömförsörjningen (batterier). Markstationsinställningen används för att kommunicera med satelliten (MiniMIST) och ta emot data som ska ses och analyseras.

ESP32-Wrover användes som huvuddator för att köra MiniMIST. Den programmerades med MicroPython och C++. Klient-serverkommunikation med hjälp av HTTP-protokoll emulerade kommunikationen mellan markstationen och MiniMIST med webbserverkommunikation. Markstationen är en webbsida programmerad med HTML, CSS och JavaScript tillsammans med AJAX-ramverket.

Avhandlingen syftar till att bygga en fungerande prototyp av MiniMIST genom att dela upp den i tre sektioner: Hårdvarudesign, inbyggd datormjukvara och markstationsmjukvara.

Nyckelord

MIST, Satellit, ESP32, MicroPython, C++, AJAX, solceller

Acknowledgements

Words cannot express my heartfelt gratitude to my Examiner, Carl-Mikael Zetterling a.k.a Bellman, who gave me a fantastic opportunity and guided me throughout this project. I couldn't have taken this journey if it wasn't for his invaluable patience and support. I would also like to show my most profound appreciation to my supervisor, Matthias Becker, who generously provided knowledge and expertise to help me finalize my project.

Acronyms

ADC	analog-to-digital converter
API	Application Programming Interface
JSON	JavaScript Object Notation
MIST	Miniature Student Satellite
MPP	Maximum Power Point
PCB	Printed Circuit Board
PV	Photovoltaic
STEM	Science Technology Engineering and Mathematics

Contents

List of Figures

Chapter 1

Introduction

The project in the thesis is built for teaching purposes in the course IE120V Electronics and Programming for Space Applications provided at KTH Royal Institute of Technology. High school teachers interested in learning about electronics and embedded systems are the targeted audience of the course. The course teaches the functions of microcontrollers, different sensors connected to microcontrollers, power supply from batteries and solar panels, and radio communication with antennas. The project is a model satellite used as an example to demonstrate some of the topics mentioned above. It includes programming the ESP32 microcontroller and reading data from sensors and power supply. The model can be used to teach Science Technology Engineering and Mathematics (STEM) in high school. The name of the model satellite is MiniMIST, from the MIST(MIniature Student saTellite) project initiated by the KTH Space Center.[1]

1.1 Background

The project intends to portray an accurate implementation of how a satellite functions. A practical satellite communicates using radio waves to send signals to the antennas on the ground station. The signal from the satellite is captured and processed to acquire the data collected, including scientific data (like images, temperature, and any other observations made), the health of the satellite, and its location (GPS).[2]

The MiniMIST will read satellite data from its remote setting and communicate with the ground station sending the collected data. It will use radio communication in the form of WiFi and client-server communication using the HTTP protocol. The implementation of MiniMIST has three parts.

Hardware design includes setting up the MiniMIST with ESP32-Wrover kit, OV2640 camera, sensors, solar panels and batteries, and other components. From the final schematics,

prototype hardware was made on a Printed Circuit Board (PCB).
The Software for onboard computer involves programming the ESP32-Wrover development board to interact with the hardware devices and communicate the data to the ground station. It can be programmed with a choice of two languages, MicroPython and C++.
Software for the ground station consists of a web page for the users to interact with the MiniMIST, allowing it to display any received data from the model satellite.

1.2 Problem

The main problem addressed in the thesis is if it is possible to use a model satellite built using a microcontroller to teach and demonstrate real-life satellite functionalities as well as teaching basic STEM knowledge.

1.3 Purpose

The project's purpose is to explain the concepts taught in the program IE120V Electronics and Programming for Space Applications. It is an example project that implements some of the concepts taught during the IE120V program, such as Photovoltaic (PV) cells, Battery charging, Image capture, and data transmission. The MiniMIST is also used to explain some of the technical functions of a satellite, using block models and practical demonstration.

1.4 Goal

The final product of the thesis is to implement a working prototype of MiniMIST that satisfies all the requirements. The project is required to have the following:

- A computer connected to a camera, sensors, and other components
- Solar cells which are not necessarily used for powering the satellite
- Battery power
- An onboard computer programmed with C++ and/or MicroPython
- Ground station (laptop or phone), with a web interface to send commands and receive data

The final design for the ground station, onboard computer, and hardware layout for PCB is submitted.

1.5 Delimitation

A ready made WiFi transceiver was used to model the radio transmission in the project. The project's scope was ample to include this aspect, as this can be a separate bachelor's thesis on its own. The course doesn't focus much on radio frequency transmission. The project will not use external antenna to transmit data.

1.6 Outline

The theoretical background chapter explains the concepts required to understand the thesis.
The method chapter explains how different choices were made for different parts of the design.
The Design chapter explains the design of the system, how it functions, and what tools are used in the development?
The Result chapter shows the final design of the prototype.
Finally, the conclusion of the project.

Chapter 2

Background

2.1 Miniature Student Satellite (MIST)

The Miniature Student Satellite (MIST) project [1] is a Swedish 3U-CubeSat designed and produced by students at KTH Space Centre. A **CubeSat** is a miniature satellite that has been used exclusively in low Earth orbit for applications such as remote sensing or communications and is now being used for interplanetary missions like MArCo, which was deployed on a mission flying to Mars[3][4]. The MIST project was initiated in 2014, and the work started on January 28, 2015, supervised and mentored by Dr. Sven Grahn and Prof. Christer Fuglesang. The main purpose of the Student Satellite MIST are:

- Educational: The project is built by students working in small teams for their thesis projects. A new team of about ten students takes part every semester to perform system-level design, integration, testing, and operations
- Experimental: The KTH Student Satellite MIST holds five technical and scientific experiments.

2.2 ESP32 Micro-controller

ESP32 is a low-cost, low-power System on Chip (SOC) microcontroller that can be used for multiple purposes as it comes with multiple peripherals, including WiFi and dual-mode Bluetooth. The ESP32 has dual high-performance cores supporting clock frequency up to 240 MHz and multiple peripherals that can be set to be UART, I2C, ADC, or more by configuring the code. Its WiFi functionality allows the device to connect to an extensive physical range and connects directly to the internet through the router[5].

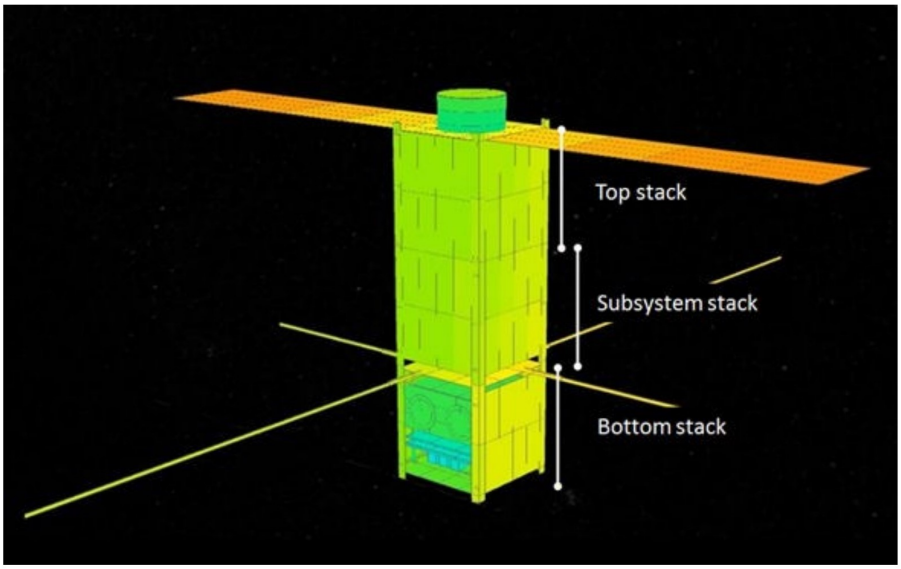


Figure 2.1.1: MIST Project

Figure 2.2.1: Picture of ESP32-Wrover module

ESP32-Wrover module We will use an ESP32-Wrover module with a pre-tuned PCB antenna and an IPEX connector for the antenna. The module is preferred over ESP32 SOC because Espressif has already pre-loaded the low-level device drivers, the wireless protocol stacks for WiFi b, g, n, Bluetooth and BLE, and FreeRTOS as the base OS. The ESP32-Wrover features a 4 MB external SPI flash and an additional 8 MB SPI Pseudo static RAM (PSRAM). The module supports data rates up to 150 Mbps, and 20 dBm output power at the antenna to ensure a wide physical range.

ESP32-Wrover development board, is developed from the modules that come with attached additional circuitry such as voltage regulators, in our case peripheral for camera integration and micro USB interface. It allows an easier connection with PCs to compile, run and download code directly on the module[6].

