Fayoum University

Engineering Faculty

Electrical Engineering Department

B. Eng. Final Year Project

Smart Farming System

By:

Mahmoud Badr Rabea Mahmoud (ECE)

Mazen Ahmed Mohamed Fahim (ECE)

Musa Mahmoud Salah Ahmed (CSE)

Omar Mohammed Alghareeb (CSE)

Mohamed Abdelaleem Saad (CSE)

Supervised By:

Prof. Ahmed Mostafa

Supervisor: Prof. Ahmed Mostafa *Date of examination: 04 July 20*

J u l y – 2 0 2 3

DEDICATION

We would like to dedicate our graduation project to our families, whose unwavering support and encouragement have been instrumental in our academic journey.

To our parents, thank you for always believing in us, for your sacrifices, and for your endless love. Your constant support and guidance have been a driving force for us. We would not have been able to achieve this milestone without your unwavering support.

To our siblings, thank you for being our pillars of strength, for your motivation, and for always cheering us on. Your support and encouragement have been a source of inspiration for us.

We are grateful for having such a wonderful family that has always been there for us, through thick and thin. This project is dedicated to you as a token of our love and appreciation.

Once again, thank you to our families for being our biggest supporters and for all the sacrifices you have made to help us achieve our dreams.

ACKNOWLEDGMENT

We would like to express our deepest gratitude to Dr. Ahmed Mustafa and Eng. Amgad Sameer for their invaluable support and guidance throughout our graduation project. Their expertise, patience, and dedication have been instrumental in shaping our research and helping me achieve my goals.

Dr. Ahmed Mustafa, our supervisor, has provided us with insightful feedback and constructive criticism that helped me refine my ideas and improve the quality of my work. His wealth of knowledge and experience in the field has been a constant source of inspiration and motivation for us.

Eng. Amgad Sameer, our mentor, has been instrumental in providing us with technical support and guidance throughout my project. His attention to detail and meticulous approach to problem-solving have been a great asset to us.

I am grateful for the time and effort that Dr. Ahmed Mustafa and Eng. Amgad Sameer have invested in my project. I am proud to have had the opportunity to work with such skilled and dedicated professionals.

Once again, thank you, Dr. Ahmed Mustafa and Eng. Amgad Sameer, for your invaluable contributions to our project and for being such wonderful mentors.

DECLARATION

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Bachelor of Science in Electrical Engineering is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Mahmoud Badr Rabea

Registration No.: 1

Date: Tuesday, 04 July 2023.

Signed: Mazen Fahim

Registration No.: 1

Date: Tuesday, 04 July 2023.

Signed: Musa Mahmoud

Registration No.: 1

Date: Tuesday, 04 July 2023.

ABSTRACT

Our project is a smart farming system that reads several agricultural sensors including a camera and feeds the data to a backend server. The server then saves the received data in a database for future reference and renders it to a website on which the farm owner can monitor his farm health. The owner will be notified if pests are detected using a computer vision model. The Farm will also have a timed watering system to limit water wastage.

Contents

Table of Figures

Table of Tables

Table of Equations

LIST OF ACRONYMS/ABBREVIATIONS

ACRONYM	Definition of Acronym
SRS	Software Requirements Specification.
ARM	Advanced RISC Machine
RISC	Reduced Instruction Set Computer
MCU	Microcontroller Unit
PCB	Printed Circuit Board
MCAL	Microcontroller Abstraction Layer
APIs	Application Programming Interfaces
SRAM	static Random Access Memory
ECU	Electronic Control Unit
PLL	Phase Locked Loop
HSI	High-Speed Internal
HSE	High-Speed External
LSE	Low-Speed External
RCC	Reset and Clock Control
GPIO	General Purpose Input Output
NVIC	Nested Vector Interrupt Controller
IRQ	Interrupt Request
DMA	Direct Memory Access
ADC	Analog to Digital Converter
DCMI	Digital Camera Interface
EXTI	External interrupt
USART	Universal Synchronous Asynchronous Receiver Transmitter
Synch	Synchronous
Asynch	Asynchronous
I2C	Inter integrated Circuit
HAL	Hardware Abstraction Layer
ESP	Espressif Systems
MQTT	Message Queuing Telemetry Transport
QoS	Quality of service
OV	Omni Vision
LIB	Library
STD	Standard
ISR	Interrupt Service Routine
IoT	Internet of things
TCP	Transmission Control Protocol
UDP0	User Datagram Protocol
QoS	Quality of service
PUBSUB	Publish Subscribe
MSB	Most significant bits
CONNACK	Connect acknowledgment
PUBACK	Publish acknowledgment
SUBACK	Subscribe acknowledgment
UTF-8	Unicode Transformation Format 8-bit

ID	Identity
DUP	Duplicate
FIFO	First In First Out
LDR	Light Dependent Resistor
AHT	Aosong Humidity and Temperature
WI-FI	Wireless Fidelity
kPa	Kilo Pascal
IC	Integrated Circuit
LM	Linear Monolithic
HVAC	Heating, Ventilation and Air Conditioning
ASIC	Application-Specific Integrated Circuit
MEMS	Micro Electronic Mechanical Systems
LED	Light-Emitting Diode
PSU	Power Supply Unit
V	Voltage
A	Ampere
AC	Alternating Current
DC	Direct Current
IP	Ingress Protection
NPN	Negative, Positive, Negative
TO	Transistor Outline
BJT	Bipolar Junction Transistor
Pd.	Power Dissipation
V_{BE} (on)	Base-Emitter ON Voltage
Max	Maximum
approx.	Approximately
RH	Relative humidity
CRC	Cyclic redundancy check.
SCCB	Serial Camera Control Bus
fps	Frames per seconds
VGA	Video Graphics Array
AEC	Automatic Exposure Control
AGC	Automatic Gain Control
AWB	Automatic White Balance
ABF	Automatic Band Filter
ABLC	Automatic Black-Level Calibration
WTR	Water

Introduction

We start off by defining the problem we aim to solve. Then, we discuss the motivation behind solving this problem and what goals we aim to achieve.

Problem Definition

The agriculture in Egypt has some problems that cause lower crops production and quality and need to be solved. The agriculture sector in Egypt represents about 14.7% of the national product. There are about 8.5 million people that work in that sector which represents 27% of the Egyptian work force as stated by Mohamed Marzok the minister of agriculture of Egypt. Some of the problems we aim to solve are:

1. No convenient way to easily monitor the current state of the farm to check on its health.
2. A huge amount of effort is made, and time taken to regularly water the crops.
3. Waste of water in irrigation process.
4. Late discovery of pests or crops infection.
5. Lack of sunlight.

Project Goals

This system enables owners to monitor their farm easily without effort and from the comfort of their home.

1. Automates the irrigation process to limit wastage.
2. Detects any pests or crop infections.
3. Ensure higher crops production and quality.

System Description

The main goal of the project is for the farm owner to be able to monitor his farm health and vital parameters. As well as automating the irrigation process for him.

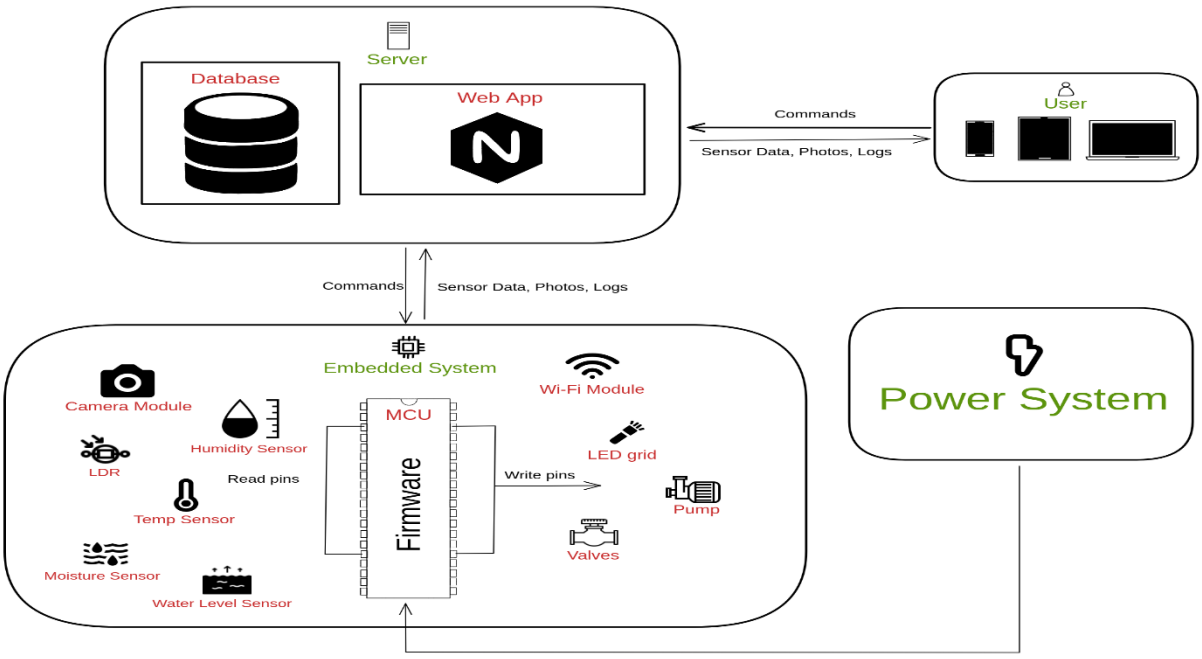

Figure 0.1 System Overview

The main component that controls the overall system and has the core logic on which the system runs is the STM32F429ZI microcontroller. It communicates with a server via the ESP8266 Wi-Fi module sending sensors' data and images through MQTT protocol.

Every 5 minutes the MCU sends readings taken by the sensors and an image taken by the OV7670 camera module of the farm to the server. The MCU reads data the following sensors:

1. **Camera module.**
2. **Temperature sensor.**
3. **Moisture sensor.**
4. **Humidity sensor.**
5. **Water level sensor.**
6. **LDR.**

Moisture, temperature, and humidity sensors are used to give an overall view of the current state of the farm and how healthy it is.

A LDR sensor is used to detect if there is enough light coming through to the plants. If there is not the MCU will send a control signal to light up the LED grid giving the plant the light they need for photosynthesis.

A water level sensor is used to detect if the main water tank is running low. If it is, the MCU will send a control signal to the pump to refill the empty tank.

The server backend receives all the data sent by the MCU and stores it in a database for future reference. Then it renders all the data received in a user-friendly UI. So, the user will be able to monitor the overall current state of his farm.

It also feeds the image received through an AI model that's able to detect if there're any pests or infections in the plants. The user will be notified if there's any.

The MCU sends control signals for the following modules to achieve automatic irrigation:

1. **LED grid.**
2. **Valves.**
3. **Pumps.**

A LED grid is needed to provide enough light for the photosynthesis process if it gets cloudy or if it's an indoor smart farm that can't get much sunlight.

Valves are needed to control the opening and closing of the main water tank to automate the irrigation process.

Part 1: Embedded Module

Chapter 1: System Analysis and Planning

System Analysis

system analysis is a critical process for ensuring that a system meets the needs of its stakeholders and is designed and implemented in a way that is efficient, effective, and sustainable over time.

System analysis typically involves the following steps:

a- Requirements gathering:

This involves identifying the needs and requirements of the system's stakeholders, including users, customers, and other interested parties. And this includes our SRS.

SRS

stands for "Software Requirements Specification." It is a document that outlines the functional and non-functional requirements for a software project. The SRS document serves as a blueprint for the software development team, providing a clear and detailed description of what the software is supposed to do and how it should function.

The SRS document typically includes sections such as an introduction, functional requirements, non-functional requirements, system architecture, user interface design, system constraints, and testing requirements. The document is usually created during the initial stages of the software development lifecycle and serves as a reference point throughout the development process.

The purpose of the SRS document is to ensure that the software development team and stakeholders are on the same page with regards to the project requirements. It helps to prevent misunderstandings and miscommunication, which can lead to costly delays and errors in the development process. By having a clear and detailed SRS document, the development team can ensure that they are building the software that the stakeholders want and need.

Our SRS

I. Introduction:

The Smart Farming System is a software system that aims to improve the efficiency and productivity of farming operations. The system uses sensors and other technologies to monitor and control various aspects of the farming process, including soil moisture, temperature, and humidity.

II. Functional Requirements:

1. Sensor Integration:

The system should be able to integrate with various sensors, including soil moisture sensors, temperature sensors, humidity sensors, and other environmental sensors.

2. Irrigation Control:

The system should be able to control irrigation systems based on the data collected from the sensors. This includes automatically turning on or off irrigation systems based on soil moisture levels.

3. Pest Detection:

The Smart Farming System should be able to detect pests and diseases in crops using image processing and machine learning techniques.

4. Image Processing:

The system should be able to process images captured by cameras installed in the fields to identify and classify pests and diseases.

5. Machine Learning:

The system should use machine learning algorithms to train on a dataset of images of healthy and diseased crops to improve the accuracy of pest and disease detection.

6. Notification:

The system should notify farmers when pests or diseases are detected in their crops, including the type of pest or disease and the severity of the infestation.

III. Non-Functional Requirements:

1. Security:

The system should be designed with strong security measures to protect the data collected from the sensors and ensure that only authorized users can access the system.

2. Reliability:

The system should be reliable, with a high degree of uptime and minimal downtime due to hardware or software failures.

3. Scalability:

The system should be able to scale to accommodate additional sensors and devices as needed.

4. Usability:

The system should be easy to use, with a user-friendly interface that allows farmers to easily monitor and control their farming operations.

IV. System Architecture:

The Smart Farming System will be a cloud-based system, with a central server that collects and analyzes data from the various sensors and devices.

V. User Interface Design:

The system's user interface will be designed to be intuitive and easy to use, with a dashboard that displays real-time data on soil moisture, temperature, humidity, and livestock health. The interface will also include controls for managing irrigation systems, monitoring crop growth, and managing livestock.

VI. System Constraints:

The Smart Farming System will require an internet connection to transmit data to the central server and receive commands from the mobile application. The system will also require compatible sensors and devices to be installed on the farm.

VII. Testing Requirements:

The Smart Farming System will undergo rigorous testing to ensure that it is reliable, secure, and meets all functional and non-functional requirements. This will include testing the system with various sensors and devices, as well as testing its scalability and security measures.

b- System modeling:

This involves creating models and diagrams to represent the system's components, functions, and interrelationships. This section will involve system block diagram to have an initial image of the whole system.

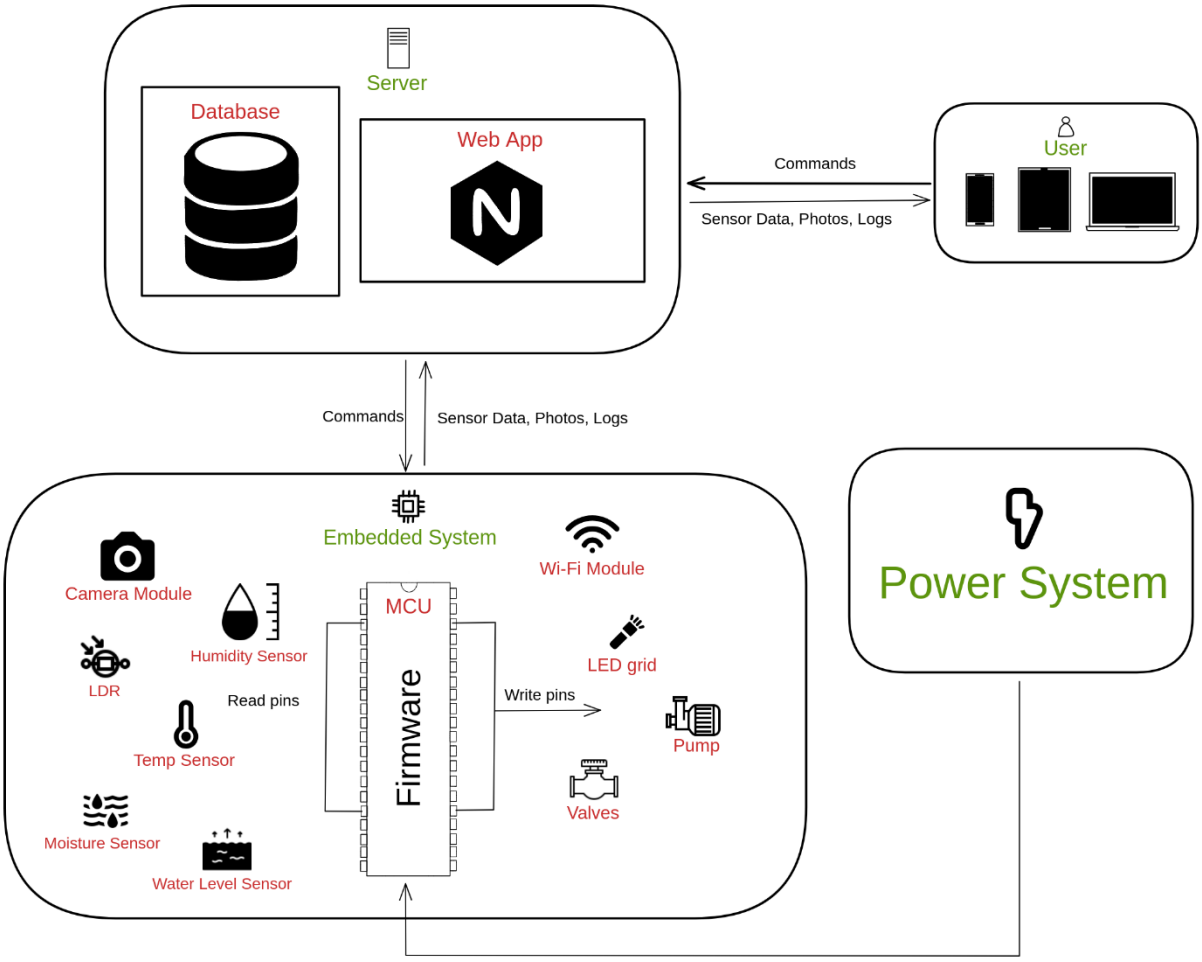

Figure 0.1 System Block Diagram

c- System design

This involves designing the system architecture, including hardware, software, and network components. This can include selecting the appropriate hardware and software platforms, designing the database schema, and designing the user interface. And this will be discussed later in the incoming chapters.

d- Implementation:

This involves building and testing the system, including writing code, configuring hardware and software components, and testing the system for functionality, performance, and security. And this will be discussed later in the incoming chapters.

e- Maintenance:

This involves maintaining and updating the system over time, including fixing bugs, adding new features, and addressing security vulnerabilities. And this will be discussed later in the incoming chapters.

Main Plan

Before we start designing and implementing the system, we need to have some rules to follow since there are multiple team members working on the same codebase. These rules will make it easier for us to communicate with each other without any confusion whatsoever. It also makes it easier for us since it constrains you to limited choices so that we don't waste time trying to decide what we should do. For example, when naming a variable, the possibilities are endless. Without any templates or rules that are agreed upon the codebase will look messy and unorganized resulting in some confusion.

Naming Rules

The first thing any professional work starts with is to define some naming rules that have to be followed by all of the team members with no exception. As it facilitates the readability of the code and makes it much easier to maintain.

I. Readability and Understanding: Consistent naming conventions make code more readable and understandable for developers. When everyone follows the same rules, it becomes easier to comprehend the purpose and functionality of variables, functions, classes, and other code elements.
II. Collaboration and Teamwork: In software development, teamwork and collaboration are vital. When multiple developers work on the same codebase, having consistent naming conventions helps them understand and navigate each other's code more efficiently. It promotes seamless collaboration and reduces confusion.
III. Maintainability and Scalability: As software projects grow in size and complexity, maintaining and modifying code becomes challenging. Consistent naming conventions make it easier to locate and modify specific code elements. It simplifies maintenance tasks, such as debugging, refactoring, and adding new features, saving time and effort in the long run.
IV. Code Reusability: Consistent naming conventions enhance code reusability. When code is organized and named consistently, it becomes easier to identify reusable components and functions. Developers can quickly understand how to integrate existing code into new projects, improving efficiency and reducing redundancy.

Our naming rules is as follows:

Folder Name	Swc
File Name	Swc_FileName.c
Local Variable	Local_VariableName
Global Variable	Swc_VariableName
Const Macro	SWC_MACRO_NAME
Function-Like Macro	Swc_MacroFunctionName(Parameter)
Type definition Name	Swc_TypeName_t

Table 0.1 Naming Rules

Templates

Now that we have agreed upon what naming rules we should follow, we need to answer the question of how our files will be structured whenever we implement a new software module. The template should define the directories and the files structures. The templates should also define boilerplate comments and code for each file type. Our team agreed upon the following directory and file structure:

Figure 0.2 Files Structure

As mentioned above the template defines boilerplate code and comments that needs to be present depending on the file type. For example, the interface file has a different template than the config file and so on. When we first started working on the project the process of copying and pasting each template of each file to it appropriate file type was a hideous process that took a lot of time and effort.

We have automated the task using a python script that's able to generate the entire folders and files with the templates already copied inside of them given nothing but the software component name, which layer it belongs to and the name of the author. We have named the tool "mkmodu" and the command is run as follows:

python mkmodu.py Swc Layer "Author Name".

This simple command will generate everything we need to start implementing the module. The source code for the script is of course on the GitHub repo of the project. This moves us to the next point which is how different people worked in parallel on the same codebase.