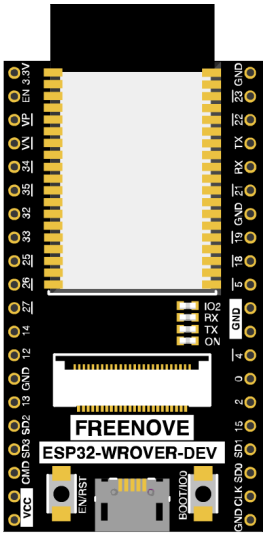


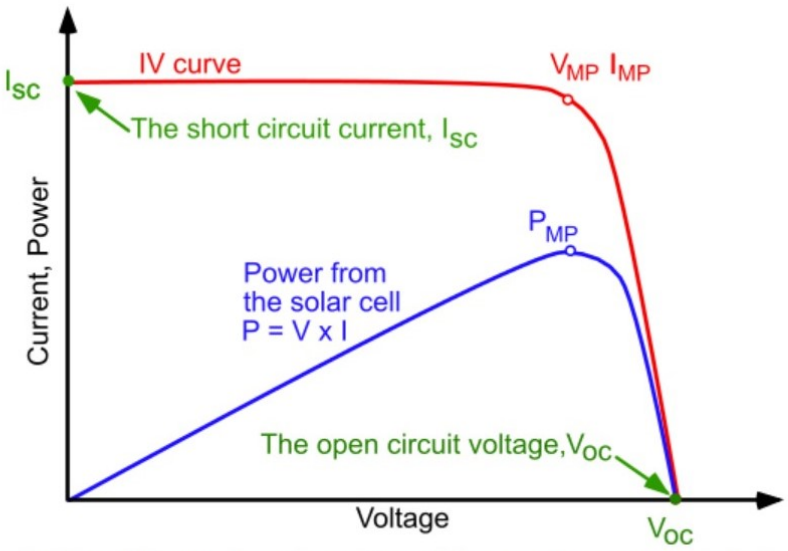
Figure 2.2.2: Picture of ESP32-Wrover Development Board

2.3 Photovoltaic Cells

Photovoltaic cells are a crucial part of any space satellite to provide energy in space. Solar Panels (Photovoltaic cells) convert sunlight into electrical power for satellite operations. As the name implies (photo = light, volt = the electromotive force), a photovoltaic cell converts the sun's energy into a flow of electrons, i.e., electric power; this is called the photovoltaic effect. Photovoltaic cells are the primary source of power in a Satellite, as there is no other source of energy readily available for use in space. During the presence of sunlight, electrical power generated charges the batteries onboard. This ensures, for instance, that the satellite is continuously supplied with power even during an eclipse.

IV(Current-Voltage) Curve

The Solar Cell I-V Characteristic Curve shows the current and voltage (I-V) characteristics of a particular photovoltaic (PV) cell [7]. The I-V curve provides the information required to configure a solar cell system to operate close to its maximum power point. The Maximum Power Point (MPP) is where the power supplied by the solar cells that are connected to the load (batteries, resistor) is at its maximum value, measured in Watts (W) or peak Watts (Wp). When the solar cell is open-circuited, not connected to any load, the current will be at its minimum (zero), and the voltage across the cell is at its maximum, known as the solar cell open circuit voltage, or V_{oc}. When the solar cell is short-circuited, the two ends of the terminal are directly connected, the voltage across the cell is at its minimum (zero), but the current flowing out of the cell reaches it's maximum, known as the solar cell short circuit current, or I_{sc}. At a particular combination of current and voltage, the power from the solar cell reaches its maximum value. This point is known as the "maximum power point" or MPP, defined as the ideal operation of a photovoltaic cell.



Current voltage (IV) cure of a solar cell. To get the maximum power output of a solar cell it needs to operate at the maximum power point, P_{MP}.

Figure 2.3.1: IV Characteristic of a Photovoltaic Cell[8]

2.4 Hardware components

OV2640 Camera

The OV2640 is a world's first 1/4 inch 2-megapixel sensor. The camera sensor can be connected to the Freenove ESP32-wrover board.

DHT11

The DHT-11 is a low-cost Digital Humidity and Temperature Sensor. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component. It connects to a high-performance 8-bit microcontroller. The operating voltage of the sensor is 3V to 5V. The measurements taken from the surroundings are sent as digital signal output to the data pin, which can be read with a 4.7kOhms pull-up resistor to VCC [9]. Although humidity is not measured in real satellites it is included in MiniMIST as it is a nice addition to the model.

Buzzer

The buzzer is an electrical device with an integrated structure that produces a buzzing sound. They are of three types: mechanical, electro-mechanical, and piezo-electric. These types are classified based on the type of tones they produce. Its operating voltage is 3 V - 16 V DC. The frequency range of a piezo-electric buzzer is 31 Hz - 65535 Hz and can differ from type to type [10].

2.5 Programming Languages for micro-controller

Users can use a variety of development platforms to program the ESP32. For example, Espruino – JavaScript SDK and firmware closely emulating Node.js, Mongoose OS – An operating system for IoT devices (recommended platform by Espressif Systems and Google Cloud IoT), or use a software development kit (SDK) provided by Espressif or one of the platforms listed on WiKiPedia. The two languages chosen to program the ESP32 are MicroPython because of its simplicity and Arduino C for its popularity [11]. Here is more information about the languages.

2.5.1 MicroPython

MicroPython is an open-source Python programming language interpreter on small embedded development boards. With MicroPython, you can write clean and simple Python code to control hardware instead of using complex low-level languages like C or C++ (what Arduino uses for programming). It is a complete re-implementation of python 3, designed to work under constrained conditions that the micro-controller supports. MicroPython consists of a Python compiler to bytecode and a runtime interpreter of that bytecode. The user is presented with an interactive prompt (the REPL) to execute supported commands immediately. It is included with a selection of core Python libraries. MicroPython also includes modules which gives the programmers access to low-level hardware on the microcontroller.

2.5.2 Arduino Language (C++)

Arduino language is a standard Application Programming Interface (API) which is used to program the microcontrollers using the C and C++ programming languages. Arduino Language is a subset of C++. C++ is a human-readable programming language. A program written in the Arduino Programming Language is called sketch, which is processed and compiled into machine language when uploaded on a microcontroller. The Arduino Integrated Development Environment (IDE) is the main text editing program used for Arduino programming. It is where you'll be typing your code before uploading it to the board you want to program.

2.6 Socket Communication (Web Server)

The MiniMIST will not mimic practical satellite communication. It uses WiFi to represent satellite communication by running a web-server on the ESP32. The ground station (teacher laptop or phone) will be a client (using a browser) that requests data from the ESP32 socket server.

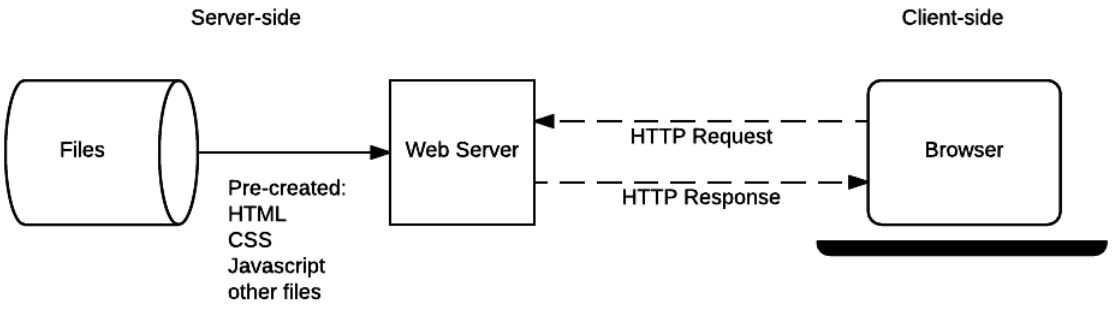


Figure 2.6.1: HTTP Server-Client Communication [12]

A web server is a place that stores, processes, and delivers web pages to Web clients. A web client is nothing but a web browser that connects to the web server. The communication between client and server takes place using a special Hypertext Transfer Protocol (HTTP).

2.6.1 HTTP protocol

The text-based Hypertext Transfer Protocol (HTTP) protocol enables online communication between web browsers and servers. The HTTP protocol allows us to fetch resources like HTML documents. The client, usually a Web browser, sends messages to the server, called Request, and the reply message from the server is called the response. The server responds with the content of that web page or an error message if unable to do so (like the famous 404 Error). The client and server connection over the internet is served after the server responds to the Request. For each Request, the client must establish a new connection.

2.6.2 WiFi mode

The ESP32 has a feature where it can connect to the WiFi network or start its network where other devices can connect. The MiniMIST uses STATION mode to connect to the WiFi network. The ESP32 can operate in the following modes:

- *STATION(STA) Mode* allows the ESP32 to connect to an existing WiFi network. The wireless router sets the IP address of the ESP32. With this IP address, it can set up a web server and deliver web pages to all connected devices under the existing WiFi network.

- *Soft Access Point (AP) Mode* allows the ESP32 that creates its WiFi network and act as a hub (Just like a WiFi router) for one or more stations.

2.7 Front-end Design

2.7.1 HTML, CSS and CSS Grids

HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout for various devices. They are the basis of building Web pages and Web Applications.

CSS Grid is a two-dimensional layout that you can use to create web-responsive items. The Grid items are arranged in columns and rows, allowing us to give the website a good structure without altering the HTML code.

The website uses 98.css, a CSS library for building interfaces that look like Windows 98, to give users an old ground station feel.

2.7.2 Highcharts.js

Highcharts is a JavaScript-based charting library that enhances web applications by adding interactive charting capability. Highcharts improve data visualization on web pages with various options, such as chart types, animations, and more [13].