Inclusion model

Now, it's time to use how to use these files together to start building the project so we start to plan our inclusion model and after studying all possible cases and see the scenarios we finally have the model in the figure below:

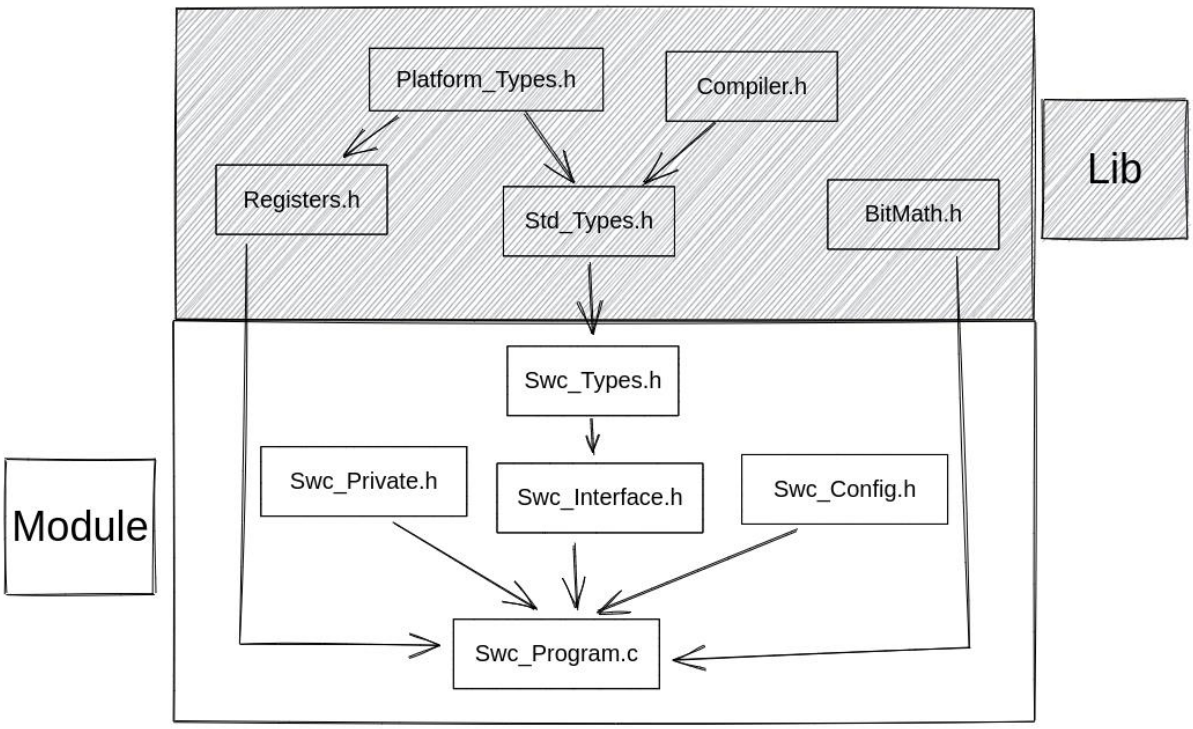

Figure 0.3 Inclusion Model

Version Control

Version control is a system or a set of tools used to track and manage changes to files and documents over time. It allows multiple people to collaborate on a project by keeping a record of every modification made to the files, along with who made the changes and when they were made.

Version control systems (VCS) provide a repository where all the files and their versions are stored. These systems enable users to create new versions of files, compare different versions, revert to previous versions if needed, and merge changes made by different users.

There are two main types of version control systems: centralized version control systems (CVCS) and distributed version control systems (DVCS). In a CVCS, there is a central repository that stores all the versions of the files, and users check out files from that repository to make changes. In a DVCS, each user has their own local repository that contains the complete history of the project, including all versions and changes made by themselves and others. This allows for more flexibility and offline work capabilities.

In our project, we have used a DVCS called “git”. This tool allowed us to have a single repository where all the team members cooperate in. each team member can work offline wherever he is on his task then whenever he has access to the internet, he pushes the changes to the repo for all other team members to see. The flow usually goes like this:

1. Branch from the main branch to dev branch where we can develop whatever we want without worrying about corrupting our main branch.
2. During the development of the software module, we commit changes on this dev branch
3. Once we have implemented the software module and is working fine and tested, we merge the dev branch with main.
4. In the final step, we push the changes to the remote repo for all other team members to see.

Timeline and execution Plan

Task	Start Data	End Date	Members
Hardware Selection	01-Oct	20-Oct	Mahmoud, Mazen & Musa
ARM MCAL Drivers	01-Oct	30-Nov	Mahmoud & Mazen
HAL Drivers (Agricultural Sensors)	20-Oct	30-Nov	Musa
Camera Module Driver	01-Dec	15-Dec	Mahmoud, Mazen & Musa
WIFI Module Driver	15-Dec	30-Dec	Mahmoud & Mazen
Application Implementation	01-Feb	15-Mar	Musa
Hardware integration / Testing	01-Mar	15-Apr	Mahmoud, Mazen & Musa

Table 0.2 - Embedded System Module

Task	Start Data	End Date	Member
Essential CS topics	01-Oct	11-Jan	Omar
Essential web technologies	01-Nov	30-Dec	Omar
IOT essentials	01-Feb	28-Feb	Omar
Essential Cloud technologies	01-Mar	30-Mar	Omar
Connection With MCU	01-Apr	30-Apr	Mahmoud, Mazen & Musa

Table 0.3 - Web Module

Task	Start Data	End Date	Member
Collect Dataset	15-Oct	30-Nov	Mohamed Abdelaleem
Manipulate Data	01-Dec	15-Dec	Mohamed Abdelaleem
Extract The Features	15-Dec	30-Dec	Mohamed Abdelaleem
Create The Model	15-Feb	30-Mar	Mohamed Abdelaleem
Feed Data into Model	01-Apr	30-Apr	Mohamed Abdelaleem

Table 0.4 - Computer Vision Module

Task	Start Data	End Date	Member
Farm Model Structure	01-Dec	31-Dec	Mahmoud, Mazen & Musa
Water System	01-Dec	31-Dec	Mahmoud, Mazen & Musa
Lighting System	01-Feb	28-Feb	Mahmoud, Mazen & Musa
Power Supply System	01-Feb	28-Feb	Mahmoud, Mazen & Musa
Overall System Integration	01-Mar	31-May	Mahmoud, Mazen & Musa

Table 0.5 - Farm Integration Module

Conclusion

Overall, system analysis is a critical process for ensuring that a system meets the needs of its stakeholders and is designed and implemented in a way that is efficient, effective, and sustainable over time. system planning is an essential process that helps organizations to identify their information system requirements and develop an appropriate strategy for meeting those needs. System planning is an ongoing process that requires continuous evaluation and improvement to ensure that the system remains relevant and effective in meeting the organization's needs. By adopting a systematic approach to system planning, organizations can maximize the value of their information systems and gain a competitive advantage in today's fast-paced business environment.

Chapter 2 System Design

Intro

Before we start implementing the system, we first started by the whole system design as a step of project planning so that we have the complete concept of the project that make it easier for us to implement the system in a systematic way and make best use of our time to achieve our goal. in the following, we will discuss the design in more detail.

Our system consists of two major parts the hardware and software so, we can divide the system design into two categories:

Hardware Selection and Design:

The goal of hardware selection is to identify the best combination of hardware components that will meet the requirements of the system while also being cost-effective and easy to manufacture.

We start by choosing our microcontroller based the requirement of the system that we need after searching, we decided to choose a microcontroller based on ARM.

MCU

There are several reasons why one might choose a MCU based on ARM processor for their application, including:

I. Minimal Cost and Power
II. Customizable
    - ARM processors come in a variety of configurations and can be customized to meet the specific needs of an application. This flexibility allows developers to choose the right processor for their project, whether it's a low-power microcontroller or a high-performance application processor**.**
III. 32-bit address and data bus
IV. Ultra Low Power with High Performance
V. Very Powerful and east to use Interrupt Controller
    - Supports up to 240 external interrupt source and 15 internal.
VI. RTOS Friendly
VII. ARM provides lots of documentations, and technical references manuals.

Overall, the choice of processor depends on the specific requirements of the application, including performance, power consumption, cost, and available resources. ARM processors offer a good balance between these factors and are widely used in a variety of applications.

The next step was to choose a specific MCU that has the needed requirements. After searching we find that the best choice is STM32F4xx series that produced by ST and we found NUCLEO PCB board shown in the figure below that is easy to use and powerful.

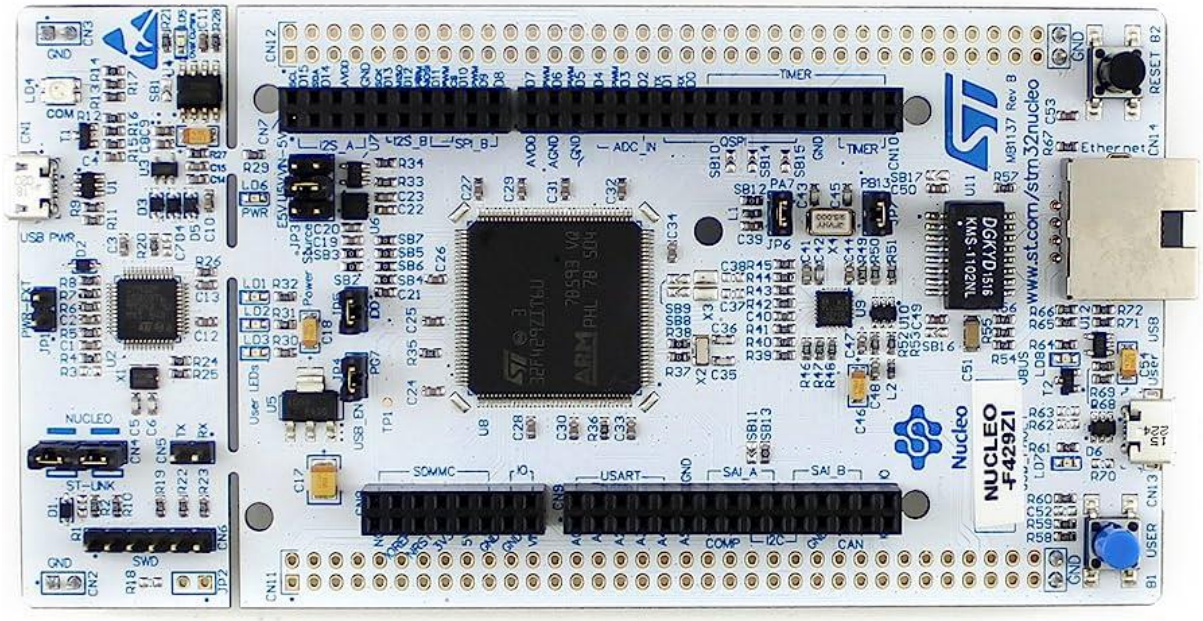

Figure 0.1 NUCLEO-F429

The specifications of NUCLEO are that it has:

- ✓ Core: ARM Cortex-M4F with FPU
- ✓ Clock frequency: up to 180 MHz
- ✓ Flash memory: 2 MB
- ✓ SRAM: 256 KB
- ✓ Timers: up to 17 timers (including 3 32-bit timers, 2 16-bit timers, and 9 general-purpose timers)
- ✓ ADC: up to 24 channels of 12-bit ADC with a conversion rate of up to 2.4 MSPS
- ✓ DAC: up to 2 channels of 12-bit DAC with a conversion rate of up to 1 MSPS
- ✓ Communication interfaces: up to 4 USARTs, up to 6 SPIs, up to 3 I2Cs, up to 2 CANs, up to 2 SDIOs, and up to 2 USB OTG FS/HS
- ✓ Up to 168 I/O pins with interrupt capability
- ✓ Two DMA controllers that acts as master in system and can transfer data up to 10 times faster than the processor
- ✓ DCMI that can work with digital camera and can process images with different extensions with help of DMA.
- ✓ Other peripherals: RTC, WDT, CRC, RNG, and more.
- ✓ Operating voltage: 1.7V to 3.6V

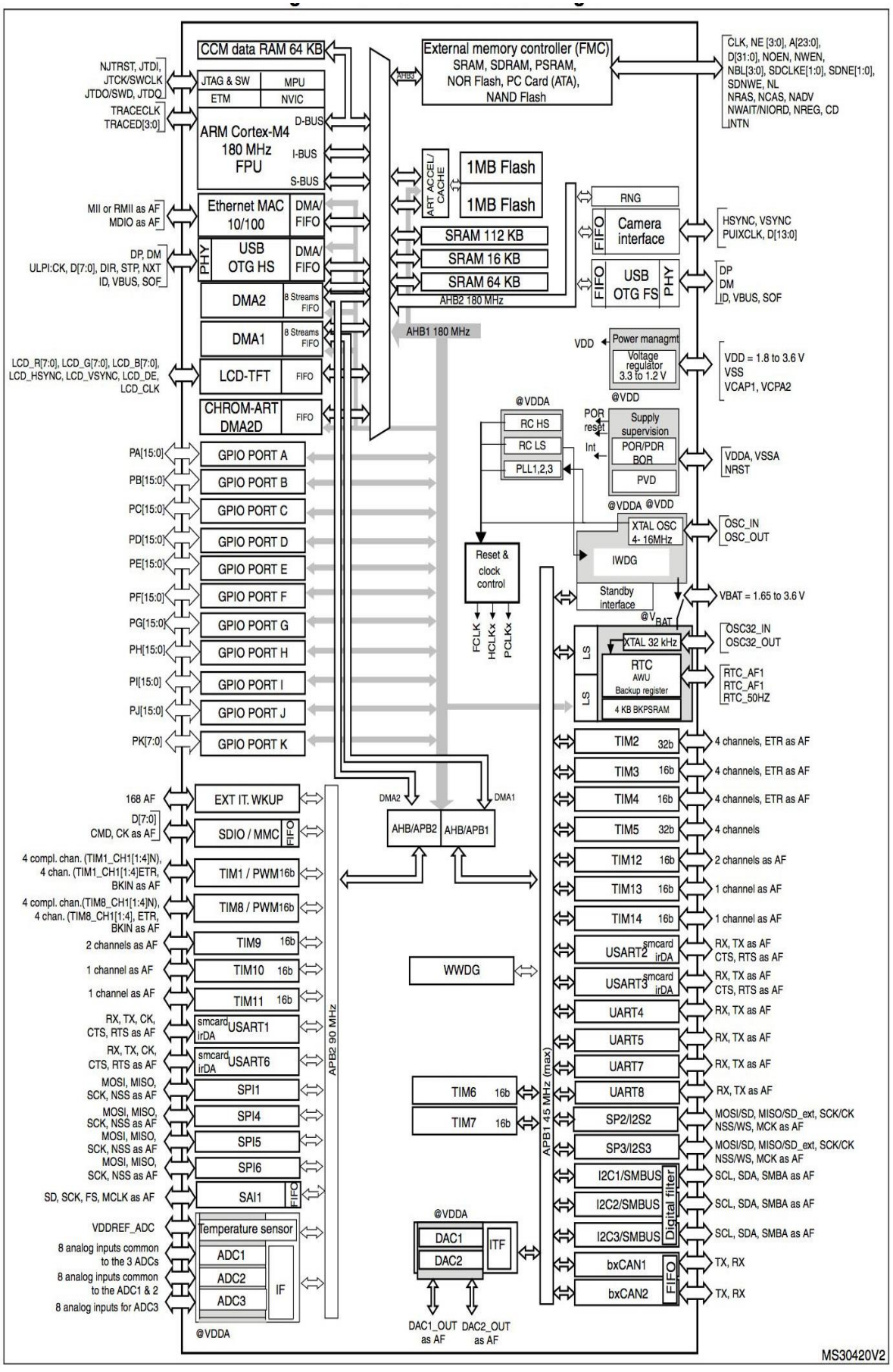

Figure 0.2 STM32f429xx Block Diagram

After choosing the MCU, it's the time to choose the on-board devices that will be used to integrate the system. Now, we are searching for devices that has three major features:

1. **Low power consumption**
2. **Output with High Accuracy**
3. **Low Cost**

Based on these features we start our journey of searching for these components, and after taking our time and see lots of components and see its rate we have chosen our components that are:

ESP8266

ESP8266 is a low-cost, highly integrated Wi-Fi microchip that is developed by Espressif Systems. It is designed for Internet of Things (IoT) applications and allows devices to connect to Wi-Fi networks, send and receive data over the internet, and communicate with other devices over a Wi-Fi network.

The ESP8266 microchip includes a powerful 32-bit RISC processor, 64KB of instruction RAM, and up to 1MB of data flash memory. It also has an integrated Wi-Fi radio, which supports 802.11 b/g/n standards and operates in the 2.4 GHz frequency band.

It also has a built-in TCP/IP protocol stack, which provides a complete TCP/IP protocol suite for easy integration with other devices and systems.

Due to its low cost, small size, and easy-to-use programming interface, the ESP8266 is widely used in a variety of IoT applications, such as smart home devices, industrial automation, and sensor networks. It also has a range of development boards and modules available, which make it easy for developers to integrate the ESP8266 into their projects.

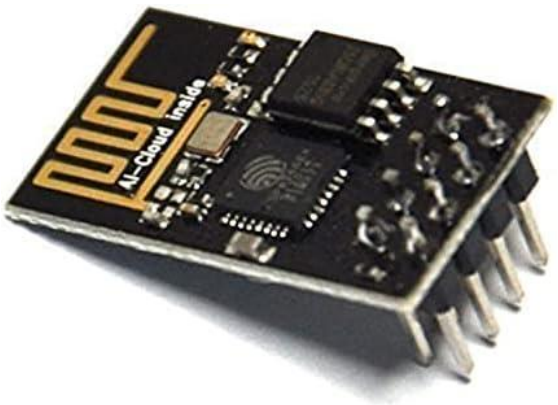

Figure 0.3 ESP8266

We use it as our port to the internet and the key of making bar metal speak to server. It can be connected to USART and talk to it using AT commands that will be discussed later on implementation chapter.

OV7670:

The OV7670 image sensor is a low voltage CMOS device that provides the full functionality of a single-chip VGA camera and image processor in a small footprint package. The OV7670 provides full-frame sub-sampled or windowed 8-bit images in a wide range of formats, controlled through the Serial Camera Control Bus (SCCB) interface.

It has an image array capable of operating at up to 30 fps in VGA with complete user control over image quality, formatting, and output data transfer. All required image processing functions are programmable through the SCCB interface. Including:

1. **Exposure control**
2. **Gamma**
3. **White balance**
4. **Color saturation**
5. **Hue control**

OV7670 camera module supports the following formats:

1. **Raw RGB**
2. **GRB 4:2:2**
3. **RGB565**
4. **RGB555**
5. **RGB444**
6. **YUV 4:2:2 / YCbCr 4:2:2**

We choose to use the OV7670 camera module in our system as it supports important features. Including:

1. **Automatic Exposure Control (AEC).**
2. **Automatic Gain Control (AGC).**
3. **Automatic White Balance (AWB).**
4. **Automatic Band Filter (ABF).**
5. **Automatic Black-Level Calibration (ABLC)**
6. **Saturation level auto adjust.**
7. **Edge enhancement level auto adjust.**
8. **De-noise level auto-adjust.**

These are some key specs of the OV7670 module:

Active Array Size	640 x 480
Power Supply: Digital Core	1.8VDC ± 10%
Power Supply: Analog	2.45 to 3.0V
Power Supply: I/O	1.7V to 3.0V
Power Requirement (Active)	60 mW
Operation Temperature Range	-30°C to 70°C
Lens Size	1/6”
Maximum Image Transfer Rate	30 fps for VGA

Table 0.1 OV7670 Specs

Functional block diagram:

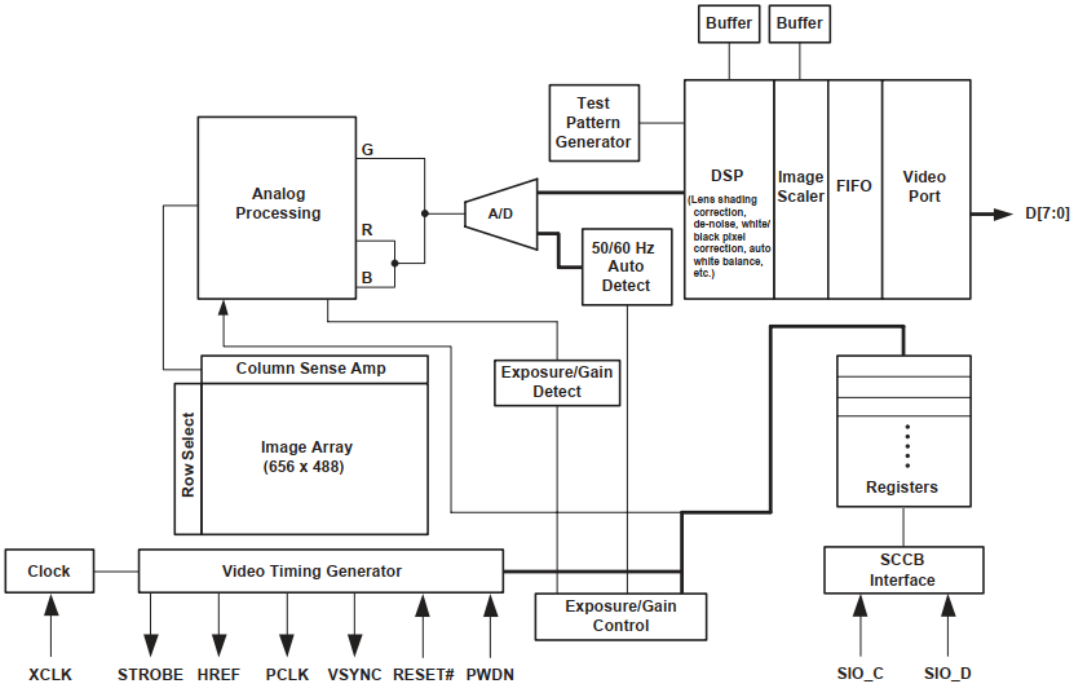

Figure 0.4 OV7670 Functional block diagram

In our system we choose this module as it has a lot of useful digital signal processing internally without the need to implement any algorithms ourselves.