2.7.3 AJAX

AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and JavaScript. AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. With AJAX, it is possible to update parts of a web page without reloading the whole page.

It uses a browser's built-in **XMLHttpRequest (XHR)** object to communicate with the server in the background, asynchronously, without interfering with the user experience or blocking the page in any way. XMLHttpRequest Object is an API in the form of an object whose methods help transfer data between a web browser and a web server[14].

The satellite in our project sends more than one type of data, making AJAX a vital part of a user-friendly interface. See the 2.7.1 below shows how AJAX communication occurs.

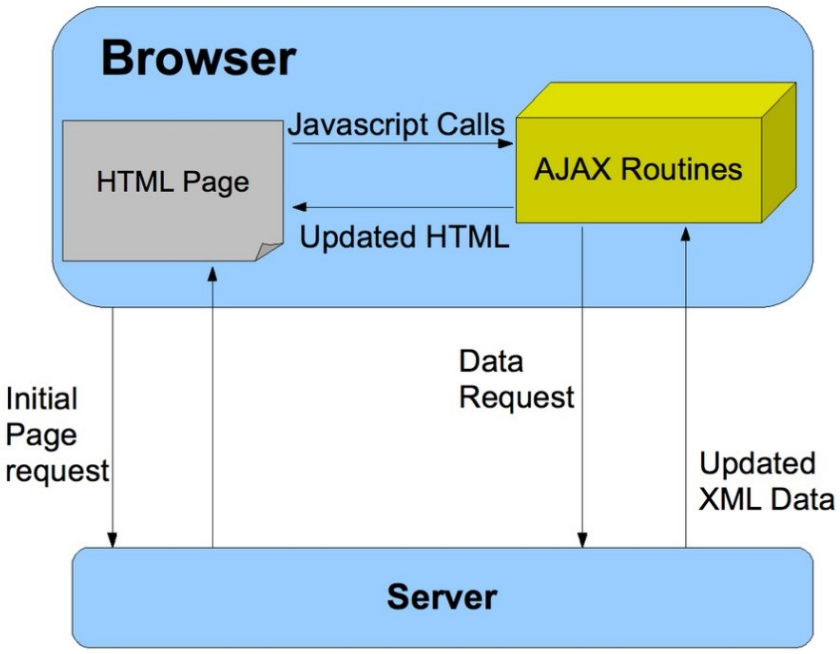


Figure 2.7.1: AJAX Communication[15]

Chapter 3

Methods

The method section will describe how the system is constructed and how each part of the functionality is implemented. It involves sensors, micro-controller, communication system, programming languages, User interface, and other components used, the motivation behind them, and tests used to verify the parts.

Firstly, the project's purpose and goals were defined and discussed with the project owner. The requirements for the MiniMIST was finalized and can be found in Section 1.4. Second, each part of the system was researched to decide on the different sections of this project. After being agreed upon, the main components were selected, and the general data flow was completed. It was decided to use micropython and Arduino C to program the ESP32. The project was meant for teaching purposes, and it was seen as positive to include both programming languages for a better demonstration. The user interface was developed and finalized. The design was tested and configured for the device.

3.1 Defining the requirements

The satellite project has to approximate a practical satellite simulation for teaching embedded programming. It is required to have essential functions that a satellite is used for in the real world, such as measuring temperatures, but not humidity, from the surrounding, getting images of the surrounding, and providing data on the power supply onboard. These were the finalized data used in the project.

3.2 Choice of hardware and software

3.2.1 Choice of sensor

The sensor used in the system has to be low-powered. The data collected from the sensors should be relevant to satellite data. The thesis narrowed it down to having a temperature and Hall sensor. The thesis project included DHT11 having low power consumption and being able to read humidity and temperature in one device. Although humidity would not be measured in a satellite, in a classroom demonstration it can be used. The ESP32 supports a built-in Hall sensor (to measure the magnetic force), but there was a clash when using the Hall sensor and camera module at the same time. The pins used in the Hall sensor had the same GPIO pins used by the camera, which caused a problem when capturing images and initializing the camera. Hence the hall sensor was rejected and finally narrowed down to only the DHT11 sensor.

3.2.2 Choice of hardware components

The MiniMIST will be run remotely powered from battery. Hence it was necessary to indicate if errors were occurring during the initiation or while the system was running. The thesis project decided to include buzzers and LEDs as indicators in the project, even though buzzers and LEDs can't be used in an actual satellite. Buzzers are useless in space as you cannot hear sound and LEDs are not useful as they consume unnecessary energy.

3.2.3 Choice of Interface

The user interface had to be asynchronous. The data requested from the ground station is displayed on the web page without being requiring a refresh command from the user. Hence AJAX was the most viable choice to use. The device transfers more than one data with the ground station(client browser), and it is a bad user interaction if the entire page is refreshed for every request from the satellite or if there are multiple routes (like ”IPAddr/camera,” ”IPAddr/sensor,” ”IPAddr/time”) for every data that is requested. AJAX is an efficient way to communicate with the web browser in the background without interfering with the website on display unless the function is intended.

To give an old satellite ground station look, 98.css was used. It is an open-source design that provides the display style like the Windows 98 design. It offers more flavor to the website and makes it look more relevant to the project.

The thesis required a graph to give output data to display the collected data better. Highchart was the most viable option amongst other charts available open-source. Highchart supports the real-time update of a graph and other options, such as zooming in and moving through the

Setup 1	Closest		Setup 2	medium			Setup 3	far			Setup 4	Off	
Vos	9.5V		Vos	8.3V			Vos	7.4V			Vos	6.7V	
Isc	1mA		Isc	0.36mA			Isc	0.15mA			Isc	0.09mA	
Rc	9.5Kohm		Rc	23Kohms			Rc	49kOhms			Rc	75kohms	
Rl	9.4Kohms		Rl	Vmp	Imp		Rl	Vmp	Imp		Rl	Vmp	Imp
Vmp	7.8V		16kOhms	5.2V	0.33mA		38kohms	4.9V	0.12mA		69kOhm	4.9V4.7V	0.06mA
Imp	0.84mA		23Kohm	6.2V	0.28mA		42Kohms	5.1V	0.12mA		42Kohms	5.1V	0.12mA
							47kohms	5.5V	0.11mA		47kohms	5.5V	0.11mA
Vos	Voltage open soursce												
Isc	Current short circuit												
Rc	Resistance assumed												
Rl	Resistance of load used												
Vmp	Voltage at maximum power point												
Imp	Current at maximum power point												
Setup	Distance of lamp from solar cells												

Figure 3.3.1: Solar Panel Reading Data with different light settings

graph to see patterns.

3.2.4 Choice of communication

An actual satellite will not use a web server to communicate with the ground station but will use radio communication to send predefined blocks of information regularly. However MiniMIST uses WiFi communication and HTTP to access the information in the model satellite. It connects to the network and starts a server on the ESP32. The web server will send anybody accessing the IP address of the server the HTML response.

3.3 Solar Panel

The Solar panel used in the experiment was AK53X30. In experiments and measurements made from different illuminated surroundings, the open-source voltage was a maximum of 5 V, and the short circuit current was 1 mA. The Solar panels were meant to power the ESP32 and batteries with the help of a buck-boost converter. The low power output from the solar panels, when used indoors, was not enough to charge the battery or ESP32. Therefore the solar panel is only used to analyze using the ADC pins to measure the voltage and current supplied.

An experiment was conducted to measure the V_{oc} and I_{sc} and the I_{mp} and V_{mp} under different light to get the maximum power from the solar panels. Two solar panels were connected in series that could give us a maximum of 10 V and 1 mA as output. The solar panels is shown with lights from a lamp inside a classroom. Different readings were measured depepnding on the distance of the lamp from the solar panels. When the Lamp is turned off the light on the solar panel is from the classroom where there is no other source of light. The 3.3.1 shows the results from measurements.

After an experiment, the V_{oc} and I_{sc} in a classroom without sunlight were calculated to be approximately 7.6 V and 0.25 mA. To find the MPP (Maximum Power Point), we can change the resistance of the load and calculate the V and I for which we get maximum power. It was

measured to be around 30 kΩ. The final decision was to use a 30 kΩresistance load to obtain maximum capacity from the light inside a classroom setting. The analog-to-digital converter (ADC) GPIO pin in ESP32 can read only a maximum of 3.3 V. A higher voltage can damage the ESP32 chip and reduce its lifespan. A solid 10 V is way above the acceptable voltage range, so a voltage divider is used to produce a fraction of the voltage from Solar panels. With 8 kΩresistor and 22 kΩresistors to retrieve desired output range between 0 V to 2.67 V for the input voltage range 0 V to 10 V.

Chapter 4

Design

4.1 ESP32-Wrover Development Board

The MiniMIST aims to mimic the onboard computer of a satellite, this is done by starting a web server, on the ESP32, by connecting to the WiFi network with the provided network credentials. Once the network connection completes, the ESP32 listens to requests from the client. When the client initially connects to the web server, it receives a web page to display on the web browser. The web page has a built-in JavaScript functions that allow the users to request additional data by clicking respective buttons available to perform requests to get satellite data, such as an image, temperature and humidity, solar panel voltage, and the remaining battery percentage. Figure 4.1.1 shows a flow chart depicting the process occurring when ESP32 starts.