We use it to take a periodic picture every 5 minutes of the farm and all the plants. Then send it to a server through the Wi-Fi module using MQTT protocol. Then, the server feeds the image to an AI model that will be able to detect if there are any pests or infections and notify the user about it if there is any.

In our case since we’re working with a microcontroller we are limited by our resources. The MCU we use has 256KB of SRAM. So, the Image taken needs to be smaller than that to leave space for other sections in memory like the BSS, DATA, STACK and HEAP.

So, in our system, we've configured the camera as follow:

1. **Format: YUV.**
2. **Resolution: 320x240 down sampled by 4 from VGA 640x480.**
3. **Gamma curve 2.2**
4. **We have enabled the following features:**
   - **4.1. AEC**
   - **4.2. AGC**
   - **4.3. AWB**
   - **4.4. De-noise**
   - **4.5. Edge enhancement**

These configurations of the format and the resolution will result in an image that takes 150KB of the 256KB SRAM. There will be enough space for the application.

We've also used the internal PLL of the OV7670 module as it produces better images than using the raw clock signal fed from our MCU to the module.

Water Pump and Hoses

This innovative device combines small size with powerful performance, making it ideal for a range of applications. With exceptional efficiency and adjustable flow control, the mini water pump adapts to diverse environments, whether for water features, hydroponic systems, or aquarium filtration.

Figure 0.5 Mini Water Pump

We've used two mini water pumps

VIII. **First one:** It is used for pumping the water from the tap or the main source of water to the water tank**.**

IX. **Second one:** It is used for pumping the water from the water tank to the main mini solenoid valve, then to each valve of specific plant.

Specs	Value
DC Voltage	2.5-6V
Working current	130-220mA
Power	0.4-1.5W
Maximum lift	40-110cm / 15.75"-43.4"
Flow rate	80-120L/H
Outside diameter of water outlet	approx. 7.5mm / 0.3"
Inside diameter of water outlet	approx. 4.7mm / 0.18"
Diameter	approx. 24mm / 0.95"
Length	approx. 45mm / 1.8"
Height	approx. 33mm / 1.30"
Wire length	about 15-20cm (red: " + ", black(white): " - ")
Material	plastic

Table 0.2 Water Pump Specs

Valve

I. Solenoid Valve ½”

An advanced solution for precise control of water flow. This innovative valve combines the convenience of plastic construction with the efficiency of electric operation, making it an ideal choice for a wide range of applications. The device supports the control of fluid flow using a computer or microcontroller, including Raspberry Pi or Arduino using the amplifier or relay. The device is powered with voltage of 12 V. The diameter of the threaded inputs for water is ½”. The minimum value of the pressure is 0.2 bar (20 kPa).

Figure 0.6 Solenoid Valve

We’ve used one of the solenoid valves ½” to control the water flow from the main water source to the water tank.

Specs	Value
Supply voltage	12 V
Current consumption	approx. 320 mA
Closing state	Closed in the rest position
Thread	1/2 “: 20 mm
Inner diameter	13 mm
Opening time	< 0.15 sec
Closing time	< 0.3 sec
Pressure range	0.0 to 0.05 MPa
Fluid temperature	0 to 100 °C

Figure 0.7 Solenoid Valve Specs

II. Solenoid Valve ¼”

A compact and versatile solution for efficient control of air and water flow. This innovative valve combines a small form factor with powerful functionality, making it an ideal choice for a wide range of applications. The device supports the control of fluid flow using a computer or microcontroller, including Raspberry Pi or Arduino using the amplifier or relay. The device is powered with voltage of 12 V. The diameter of the threaded inputs for water is ¼”.

Figure 0.8 Mini Solenoid Valve

We’ve used three of the solenoid valve ¼”

1. First one: It is used to control the water flow from the water tank to each valve for a specific plant.
2. Second one: It is used to control the water flow from the main ¼” valve to the valve of the first plant or column of plants.
3. Third one**:** It is used to control the water flow from the main ¼” valve to the valve of the second plant or column of plants.

Specs	Value
Rated voltage	DC 12V
Current	<150mA
Power	<2W
Pressure range	0-450mmhg
Use fluid	water/air
Service life	>50000 times
Working Mode	Normally Closed
Line Length	20cm
Weight	40g

Table 0.3 Mini Solenoid Valve Specs

Water Level Switch - Right Angle

A reliable and efficient solution for monitoring and controlling water levels in tanks and pools. This innovative float switch is designed with a right-angle configuration, allowing for easy installation and precise positioning within the tank or pool. It can activate a pump, an indicator, an alarm or other device.

Figure 0.9 Water Level Switch

We've used two of the right-angled water level float switches

III. **First one:** It used as high-level switch to indicate that the water tank is full or not.
IV. **Second one:** It used as low-level switch to indicate that the water tank is empty or not.

Specs	Value
Max Contact Rating	10W
Max Switching Voltage	100V DC
Max Switching Current	0.5A
Max Breakdown Voltage	220V DC
Max Carry Current	1.0A
Max Contact Resistance	100mΩ
Temperature Rating	-10 ~ +60 °C / +85 °C
Float Ball Material	PP
Float Ball Size	(Approx.) 10 x 25mm (L*D)
Thread Diameter	11mm
Hex Nut Diameter	16mm
Bottom Clip Diameter	(Approx.) 20mm
Body Total Size	54 x 25 x 28mm (L*W*H)

Table 0.4 Water Level Switch Specs

FC-28

It is a soil moisture detection sensor module. It's a low-tech sensor, but ideal for monitoring an urban garden, or your pet plant's water level. By utilizing the soil moisture detection sensor module, we can enhance our gardening or agricultural endeavors, achieving healthier plants and maximizing water efficiency. This sensor uses the two probes to pass current through the

soil, and then it reads that resistance to get the moisture level. More water makes the soil conduct electricity more easily (less resistance), while dry soil conducts electricity poorly (more resistance). It can output both analog and digital measurements, the digital measurement is done by using a comparator (LM393 IC).

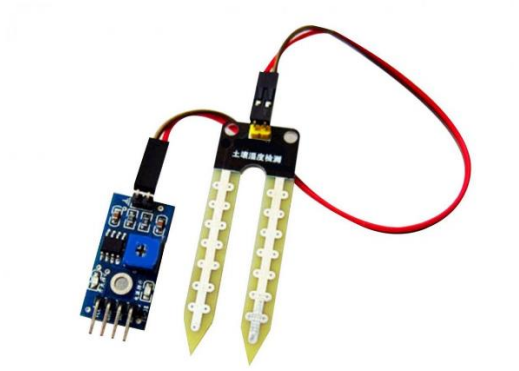

Figure 0.10 FC-28

We've used two of the FC-28 modules

I. **First one: It used to measure the soil moisture of the first plant.**
II. **Second one:** It used to measure the soil moisture of the second plant.

Specs	Value
Operating Voltage	3.3V to 5V DC
Operating Current	15mA
Output Digital	0V to 5V, Adjustable trigger level from preset
Output Analog	0V to 5V based on infrared radiation from fire flame falling on the sensor
LEDs	Indicating output and power
PCB Size	3.2cm x 1.4cm
Comparator	LM393 based design

Table 0.5 FC-28 Specs

LDR Module

It's a photosensitive resistor sensor. It's an essential component for light detection and measurement in various applications. This sensor module utilizes a light dependent resistor (LDR) to measure the intensity of light in its surroundings. It's easy to install. It comes with mounting hole screw and signal output indicator light. It can output both analog and digital measurements, the digital measurement is done by using a comparator (LM393 IC) and the sensitivity can be adjusted via a blue digital potentiometer adjustment.

Figure 0.11 LDR

We've used one of the LDR Modules to measure the intensity of light in its surroundings. This data is then utilized to dynamically adjust the LED grid, mimicking natural sunlight conditions for optimal plant growth. As the LDR sensors detect low light levels, such as during cloudy weather or in the evening, the LED grid automatically activates, providing supplementary light to the plants. Conversely, when the LDR sensors detect sufficient natural light, the LED grid dims or turns off to conserve energy.

Specs	Value
VCC	+VE power supply. Operating voltage: 3.3V-5V
GND	Ground
D0	Digital output signal
A0	Analog Output Signal
Strong light	LED light, output low level
Weak light	LED off, output high level

Table 0.6 LDR Specs

LED Grid

The LED grid serves as a dynamic lighting system, designed to provide tailored and precise illumination to plants within a smart farm model. By leveraging the power of light-emitting diodes (LEDs), this grid offers customizable spectra and intensity options, mimicking natural sunlight and promoting optimal growth conditions for a variety of crops. With its modular design and advanced control capabilities, the LED grid allows for fine-tuning of lighting parameters, enabling farmers, researchers, and enthusiasts to experiment with different light regimes and optimize plant growth.

Figure 0.12 LED Grid

We've used one of the LED strips to make a LED grid to provide tailored and precise illumination to plants in our smart farm.

Specs	Value
Voltage	DC 12V
Current	5A
LED Strip Model	2835
Total Power	60 W
LED Strip Length	5m
LEDs per Meter	120 LED/m
IP Code	IP20
Color	Warm White

Table 0.7 LED Grid Specs

AHT21B

The AHT21B is a kind of sensor of humidity and temperature with digital I^2C output. It can be applied to HVAC, dehumidifier, testing and inspection equipment, consumer products, automobiles, automatic control, data loggers, weather stations, home appliances, humidity control, medical and other application fields which need to detect and control temperature and humidity. AHT21B is equipped with a newly designed ASIC dedicated chip, an improved MEMS semiconductor capacitive humidity sensor component and a standard on-chip temperature sensor component. Its performance has been greatly improved and even beyond the reliability level of the previous generation of sensor, which make it more stable in the harsh conditions. Each sensor had been rigorously calibrated and tested. Due to the improvement and miniaturization of the sensors, it is more cost-effective.

Figure 0.13 AHT21B

We've used one of the AHT21B sensor to monitor the temperature and humidity of our farm.

Specs	Value
Supply voltage	DC: 2.2 - 5.5V
Scope of Work	Temperature: -40~+80°C. Humidity: 0~100%RH
Accuracy	Temperature: ±0.5℃. Humidity: ±3%RH (25℃)
Hysteresis	Temperature: ±0.1℃ Humidity: ±1%RH
Resolution	Temperature: 0.01℃ Humidity: 0.024%RH
Output signal	I²C signal
Packaging	blister tray

Table 0.8 AHT21B Specs

Power Supply Unit

It is the backbone of energy supply for powering up our smart farm systems. This robust and reliable power supply is specifically picked to meet the demanding energy requirements of a smart farm model. With its 12V output and 5A current capacity, it ensures a stable and consistent power flow to drive various components and devices within the system. It has auto-recovery after protection, universal AC input/full range, cooling by free air convection, LED power supply with a metal body for hidden installation for LED lighting, high efficiency, reliability, low energy consumption, compact size and light weight.

Figure 0.14 Power Supply

We've used one of the PSU to power up all the components and devices in our system like:

I. **MCU.**
II. **ESP.**
III. **OV7670.**
IV. **Mini Water Pump x2.**
V. **Valve: 1x valve ½" and 3x valve ¼".**
VI. **LED Grid.**

Specs	Value
Input Voltage	AC 100 – 220V 50/60Hz
Output Voltage	12V DC, 5A
Total Power	60 W
Protections	Overload, Over Voltage and Short Circuit
Shell Material	Metal Case and Aluminum Base
Color	Sliver

Table 0.9 Power Supply Specs

Electromechanical Switch

We choose the relay to provide seamless switching capabilities for a range of components in our smart farm. These relays serve as the perfect interface

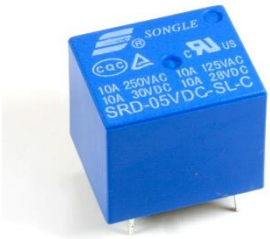

Figure 0.15 Relay

between MCU and the Mini Water Pump, Valve: valve ½", Valve: valve ¼", and LED Grid. With their ability to handle varying voltage and current requirements, these relays enable efficient control and operation of the connected devices. Their robust design ensures durability and longevity, making them suitable for both small-scale and industrial applications

We've used seven of the relays as our electromechanical switch to switch the following components:

A- Mini Water Pump x2.
B- Valve: 1x valve ½" and 3x valve ¼".
C- LED Grid.

Specs	Value
Rated load	10A 250VAC/28VDC,10A 125VAC/28VDC
Contact resistance	<=100m (ohm)
Electrical life	100,000
Mechanical life	10,000,000
Coil rated voltage	3-48VDC
Coil power	0.36W, 0.45W
Coil pick-up voltage	<=75%
Coil drop-out voltage	>=10%
Ambient temperature	-25 degrees Celsius to +70 degrees Celsius
Coil and contacts	1500VAC/min
Contact and contacts	1000VAC/min
Insulation resistance	>=100M ohm
Mounting form	PCB
Weight	10 g
Outline dimensions	19mm x 15.5mm x 15mm

Table 0.10 Electrical Switch Specs

2. Software design:

is the process of creating a plan or blueprint for a software system to meet specific requirements and objectives. It involves identifying the problem to be solved, analyzing the requirements, and then designing a solution that meets those requirements.

The main goal of software design is to create a high-quality software system that is reliable, efficient, maintainable, and scalable. We can divide this process into two subprocesses:

a- Static Design

Also known as structural design, is a type of software design that focuses on the overall structure and organization of a software system. It involves defining the static relationships and interactions between the different components of the system, including classes, objects, modules, and packages.

The primary goal of static design is to create a well-organized and modular software system that is easy to understand, maintain, and extend. This is achieved by defining a clear hierarchy of components and their relationships, as well as using standard design patterns and principles to ensure consistency and reusability.

Static design is typically done during the early stages of software development, before any code is written. It is often represented graphically using diagrams such as class diagrams, package diagrams, and component diagrams.

Some of the key principles and techniques used in static design include:

I. The Layered Architecture

is a type of software architecture that organizes the components of a software system into distinct layers, with each layer having a specific responsibility and interacting with adjacent layers through well-defined interfaces. This architecture is commonly used in enterprise applications, where the system needs to be scalable, modular, and maintainable.

The layered architecture typically consists of three or more layers, with each layer having a clear separation of concerns so that every layer has a specific responsibility.

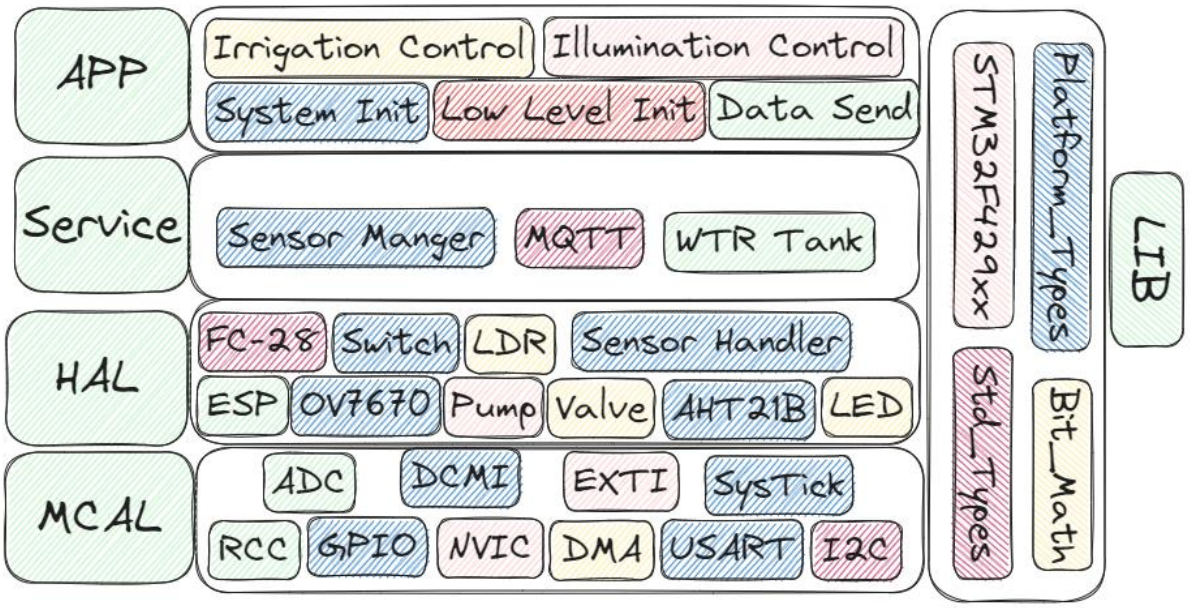

Table 0.11 Layered Architecture

II. Description For Each Module APIs and Types:

In this step we define each module APIs that will be used to communicate with the module by the others.

So, we will start with our components starting with MCAL layer to APP layer.

1. MCAL

a. RCC

i. Types

Name	Rcc_PeripheralId_t
Type	Enum
Description	The ID of peripheral

Table 0.12 RCC Types 1

Name	Rcc_ClkType_t
Type	Enum
Description	The system clock type

Table 0.13 RCC Types 2

Name	Rcc_PllType_t
Type	Enum
Description	The function type of PLL

Table 0.14 RCC Types 3

Name	Rcc_PllSrc_t
Type	Eanum
Description	PLL clock source

Table 0.15 RCC Types 3

Name	Rcc_PllConfig_t
Type	Struct
Description	PLL configurations

Table 0.16 RCC Types 4

ii. APIs

Name	Rcc_EnablePericlock
Syntax	Error_State_t Rcc_EnablePericlock(Rcc_PeripheralId_t Copy_PeripheralId, bool Copy_PeripheralClkMode).
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PeripheralId, Copy_PeripheralClkMode
Parameters (out)	None
Return Value	ErrorState_t
Description	Enable The clock to a peripheral and choose if in low power mode for the peripheral to continue working while in sleep mode.

Table 0.17 RCC APIs 1

Name	Rcc_DisablePericlock
Syntax	Error_State_t Rcc_DisablePericlock(Rcc_PeripheralId_t Copy_PeripheralId)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PeripheralId
Parameters (out)	None
Return Value	ErrorState_t
Description	Disable any clock connected to a peripheral.

Table 0.18 RCC APIs 2

Name	Rcc_SetClkState
Syntax	Error_State_t Rcc_SetClkState(Rcc_ClkType_t Copy_ClkType, bool Copy_ClkState, Rcc_PllConfig_t *Copy_PllConfigPtr)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ClkType, Copy_ClkState, Copy_PllConfigPtr
Parameters (out)	None
Return Value	ErrorState_t
Description	Turn on/off different clock sources.

Table 0.19 RCC APIs 3

Name	Rcc_SetSysClkSrc
Syntax	Error_State_t Rcc_SetSysClkSrc(Rcc_ClkType_t Copy_ClkType)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ClkType
Parameters (out)	None
Return Value	ErrorState_t
Description	Choose the source for the SYSCLK.

Table 0.20 RCC APIs 4

Name	Rcc_DisablePericlock
Syntax	Error_State_t Rcc_DisablePericlock(Rcc_PeripheralId_t Copy_PeripheralId)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PeripheralId
Parameters (out)	None
Return Value	ErrorState_t
Description	Disable any clock connected to a peripheral.

Table 0.21 RCC APIs 5

b. GPIO

i. Types

Name	Gpio_Port_t
Type	Enum
Description	GPIO Port number

Table 0.22 GPIO Types 1

Name	Gpio_PIN_t
Type	Enum
Description	GPIO Pin number

Table 0.23 GPIO Types 2

Name	Gpio_PinMode_t
Type	Enum
Description	GPIO Pin operation mode

Table 0.24 GPIO Types 3

Name	Gpio_PinOutput_t
Type	Enum
Description	GPIO Pin output mode

Table 0.25 GPIO Types 4

Name	Gpio_OutputSpeed_t
Type	Enum
Description	GPIO Pin output speed

Table 0.26 GPIO Types 5

Name	Gpio_PinPullUpDown_t
Type	Enum
Description	GPIO Pin pullup\down

Table 0.27 GPIO Types 6

Name	Gpio_PinAltFunOption_t
Type	Enum
Description	GPIO Pin Alternate function number

Table 0.28 GPIO Types 7

Name	Gpio_PinState_t
Type	Enum
Description	GPIO Pin state

Table 0.29 GPIO Types 8

Name	Gpio_PinConfig_t
Type	Struct
Description	GPIO Pin configuration

Table 0.30 GPIO Types 9

ii. APIs

Name	Gpio_PinInit
Syntax	ErrorState_t Gpio_PinInit(const Gpio_PinConfig_t *Copy_PinConfig)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PinConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	The Function Initializes the Required Pin Configuration options.

Table 0.31 GPIO APIs 1

Name	Gpio_SetPinValue
Syntax	ErrorState_t Gpio_SetPinValue(Gpio_Port_t Copy_Port,Gpio_PIN_t Copy_Pin,Gpio_PinState_t Copy_PinValue)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Port, Copy_Pin, Copy_PinValue
Parameters (out)	None
Return Value	ErrorState_t
Description	The function sets an output value to the required pin.

Table 0.32 GPIO APIs 2

Name	Gpio_SetPortValue
Syntax	ErrorState_t Gpio_SetPortValue(Gpio_Port_t Copy_Port,u16 Copy_PortValue).
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Copy_Port, Copy_PortValue
Parameters (out)	None
Return Value	ErrorState_t
Description	The function sets an output value to the required port.