The JavaScript function designed to get satellite data uses asynchronous communication with HTTP requests in the background. For each type of request, the Server performs a set of actions. Figure 4.1.2 is a flow chart of how the Server resolves each request from the client and the data returned.

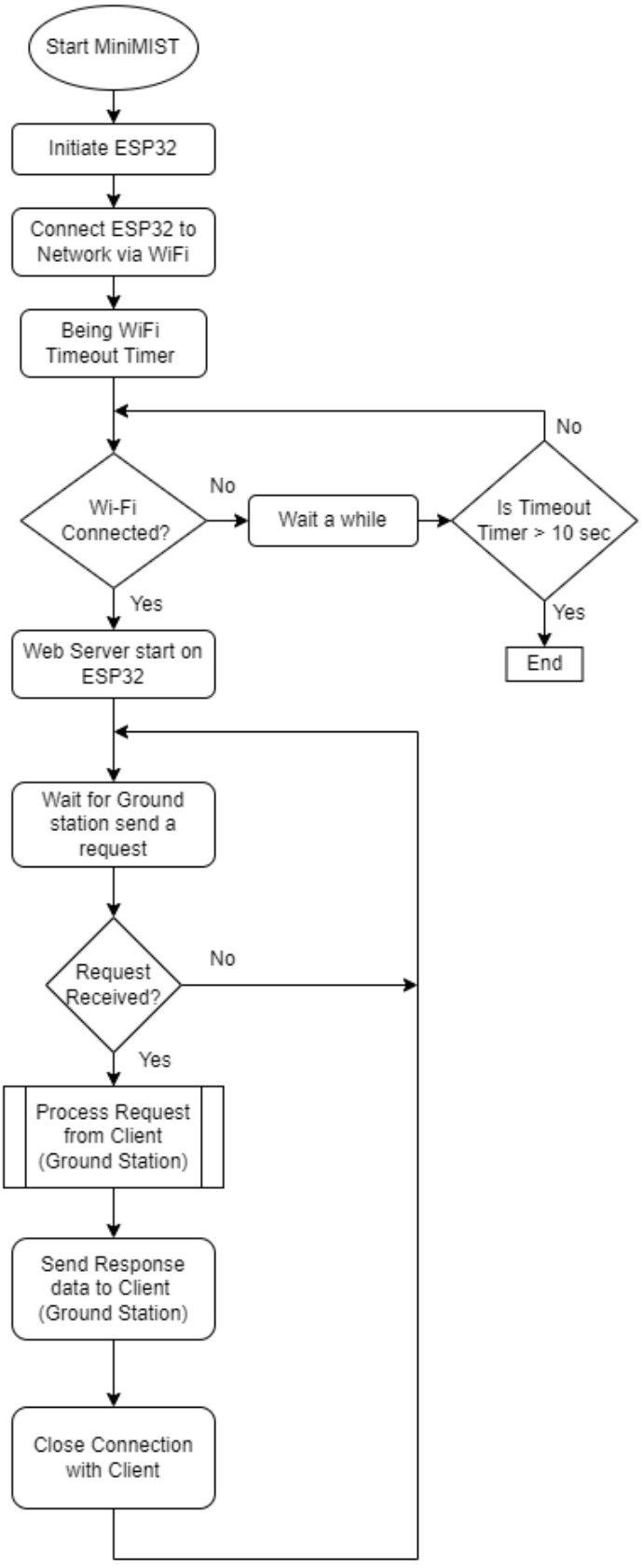


Figure 4.1.1: Main Function of ESP32

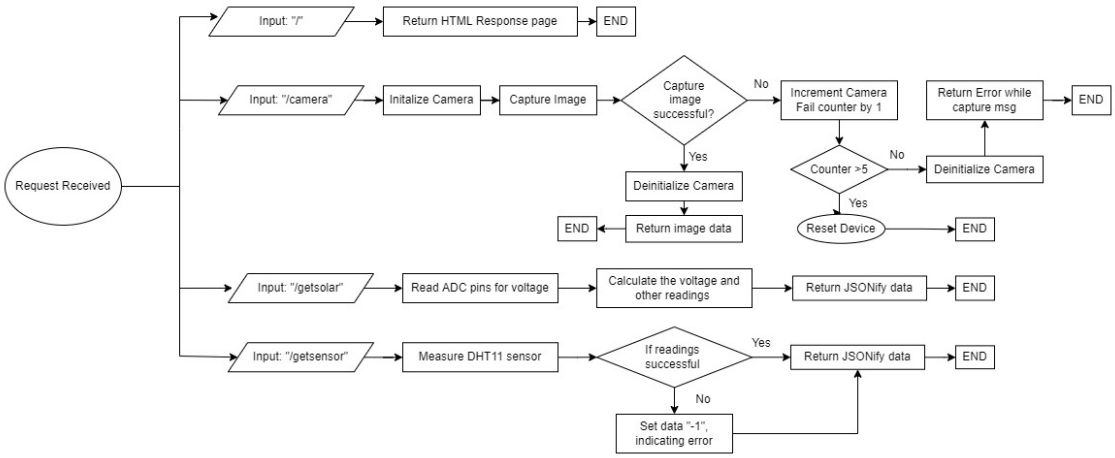


Figure 4.1.2: Processing Request

The ESP32 can accept the following request:

- "/": Request for the index page where ESP32 can display satellite data.
- "/getimage": Request for an Image from the onboard Camera.
- "/getsolar": Request to get information from the Solar Panels (Voltage, Current, and Power) and the remaining battery percentage.
- "/getsensor": Request for data from the DHT11 sensor (Humidity and Temperature in Celsius and Fahrenheit)

4.2 Hardware

4.2.1 Photovoltaic Cells

Photovoltaic cells are a significant part of a satellite, so it is unfair not to include them in the MiniMIST project, even if they produce a low power output. Therefore in this project, we simulate photovoltaic cells powering the ESP32 but read the voltage from the solar panels to demonstrate the voltage, current, and power, like in an actual satellite. We use the AK53X30 Solar Panels that give us a maximum of 5 V and 1 mA (5 mW), which can barely power the ESP32 at its peak performance. Hence the voltage from the solar panel array is only measured by an ADC in the ESP32-wrover development board.

The solar panel array consists of the cells connected in series, so the maximum output Voltage is 10 V, called the open circuit voltage (V_{oc}), and the maximum current is 1 mA, called the short circuit current (I_{sc}). We have the maximum voltage and current we can get, but we need to find the maximum power that the load can attain from the solar panels, for which we need to find I_{mp} (current at maximum power) and V_{mp} (voltage at maximum power).

The MiniMIST will be used majorly in an indoor classroom setting. For this reason, the load resistance was chosen to be approximately 30 kΩ.A voltage divider must be used because the ESP32 ADC pin can have only a maximum 3.3 V input, otherwise it will be damaged. The 30 kΩwas built from a 22 kΩand 8 kΩresistor. The return voltage on the ADC pin can never go above 3.3 V for the maximum voltage provided by the two solar panels in series, which is 10 V to 11 V. The current is calculated from the voltage measured from solar panels and the resistance of 30 kΩ.

4.2.2 Power Supply

Three triple-A batteries are connected to a voltage regulator that converts the input voltage to 3.3 V output voltage. The battery is the main power supply to the ESP32 when it is not connected to a PC with a micro-USB cable[16]. The voltage regulator's output is connected to the 3.3 V input Pin on the ESP32 using a switch, so the user can turn off the power supply when the ESP32-Wrover development board is connected to a PC with a micro-USB cable. The battery percentage was measured using a voltage divider circuit connected to the three triple-A batteries. This voltage is input into the ADC pin 33 of ESP32.

4.2.3 DHT11 sensor

The DHT11 sensor reads the temperature and humidity of its surroundings and sends digital signals of the values in the Data line pin. It can be powered with a range of 3.3 V to 5.5 V. The data line is connected to a 4.7 kΩpull-up resistor to the 3.3 V pin.

4.2.4 Other Components

Other components involve a buzzer and LEDs to indicate different functions performed by the ESP32. It is used to indicate when a picture is taken from ESP32 and when error occurs in reading data.

Buzzer: The Buzzer is used here to indicate the users during a case of error and essential activities when the device is running. The ESP32 will be working remotely, and there is no way to monitor the activities on its Server unless it is connected to the PC. One of the requirements for the MiniMIST is that it works remotely, powered by the batteries onboard. The Buzzer uses minimal current and does not need an external resistance; therefore, the Buzzer was connected to the Pin 14 and cathode to the ground without any load resistor.

Camera: ESP32-Wrover board has an interface to connect the OV2640 Camera. The Camera uses some of the pins on ESP32-Wrover, and it is essential not to use these designated pins when the Camera is initiated and running.

4.2.5 Schematics

After the components are put together, the final design is shown in the schematics. Two PCBs must be made for the MiniMIST; one is the main board with all the parts except the solar panels. The other is the solar panel circuit connected perpendicular to the main board, facing upwards in the direction of a light source. The figure 4.2.1 shows the schematics of the main board and figure 4.2.2 shows the schematics for the solar panel board.