Table 0.33 GPIO APIs 3

Name	Gpio_GetPinValue
Syntax	ErrorState_t Gpio_GetPinValue(Gpio_Port_t Copy_Port,Gpio_PIN_t Copy_Pin,Gpio_PinState_t* Copy_PinValue)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Port, Copy_Pin
Parameters (out)	Copy_PinValue
Return Value	ErrorState_t
Description	The function Reads an Input Value of the Required Pin.

Table 0.34 GPIO APIs 4

Name	Gpio_GetPortValue
Syntax	ErrorState_t Gpio_GetPortValue(Gpio_Port_t Copy_Port,u16* Copy_PortValue)
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Copy_Port
Parameters (out)	Copy_PortValue
Return Value	ErrorState_t
Description	The function reads an Input value of the required port.

Table 0.35 GPIO APIs 5

Name	Gpio_TogglePinValue
Syntax	ErrorState_t Gpio_TogglePinValue(Gpio_Port_t Copy_u8Port,Gpio_PIN_t Copy_Pin)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Port, Copy_Pin
Parameters (out)	None
Return Value	ErrorState_t
Description	The function toggles the value of output pin.

Table 0.36 GPIO APIs 6

c. NVIC

i. Types

Name	Nvic_IRQn_t
Type	Enum
Description	IRQ number

Table 0.37 NVIC Types 1

Name	Nvic_PrioGroup_t
Type	Enum
Description	Group and sub group priority

Table 0.38 NVIC Types 2

ii. APIs

Name	Nvic_EnableIRQ
Syntax	ErrorState_t Nvic_EnableIRQ(Nvic_IRQn_t Copy_IRQ)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	None
Return Value	ErrorState_t
Description	The function enables the required IRQ.

Table 0.39 NVIC APIs 1

Name	Nvic_DisableIRQ
Syntax	ErrorState_t Nvic_DisableIRQ(Nvic_IRQn_t Copy_IRQ)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	None
Return Value	ErrorState_t
Description	The function Disables the required IRQ.

Table 0.40 NVIC APIs 2

Name	Nvic_SetPendingIRQ
Syntax	ErrorState_t Nvic_SetPendingIRQ(Nvic_IRQn_t Copy_IRQ)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	None
Return Value	ErrorState_t
Description	The function sets the pending flag of the required IRQ by software.

Table 0.41 NVIC APIs 3

Name	Nvic_ClearPendingIRQ
Syntax	ErrorState_t Nvic_ClearPendingIRQ(Nvic_IRQn_t Copy_IRQ)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	None
Return Value	ErrorState_t
Description	The function resets the pending flag of the required IRQ by software.

Table 0.42 NVIC APIs 4

Name	Nvic_GetPendingIRQ
Syntax	ErrorStateErrorState_t Nvic_GetPendingIRQ(Nvic_IRQn_t Copy_IRQ, bool *pState)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	pState
Return Value	ErrorState_t
Description	The function gets the pending flag state of the required IRQ.

Table 0.43 NVIC APIs 5

Name	Nvic_GetActive
Syntax	ErrorState_t Nvic_GetActive(Nvic_IRQn_t Copy_IRQ, bool *pState)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	pState
Return Value	ErrorState_t
Description	The function gets the active flag state of the required IRQ.

Table 0.44 NVIC APIs 6

Name	Nvic_SetPriorityGrouping
Syntax	ErrorState_t Nvic_SetPriorityGrouping(Nvic_PrioGroup_t Copy_PrioGroup).
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PrioGroup
Parameters (out)	None
Return Value	ErrorState_t
Description	The function sets the group and sub-group numbers.

Table 0.45 NVIC APIs 7

Name	Nvic_SetPriority
Syntax	ErrorState_t Nvic_SetPriority(Nvic_IRQn_t Copy_IRQ, u8 Copy_Prio).
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Copy_IRQ, Copy_Prio
Parameters (out)	None
Return Value	ErrorState_t
Description	The function sets the priority of the required IRQ.

Table 0.46 NVIC APIs 8

Name	Nvic_GetPriority
Syntax	ErrorState_t Nvic_GetPriority(Nvic_IRQn_t Copy_IRQ, u8 *pPrio)
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	pPrio
Return Value	ErrorState_t
Description	The function gets the priority of the required IRQ.

Table 0.47 NVIC APIs 9

Name	Nvic_GenerateInterrupt
Syntax	ErrorState_t Nvic_GenerateInterrupt(Nvic_IRQn_t Copy_IRQ)
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Copy_IRQ
Parameters (out)	None
Return Value	ErrorState_t
Description	The function generates interrupt by software to the required IRQ.

Table 0.48 NVIC APIs 10

d. DMA

i. Types

Name	Dma_Num
Type	Enum
Description	DMA number

Table 0.49 DMA Types 1

Name	Dma_StreamNum
Type	Enum
Description	DMA Stream Number

Table 0.50 DMA Types 2

Name	Dma_ChannelNum
Type	Enum
Description	DMA Stream channel Number

Table 0.51 DMA Types 3

Name	Dma_StreamMode
Type	Enum
Description	DMA Stream Mode

Table 0.52 DMA Types 4

Name	Dma_StreamFifoTreshold
Type	Enum
Description	DMA Stream FIFO Threshold

Table 0.53 DMA Types 5

Name	Dma_ChannelMode
Type	Enum
Description	DMA Channel Mode

Table 0.54 DMA Types 6

Name	Dma_TransDirction
Type	Enum
Description	DMA Transfer Direction

Table 0.55 DMA Types 7

Name	Dma_StreamPriority
Type	Enum
Description	DMA Stream Priority

Table 0.56 DMA Types 8

Name	Dma_SrcDesState
Type	Enum
Description	DMA Source and Destination Increment state

Table 0.57 DMA Types 9

Name	Dma_PeripheralSize
Type	Enum
Description	DMA Peripheral Size

Table 0.58 DMA Types 10

Name	Dma_MemorySize
Type	Enum
Description	DMA Memory Size

Table 0.59 DMA Types 11

Name	Dma_Transfer_t
Type	Enum
Description	DMA Transfer type

Table 0.60 DMA Types 12

Name	Dma_Config_t
Type	Struct
Description	DMA configuration structure

Table 0.61 DMA Types 13

Name	Dma_StreamStartTrans
Type	Struct
Description	DMA Stream Transaction Configurations

Table 0.62 DMA Types 14

ii. APIs

Name	Dma_Init
Syntax	ErrorState_t Dma_Init(Dma_Config_t* Copy_DmaConfig)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_DmaConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the required DMA with required configuration.

Table 0.63 DMA APIs 1

Name	Dma_StreamStartSynch
Syntax	ErrorState_t Dma_StreamStartSynch(Dma_StreamStartTrans* Copy_StartTrans)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_StartTrans
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the transaction of DMA.

Table 0.64 DMA APIs 2

Name	Dma_StreamStartASynch
Syntax	ErrorState_t Dma_StreamStartASynch(Dma_StreamStartTrans* Copy_StartTrans, void(*Copy_NotificationFunc)(void)).
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_StartTrans, Copy_NotificationFunc
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the transaction of DMA.

Table 0.65 DMA APIs 3

Name	DMA_StopStream
Syntax	void DMA_StopStream(Dma_Num Copy_DmaNumber, Dma_StreamNum Copy_StreamNum)
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Copy_DmaNumber, Copy_StreamNum
Parameters (out)	None
Return Value	Void
Description	Stop the DMA Stream.

Table 0.66 DMA APIs 4

e. ADC

i. Types

Name	Adc_Num
Type	Enum
Description	ADC Number

Table 0.67 ADC Types 1

Name	Adc_ChannelNum
Type	Enum
Description	ADC channel Number

Table 0.68 ADC Types 2

Name	Adc_Resolution
Type	Enum
Description	ADC Resolution

Table 0.69 ADC Types 3

Name	Adc_TriggerType
Type	Enum
Description	ADC Trigger Type

Table 0.70 ADC Types 4

Name	Adc_ExtTriggerSense
Type	Enum
Description	ADC External Trigger edge Sense

Table 0.71 ADC Types 5

Name	Adc_RegularExtTrigger
Type	Enum
Description	ADC Regular Channel External Trigger Source

Table 0.72 ADC Types 6

Name	Adc_InjectedExtTrigger
Type	Enum
Description	ADC Injected Channel External Trigger Source

Table 0.73 ADC Types 7

Name	Adc_DataAlignment
Type	Enum
Description	ADC Output Alignment

Table 0.74 ADC Types 8

Name	Adc_ChannelSampleTime
Type	Enum
Description	ADC Sampling Clock Cycles

Table 0.75 ADC Types 9

Name	Adc_ChannelType
Type	Enum
Description	ADC Channel Type Regular / Injected

Table 0.76 ADC Types 10

Name	Adc_ChainConvType
Type	Enum
Description	ADC Group Conversion Type

Table 0.77 ADC Types 11

Name	Adc_ConversionConfig_t
Type	Struct
Description	ADC Channel Conversion Configurations

Table 0.78 ADC Types 12

Name	Adc_ChainConvConfig_t
Type	Struct
Description	ADC Chain of Channels Conversion Configurations

Table 0.79 ADC Types 13

ii. APIs

Name	Adc_Init
Syntax	ErrorState_t Adc_Init(Adc_Num Copy_Adc, Adc_Resolution Copy_AdcRes, Adc_DataAlignment Copy_DataAlignment)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Adc, Copy_AdcRes, Copy_DataAlignment
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Initialize the ADC.

Table 0.80 ADC APIs 1

Name	Adc_StartConversionSynch
Syntax	ErrorState_t Adc_StartConversionSynch(Adc_ConversionConfig_t* Copy_ConvConfig, u16* Copy_Reading)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ConvConfig
Parameters (out)	Copy_Reading
Return Value	ErrorState_t
Description	Convert channel in a synchronous way.

Table 0.81 ADC APIs 2

Name	Adc_StartConversionAsynch
Syntax	ErrorState_t Adc_StartConversionAsynch(Adc_ConversionConfig_t* Copy_ConvConfig, u16* Copy_Reading, void(*Copy_NotificationFunc)(void))
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ConvConfig, Copy_NotificationFunc
Parameters (out)	Copy_Reading
Return Value	ErrorState_t
Description	Convert channel in an asynchronous way.

Table 0.82 ADC APIs 3

Name	Adc_StartChainConversionSynch
Syntax	ErrorState_t Adc_StartChainConversionSynch (Adc_ChainConvConfig_t* Copy_ChainConfig, u16* Copy_Reading)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ChainConfig
Parameters (out)	Copy_Reading
Return Value	ErrorState_t
Description	Convert chain of channels in a synchronous way.

Table 0.83 ADC APIs 4

Name	Adc_StartChainConversionAsynch
Syntax	ErrorState_t Adc_StartChainConversionAsynch (Adc_ChainConvConfig_t* Copy_ChainConfig, u16* Copy_Reading, void(*Copy_NotificationFunc)(void))
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ChainConfig, Copy_NotificationFunc
Parameters (out)	Copy_Reading
Return Value	ErrorState_t
Description	Convert chain of channels in an asynchronous way.