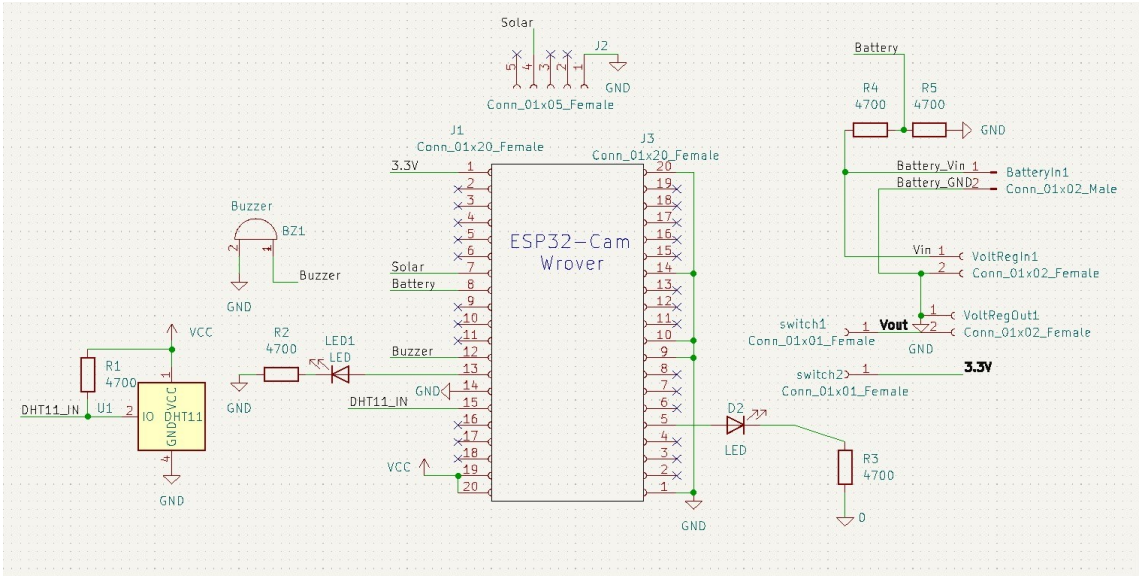


Figure 4.2.1: Schematics of Main board

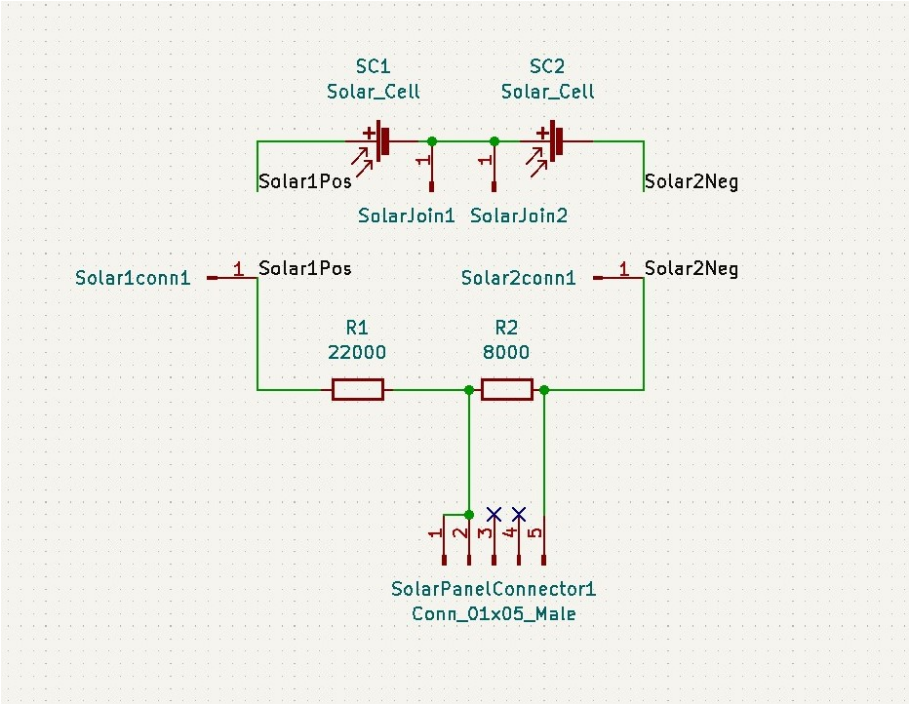


Figure 4.2.2: Schematics of Solar board

4.3 Software for Onboard station

4.3.1 WiFi Module

ESP32 can connect to the network and set it up, which other devices can connect. In this project, the ESP32 connects to the router/ network locally. This allows the other devices connected in the network to access the device by its IP address, assigned by the router or hostname, manually set in the code: ”minimist-thesis.” The user interface uses external libraries like 98.css and highchart.js that are not stored in the ESP32. Therefore it is required for the client to be connected to the internet. Consequently, the user chose STATION mode on the WiFi module. To connect to the WiFi, we create an object configured to the desired mode, STATION(STA_IF) mode, and then set active. Once the object is created, we connect to the router with the SSID and password. With a 10s timeout, if the WiFi does not connect, the device prints a cannot connect notification, is notified with a buzzer sound, and resets the machine. If the connection succeeds, the ESP32 prints its IP configuration and make a sound on the Buzzer.

4.3.2 Socket Communication (Web-Server)

A socket represents an endpoint on a network device, and communication can proceed when two sockets are connected. Internet protocols are built on top of sockets, such as email (SMTP), the web (HTTP), telnet, and ssh, among many others. Each protocol is assigned a specific port, which is just an integer. Given an IP address and a port number, you can connect to a remote device and start talking with it. This project uses HTTP, which is port 80. The default port is when you type the address into the web browser.

4.3.3 Camera

”/camera”: When the request is to get the image function, getimage is executed. First, the camera is initiated with the appropriate arguments. For the type of ESP32, the assignments of the pins to initiate the camera might vary. For the ESP32-Wrover, the configurations are:

```
camera.init(
    0, d0=4, d1=5, d2=18, d3=19, d4=36, d5=39, d6=34, d7=35,
    format=camera.JPEG, framesize=camera.FRAME_VGA,
    xclk=21, pclk=22, vsync=25, href=23, siod=26, sioc=27,
    pwdn=-1, reset=-1)
```

Once the camera is initiated, the image is captured and sent to the client.

4.3.4 ADC (reading voltage)

The ESP32 integrates two 12-bit SAR (Successive Approximation Register) ADCs, supporting 18 measurement channels (analog-enabled pins). The ADC driver API supports ADC1 (8 channels, attached to GPIOs 32 - 39) and ADC2 (10 channels, attached to GPIOs 0, 2, 4, 12 - 15, and 25 - 27). However, the ADC2 channel is also used by the WiFi driver; hence it is not advisable to use this channel when WiFi is being used. The ADC is configured to get the maximum precision and attenuation provided by the device. The input from the ADC pins varies from 0 - 3.3V. Any voltage input higher than 3.3 V can damage the device. This voltage is converted to digital reading from 0 - 4095 for the highest attenuation.

”/getsolar”: This request handles two functions; it reads and calculates data about the solar panels and checks for remaining battery functions. **Reading solar panel data**: The pins designated to read the solar panel voltage are configured to ADC with maximum attenuation (Range 0 V to 3.3 V) and maximum resolution (12 bits, digital reading 0 to 4095). After the experiment found the margin of error in the readings from ADC pins, it was fixed by calibrating the measurements. Once the configuration is set, we will read the voltage from the pins. For Solar cells, the digital value read is converted into voltage with the formula: Once the calculations are done, we will get the voltage from the pins used to calculate the V_{in}, current, and power in solar panels. This data is stored in a JavaScript Object Notation (JSON) object and sent to the client.

Calibration The ADC pins are not known for their quality, resulting in a slightly deviated reading for the solar panels and batteries. The readings from the ADC pins and the actual values were compared to measure the inaccuracy. Here is the graph of the reading measured. It was seen that the ADC readings were off by 0.13 V is added to the actual reading. This might vary from device to device. To fix this error, a calibration technique was used to calculate voltage, where 0.13 V adds the result before being sent to the ground station.

4.3.5 DHT module

”/getsensor”: On request, the DHT11 sensor is read at the instance. The readings from the sensor are stored in a JSON object that the web server will return to the client. JSON is used here to make the data transfer more organized and easy to interact with. In Case where an error occurs while reading the sensor, an Error key in the JSON object is set to True, and the other values are set to -1. Since the DHT11 sensor can only measure the temperature range 0-50 °C and humidity range 20-80%, -1 will be an outlier on the display and indicate the error in the readings of the sensor.

4.3.6 HTML code

The route "/": When the client enters the device's IP address, it automatically sends an "/" index page request to the Server hosted on the designated IP address. On the index page request, the ESP32 returns the template for the website. The get index function is executed when the index page is requested; in MicroPython, the function stores a string object that has the HTML, CSS, and JavaScript code and is returned when the function is run. More information about the front-end code can be found in the Software for ground station section.

4.4 Software for the Ground Station

4.4.1 HTML structure

The ground station should have the following division to display individual satellite data: A Header for Title, Image, control system, graphs, table, date and time, and Footer.

- Header: The title of the project MiniMIST.
- Image: This section displays the image from the ESP32. It is later configured to rotate 90 degrees because the ESP32 returns an image that is turned, and there are no options to configure the rotation while capturing. Still, there are options to set quality, mirror, flip, and more.
- Graphs: This division contains all the graphs for solar voltage with time, current with time, power with time, humidity, and temperature. It uses Highchart to display the data from the Server for graphical interpretation and analysis.
- Table: It has the latest data from the Server displayed in the table format for easier viewing.
- Date and time: use the browser library to get the current time from the internet.
- Control System: It has a group of buttons that allows the user to interact with the Server. The controller will enable us to retrieve the image, sensor, and solar data. An additional button that allows stream function where the other three functions have in order with an interval of 5sec.
- Footer: Details of the project.

4.4.2 CSS Grids and design

The CSS grid is a two-dimensional layout system for web pages, allowing the display of content to be organized into columns and rows. Grids allow us to create complex layouts in a fair,

straightforward way. Each grid layout will contain any number of columns and rows and the gaps that separate them, known as gutters.

The web page for the ground station is separated into three columns and five rows, resulting in a total of 15 grids. Each grid is allocated a value specifying the division it will hold. For example, the first three grids in row one are allocated for the header in B.2.1. The code in B.1 specifies how the grids and space alignment are performed on this web page. The figure B.2 displays the grid outline when the screen is viewed on a laptop.

The web page also supports responsive mode. When the website is viewed on the phone screen or screen width of less than 653px, the display changes to align all grids in one column and multiple rows. The code that makes the web page responsive is listed in B.3. This makes the view easier to read and navigate when using smaller screens.

4.4.3 AJAX

The function of AJAX is to communicate with the Server in the background without disrupting the main page. This functionality sends requests and receives the corresponding response data from ESP32. This data consists of images, sensor information, and data from the solar panels. The buttons on the control sections allow the user to perform asynchronous communication with the Server and retrieve appropriate data. Here is how AJAX works in steps [17]:

- An event occurs on a web page (A button is clicked)
- An XMLHttpRequest object is created by JavaScript
- The XMLHttpRequest object sends a request to a web server (such as ”/getimage”, ”/getsensor”, or ”/getsolar”)
- The Server processes the request
- The Server sends a response back to the web page
- The response is read by JavaScript
- Proper action (like updating graphs and the table) is performed by JavaScript

The web page also supports a stream function that executes all the AJAX functions to retrieve satellite data one after the other with a set time interval. It uses a callback function that allows the function to be passed into another function as an argument, invoked inside the outer function to complete the respective action. This is used to get sensor data, solar data, and images one after the other with a time interval before calling each function to avoid crashing the Server. The entire callback function is executed with a set time interval of 5 seconds, which is the average time taken to complete all the functions. Other buttons to communicate with

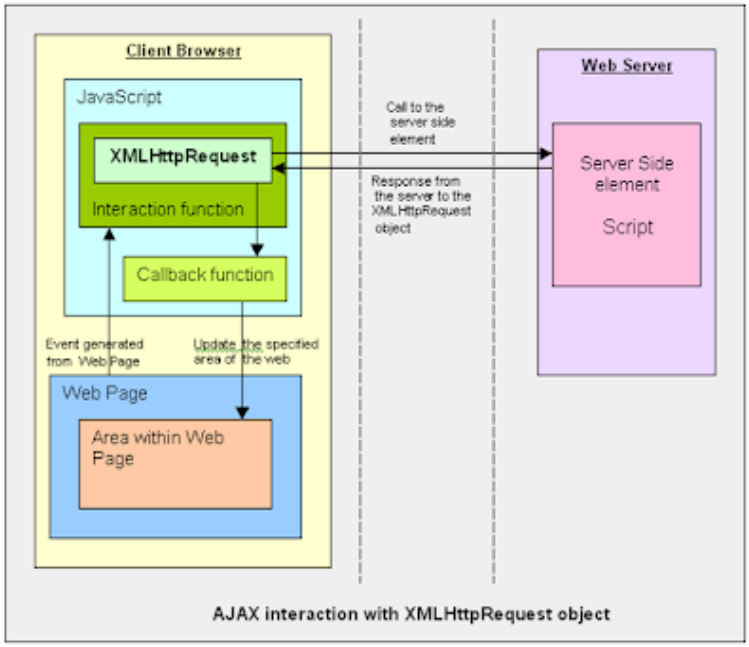


Figure 4.4.1: AJAX communication between Client(ground station/we browser) and Web Server(ESP32) [18]

the Server are disabled when the stream button is pressed. Therefore, it avoids unnecessary clashes in request handling.

In our function, three AJAX calls begin made from the client side depending on the desired data. Image, Power data, and Sensor Data. The Image functions create a request to the Server with the ”/getimage” route. The Server returns an image captured by the camera, which is updated into the Image division of the web page. The Sensor function requests the Server for data from the DHT11 sensor, and the solar function request details about the solar panels and power supply in the Server. The Server responds to the request similarly, with a JSON object containing the information. This data is processed and updated on the table and the graph.

4.4.4 Graphing with Highcharts.js

The JSON object the Server responds with when requesting sensor data and solar panel data is parsed and converted into a float data type by JavaScript. The data is updated dynamically on the data set of the respective graph. The Highchart graph displays the point on the web page with a line animation moving from the previous point to the new one. When the data set of the particular graph extends over 25 points, the oldest point on the data set gets removed, and the new point will be updated. This option is called ”shifting” in the Highchart. The chart also provides the user to zoom in to a section of the graph and move around. This option helps to analyze different data sets and time intervals in a graph.

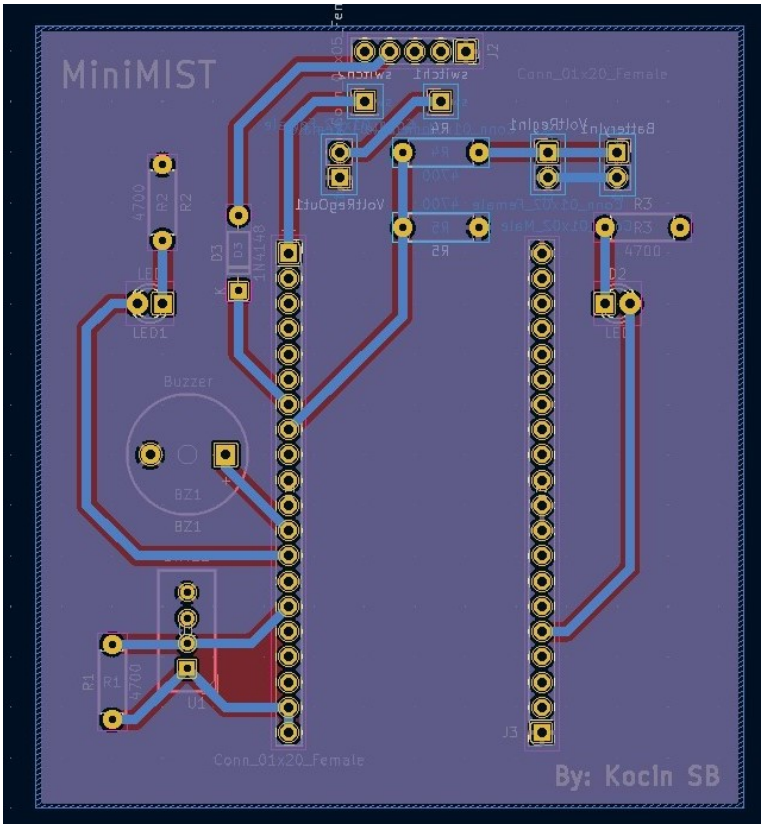


Figure 4.5.1: Main PCB board Layout

4.5 PCB

The PCB was designed using Kicad and manufactured in the Mentor Space. The final stage of the design was built with the schematics. You can look at the outcome in the Results section.

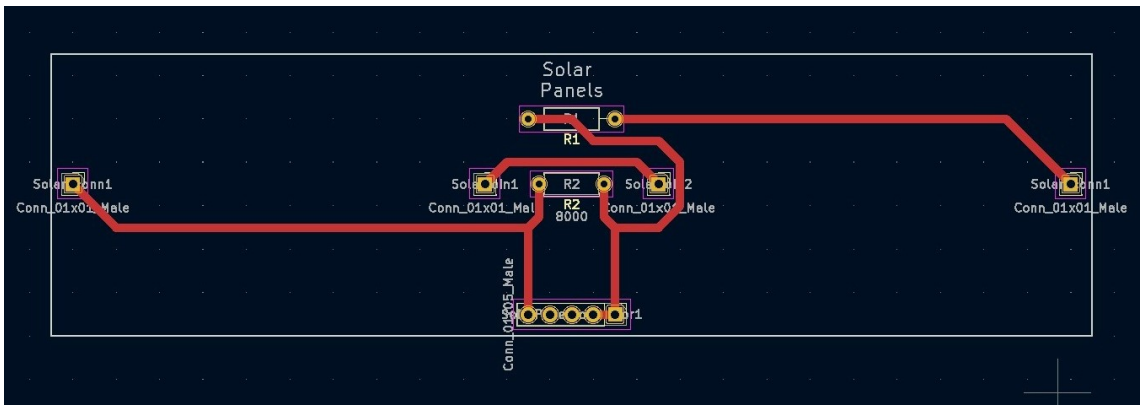


Figure 4.5.2: Solar Panel PCB Layout

Chapter 5

Result

5.1 PCB of Solar panels and Main Board

After the research and development phase, the final working prototype was built. The MiniMIST was designed to meet the essential requirement it was given. It had an onboard computer that performed actions like an experimental satellite. It had a hardware design that monitors satellite data from the components and returns it to the ground station on request. The satellite data included images that were captured by the camera onboard. The camera is sometimes difficult to initiate and results in an empty image buffer. When this problem arises, a camera fails to counter that is incremented on each consecutive count. If the camera fails to capture five times a row, the ESP32 will reset and start the system again. It is difficult to debug the error when the device runs on a remote setting. The other satellite data are temperature and humidity, read using the DHT11 sensor.

The system is powered using batteries in the remote setting and micro-USB when the device is connected to a computer. A switch can be turned on and off to connect the device to the batteries because having two power supplies can damage the device and reduce its life span.

The solar panels connected to the devices are only used for demonstration purpose and doesn't power the MiniMIST because of its low power output. The readings from the solar panels are displayed on the graph of the ground station to see how it changes with the different light settings.

The MiniMIST has met the requirement specified in the Goal of the project.

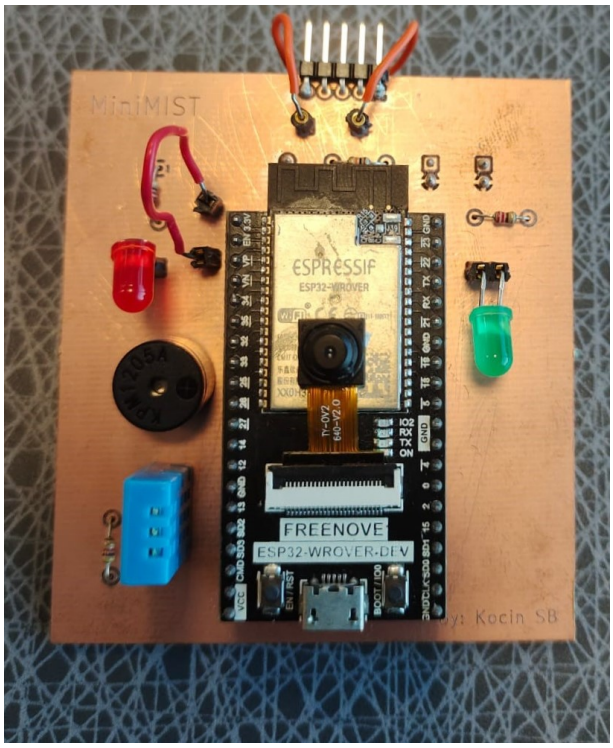
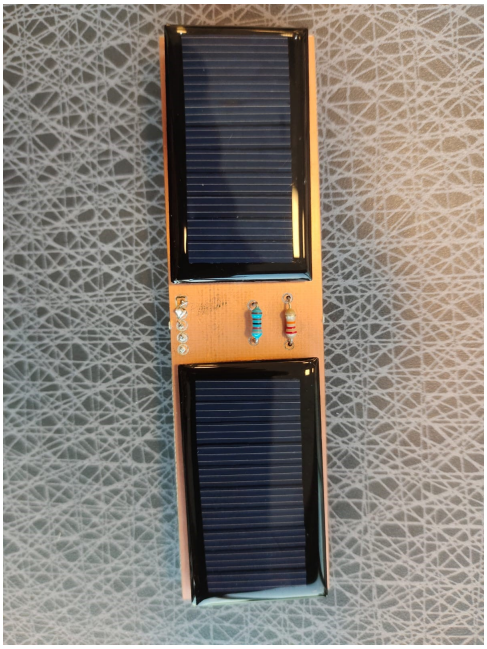
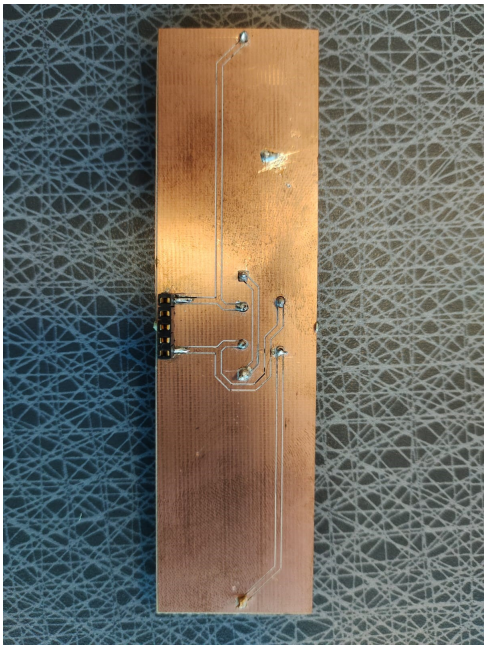


Figure 5.1.1: PCB Main Board Front



(a) PCB Solar Board Front



(b) PCB Solar Board Back

Figure 5.1.2: Printed Circuit Board for Solar Panels

Figure 5.2.1: Web page for the PC

5.2 Website

The website design had sections to display all the data received from the server. It had buttons in the control section that allowed the user to interact with the server. The function to interact with the server was built using AJAX. Hence any action happening will be in the background and not interfere with the web page displayed. Figure 5.2.1 shows a working web page connected to the ESP32 device.

Chapter 6

Conclusions

Most of the targets described earlier in the design and introduction section were met but insufficient. More work and consideration are required in order to achieve a final version. Just a short example of this would be the design for the layout of the components could have a much further improvement and be placed over the PCB in a better layout to improve the performance since some of the traces on the PCB are long and hence calls for a design which is optimized to the capacity. The current state of the project is in, cannot be described as the final version and needs to go through many revisions.

Another important conclusion that can be drawn from this project work is that project works that are highly dependent on the availability of information along with pre-made microchips and controllers. It is beneficial to only start with the project if the required material is readily available.

6.1 Future Work

The ESP32 is a power-hungry device when using the RF function in the chip. This drains the power from the battery quicker as the Server running on the device is constantly ON unless it is turned off. A way to reduce power consumption is something to consider in future work. The ESP32 can use the technique used by the MIST CubeSat to put the device in different modes depending on the power remaining on the battery.

6.2 Final Words

As an engineer, it is fascinating to learn how satellites function and operate. Furthermore, it was quite an enriching experience to learn how ESP32 functions and explore a lot of new technologies and tools available.

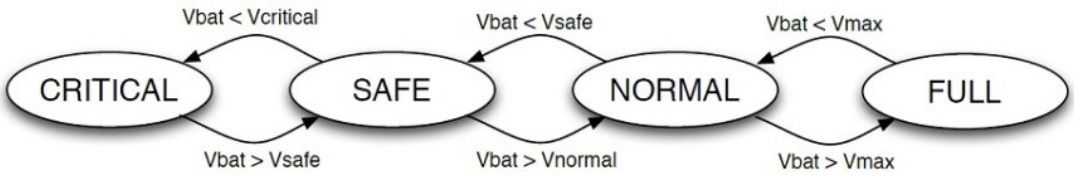


Figure 6.1.1: Software battery low-voltage protection[1]

It also came to light how different programming languages affect the performance of ESP32. For example, MicroPython can almost do anything an Arduino C++ code can do. However, one thing to realize is that MicroPython code isn't as fast and might use a little more memory compared to similar Arduino or other low-level C/C++-based code.

In conclusion, the prototype worked as described and expected with an interactive user interface. All three project sections were accomplished, the hardware, software for the onboard computer, and ground stations. The final prototype is now used in the course after some final modifications.

Bibliography

1. *Miniature Student Satellite* https://mistsatellite.space/. (accessed: 23.08.2022).
2. Campbell, A. *How do satellites communicate?* https://www.nasa.gov/directorates/heo/scan/communications/outreach/funfacts/txt_satellite_comm.html. (accessed: 23.08.2022).
3. Howell, E. *The First Cubesats to Mars Were Almost Lost Upon Arrival* https://www.space.com/nasa-marco-mars-cubesats-nearly-lost.html. (accessed: 23.08.2022).
4. Howell, E. *Tiny Payloads, Huge Benefits for Space Research* https://www.space.com/34324-cubesats.html. (accessed: 23.08.2022).
5. *ESP32* https://www.espressif.com/en/products/socs/esp32. (accessed: 23.08.2022).
6. *Freenove$_{E}SP32_{W}ROVER_{B}oard$* https://github.com/Freenove/Freenove_ESP32_WROVER_Board. (accessed: 23.08.2022).
7. *Solar Cell I-V Characteristic* https://www.alternative-energy-tutorials.com/photovoltaics/solar-cell-i-v-characteristic.html. (accessed: 23.08.2022).
8. *Solar Cell I-V Characteristic* https://www.pveducation.org/pvcdrom/solar-cell-operation/iv-curve. (accessed: 23.08.2022).
9. *DHT sensors* https://docs.steminds.com/kits/mindev_kit/dht_sensors/. (accessed: 23.08.2022).
10. *Active Passive Buzzer* https://components101.com/buzzer-pinout-working-datasheet. (accessed: 23.08.2022).
11. *Insight Into ESP32 Features Using It With Arduino IDE* https://lastminuteengineers.com/esp32-arduino-ide-tutorial/. (accessed: 23.08.2022).
12. *Client-Server Overview* https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview. (accessed: 23.08.2022).
13. *Highcharts* https://www.highcharts.com/. (accessed: 23.08.2022).

14. *JavaScript Ajax* https://www.tutorialrepublic.com/javascript-tutorial/javascript-ajax.php. (accessed: 23.08.2022).

15. Walsh, F. *AJAX in a nutshell* https://medium.com/hackernoon/ajax-in-a-nutshell-4770da7b16f3. (accessed: 23.08.2022).

16. *ESP8266 Voltage Regulator (LiPo and Li-ion Batteries)* https://randomnerdtutorials.com/esp8266-voltage-regulator-lipo-and-li-ion-batteries/. (accessed: 23.08.2022).

17. *AJAX Introduction* https://www.w3schools.com/xml/ajax_intro.asp. (accessed: 23.08.2022).

18. *Implementing simple AJAX interaction in your Web Application using XMLHttpRequest object* https://www.javareference.com/implementing-simple-ajax-interaction-in-your-web-application-using-xmlhttprequest-object/. (accessed: 23.08.2022).

Appendix - Contents

Appendix A

ESP32

A.1 ESP32-Wrover Module

Figure A.1.1: ESP32-Wrover Module

A.2 ESP32 Pin Details

Name	No.	Type	Function
GND	1	P	Ground
3V3	2	P	Power supply
EN	3	I	Module-enable signal. Active high.
SENSOR_VP	4	I	GPIO36, ADC1_CH0, RTC_GPIO0
SENSOR_VN	5	I	GPIO39, ADC1_CH3, RTC_GPIO3
IO34	6	I	GPIO34, ADC1_CH6, RTC_GPIO4
IO35	7	I	GPIO35, ADC1_CH7, RTC_GPIO5
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
IO12 [1]	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
GND	15	P	Ground
IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SHD/SD2 [2]	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SWP/SD3 [2]	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SCS/CMD [2]	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SCK/CLK [2]	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SDO/SD0 [2]	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SDI/SD1 [2]	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
NC1	27	-	-
NC2	28	-	-
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
NC	32	-	-
IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
GND	38	P	Ground

Notice:

1. GPIO12 is internally pulled high in the module and is not recommended for use as a touch pin.
2. Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the SPI flash integrated on the module and are not recommended for other uses.

Figure A.2.1: ESP32 Pin Details

A.3 Freenove ESP32-Wrover Development kit

Figure A.3.1: Freenove ESP32-Wrover development board

A.4 ESP32-Wrover Pin

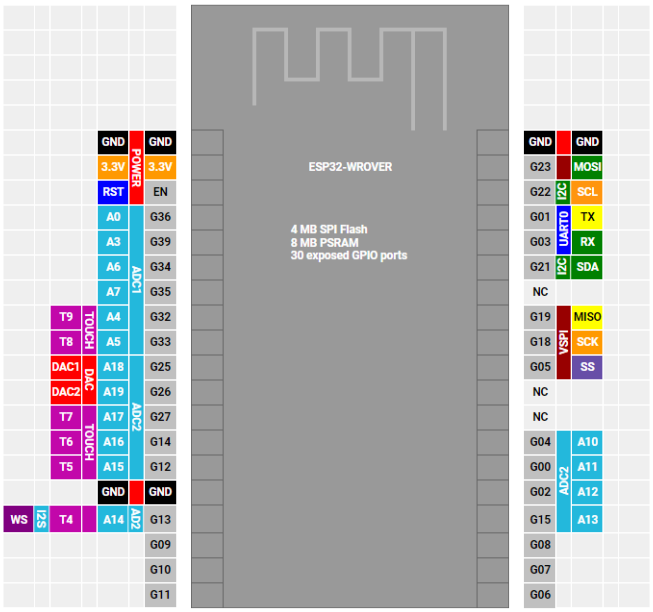


Figure A.4.1: ESP32-Wrover Module Pin Configuration

Appendix B

CSS Style

B.1 CSS Grid PC Screen Code

```
.container {
  display: grid;
  grid-template-columns: 2fr 0.6fr 1fr;
  grid-template-rows: 50px 1.5fr 0.8fr 0.2fr 55px;
  gap: 10px 10px;
  grid-auto-flow: row;
  grid-template-areas:
    "Header Header Header"
    "Image Graph Graph"
    "Image ControlImg Sensor"
    "Image ControlImg DateTime"
    "Footer Footer Footer";
}
.Header { grid-area: Header; }
.Image { grid-area: Image; }
.Graph { grid-area: Graph; }
.Sensor { grid-area: Sensor; }
.ControlImg { grid-area: ControlImg; }
.DateTime { grid-area: DateTime; }
.Footer { grid-area: Footer; }
```

B.2 CSS Grid Diagram

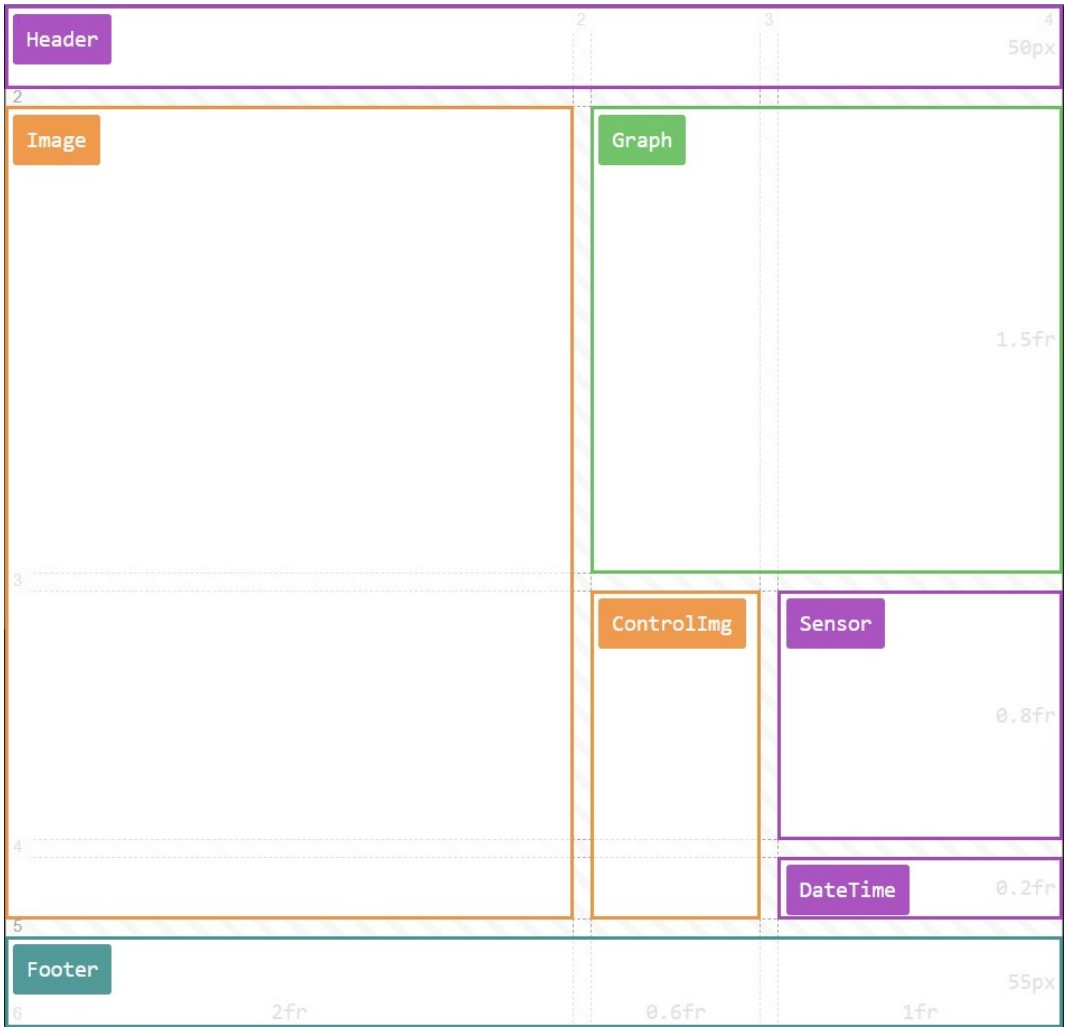


Figure B.2.1: Grid layout for website on computer screens

B.3 CSS GRID Phone Screen Code

```
@media screen and (max-width: 653px) {
    .Container {
        grid-template-columns: 100%;
        width: auto;
        height: max-content;
        grid-template-rows: 50px 200px 300px 240px 160px 45px 65px;
        grid-template-areas:
            "Header"
            "ControlImg"
            "Image"
            "Graph"
```

```
                "Sensor"
                "Datetime"
                "Footer"
            ;
    }
}
```