Table 0.84 ADC APIs 5

f. DCMI

i. Types

Name	Dcmi_InterruptId_t
Type	Enum
Description	DCMI interrupt source

Table 0.85 DCMI Types 1

Name	Dcmi_InterruptState_t
Type	Enum
Description	DCMI interrupt state

Table 0.86 DCMI Types 2

ii. APIs

Name	Dcmi_Init
Syntax	`void Dcmi_Init(void)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	Void
Description	initialize the DCMI peripheral.

Table 0.87 DCMI APIs 1

Name	Dcmi_CaptureImage
Syntax	`void Dcmi_CaptureImage(void)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	Void
Description	start the capture of frame.

Table 0.88 DCMI APIs 2

Name	Dcmi_Init
Syntax	`ErrorState_t Dcmi_ControlInt(Dcmi_InterruptId_t Copy_IntId, Dcmi_InterruptState_t Copy_IntState, void(*Copy_CallBackFunc)(void))`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_IntId, Copy_IntState, Copy_CallBackFunc`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	control the state of the interrupts.

Table 0.89 DCMI APIs 3

g. EXTI

i. Types

Name	Exti_Port_t
Type	Enum
Description	The EXTI Pin Port

Table 0.90 EXTI Types 1

Name	Exti_Pin_t
Type	Enum
Description	The EXTI Pin

Table 0.91 EXTI Types 2

Name	Exti_Trigger_t
Type	Enum
Description	EXTI Trigger Type

Table 0.92 EXTI Types 3

Name	Exti_PinConfig_t
Type	Struct
Description	EXTI Pin Configuration

Table 0.93 EXTI Types 4

ii. APIs

Name	Exti_PinInit
Syntax	`ErrorState_t Exti_PinInit(const Exti_PinConfig_t* Copy_PinConfig).`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_PinConfig`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Initialize the pin as EXTI pin.

Table 0.94 EXTI APIs 1

Name	Exti_IntEnable
Syntax	`void Exti_IntEnable(Exti_Pin_t Copy_Pin)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_Pin`
Parameters (out)	None
Return Value	`Void`
Description	Enable The External Interrupt.

Table 0.95 EXTI APIs 2

Name	Exti_IntDisable
Syntax	`void Exti_IntDisable(Exti_Pin_t Copy_Pin)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Pin
Parameters (out)	None
Return Value	Void
Description	Disable The External Interrupt.

Table 0.96 EXTI APIs 3

h. USART

i. Types

Name	Usart_Number_t
Type	Enum
Description	The required USART number to be configured

Table 0.97 USART Types 1

Name	Usart_DataLength_t
Type	Enum
Description	USART Data Length to be transmitted / received

Table 0.98 USART Types 2

Name	Usart_Parity_t
Type	Enum
Description	USART Parity Check State

Table 0.99 USART Types 3

Name	Usart_mode_t
Type	Enum
Description	USART Mode Synch/Asynch

Table 0.100 USART Types 4

Name	Usart_StopBit_t
Type	Enum
Description	USART Number of Stop Bits

Table 0.101 USART Types 5

Name	Usart_DataTransfer_t
Type	Enum
Description	USART Transfer Data type

Table 0.102 USART Types 6

Name	Usart_config_t
Type	Struct
Description	USART Configuration structure

Table 0.103 USART Types 7

ii. APIs

Name	Usart_Init
Syntax	`ErrorState_t Usart_Init(Usart_config_t* Copy_config)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_config`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	initialize USART With Required Configuration.

Table 0.104 USART APIs 1

Name	Usart_SendCharSynch
Syntax	`ErrorState_t Usart_SendCharSynch(Usart_Number_t Copy_UsartNum, u16 Copy_Data)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_Data`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Send a character through USART in Synchronous way.

Table 0.105 USART APIs 2

Name	Usart_SendCharASynch
Syntax	`ErrorState_t Usart_SendCharASynch(Usart_Number_t Copy_UsartNum, u16 Copy_Data , void (*Copy_NotificationFunc)(void))`
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_Data, Copy_NotificationFunc`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Send a character through USART in an asynchronous way.

Table 0.106 USART APIs 3

Name	Usart_SendStringSynch
Syntax	`ErrorState_t Usart_SendStringSynch(Usart_Number_t Copy_UsartNum, char* Copy_String)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_String`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Send a String through USART in synchronous way.

Table 0.107 USART APIs 4

Name	Usart_SendStringASynch
Syntax	`ErrorState_t Usart_SendStringASynch(Usart_Number_t Copy_UsartNum, char* Copy_String, void (*Copy_NotificationFunc)(void))`
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_String, Copy_NotificationFunc`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Send a String through USART in an asynchronous way.

Table 0.108 USART APIs 5

Name	Usart_SendBufferSynch
Syntax	`ErrorState_t Usart_SendBufferSynch(Usart_Number_t Copy_UsartNum, u8* Copy_Buffer, u16 Copy_BufferLen)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_Buffer, Copy_BufferLen`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Send a Buffer through USART in Synchronous way.

Table 0.109 USART APIs 6

Name	Usart_SendBufferASynch
Syntax	`ErrorState_t Usart_SendBufferASynch(Usart_Number_t Copy_UsartNum, char* Copy_Buffer, u16 Copy_BufferLen, void (*Copy_NotificationFunc)(void))`
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_Buffer, Copy_BufferLen, Copy_NotificationFunc`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Send a Buffer through USART in an asynchronous way.

Table 0.110 USART APIs 7

Name	Usart_ReceiveCharSynch
Syntax	`ErrorState_t Usart_ReceiveCharSynch(Usart_Number_t Copy_UsartNum, u8* Copy_Data)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum`
Parameters (out)	`Copy_Data`
Return Value	`ErrorState_t`
Description	Receive a character through USART in a synchronous way.

Table 0.111 USART APIs 8

Name	Usart_ReceiveCharASynch
Syntax	`ErrorState_t Usart_ReceiveCharASynch(Usart_Number_t Copy_UsartNum, u16* Copy_Data, void (*Copy_NotificationFunc)(void))`
Sync/Async	Asynchronous
Reentrancy	Copy_Data
Parameters (in)	Copy_UsartNum, Copy_NotificationFunc
Parameters (out)	Copy_Reading
Return Value	ErrorState_t
Description	Receive a character through USART in an asynchronous way.

Table 0.112 USART APIs 9

Name	Usart_ReceiveBufferSynch
Syntax	`ErrorState_t Usart_ReceiveBufferSynch(Usart_Number_t Copy_UsartNum, u8* Copy_Buffer, u16 Copy_BufferSize)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_UsartNum, Copy_BufferSize
Parameters (out)	Copy_Buffer
Return Value	ErrorState_t
Description	Receive a Buffer through USART in a synchronous way.

Table 0.113 USART APIs 10

Name	Usart_ReceiveBufferASynch
Syntax	`ErrorState_t Usart_ReceiveBufferASynch(Usart_Number_t Copy_UsartNum, u8* Copy_Buffer, u16 Copy_BufferSize, void (*Copy_NotificationFunc)(void))`
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_UsartNum, Copy_BufferSize, Copy_NotificationFunc
Parameters (out)	Copy_Buffer
Return Value	ErrorState_t
Description	Receive a Buffer through USART in an asynchronous way.

Table 0.114 USART APIs 11

i. I2C

i. Types

Name	I2c_Id_t
Type	Enum
Description	I2C Number

Table 0.115 I2C Types 1

Name	I2c_Mode_t
Type	Enum
Description	I2C Mode of Operation

Table 0.116 I2C Types 2

Name	I2c_State_t
Type	Enum
Description	I2C State

Table 0.117 I2C Types 3

Name	I2c_Config_t
Type	Struct
Description	I2C Configuration Structure

Table 0.118 I2C Types 4

ii. APIs

Name	I2c_Init
Syntax	`ErrorState_t I2c_Init(I2c_Handle_t *hi2c)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	hi2c
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the I2C with required configuration.

Table 0.119 I2C APIs 1

Name	I2c_Master_Transmit
Syntax	`ErrorState_t I2c_Master_Transmit(I2c_Handle_t *hi2c, u16 Copy_DevAddress, u8 *pData, u16 Copy_Size, u32 Copy_Timeout)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	*hi2c, Copy_DevAddress, pData, Copy_Size, Copy_Timeout
Parameters (out)	None
Return Value	ErrorState_t
Description	Master transmits to slave.

Table 0.120 I2C APIs 2

Name	I2c_Master_Receive
Syntax	`ErrorState_t I2c_Master_Receive(I2c_Handle_t *hi2c, u16 Copy_DevAddress, u8 *pData, u16 Copy_Size, u32 Copy_Timeout)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	*hi2c, Copy_DevAddress, Copy_Size, Copy_Timeout
Parameters (out)	pData
Return Value	ErrorState_t
Description	Master Receives from slave.

Table 0.121 I2C APIs 3

Name	I2c_Mem_Write
Syntax	`ErrorState_t I2c_Mem_Write(I2c_Handle_t *hi2c, u16 Copy_DevAddress, u16 Copy_MemAddress, u16 Copy_MemAddSize, u8 *pData, u16 Copy_Size, u32 Copy_Timeout)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`*hi2c, Copy_DevAddress, Copy_MemAddress, Copy_MemAddSize, pData, Copy_Size, Copy_Timeout`
Parameters (out)	`None`
Return Value	`ErrorState_t`
Description	Write a specific memory in slave.

Table 0.122 I2C APIs 4

Name	I2c_Mem_Read
Syntax	`ErrorState_t I2c_Mem_Read(I2c_Handle_t *hi2c, u16 Copy_DevAddress, u16 Copy_MemAddress, u16 Copy_MemAddSize, u8 *pData, u16 Copy_Size, u16 Copy_Timeout)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`*hi2c, Copy_DevAddress, Copy_MemAddress, Copy_MemAddSize, Copy_Size, Copy_Timeout`
Parameters (out)	`pData`
Return Value	`ErrorState_t`
Description	Read a specific memory in slave.

Table 0.123 I2C APIs 5

Name	I2c_GetFlag
Syntax	`u8 I2c_GetFlag(I2c_Handle_t *hi2c, I2c_Flag_t Copy_Flag)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`hi2c, Copy_Flag`
Parameters (out)	`None`
Return Value	`U8`
Description	Get I2C Flag.

Table 0.124 I2C APIs 6

Name	I2c_Enable
Syntax	`ErrorState_t I2c_Enable(I2c_Handle_t* hi2c)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`hi2c`
Parameters (out)	`None`
Return Value	`ErrorState_t`
Description	Enable I2C.

Table 0.125 I2C APIs 7

Name	I2c_Disable
Syntax	`ErrorState_t I2c_Disable(I2c_Handle_t* hi2c)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	hi2c
Parameters (out)	None
Return Value	ErrorState_t
Description	Disable the I2C.

Table 0.126 I2C APIs 8

j. SysTick

i. Types

Name	SysTick_ClkSrc_t
Type	Enum
Description	SysTick Timer Clock source

Table 0.127 SysTick Types 1

Name	SysTick_Config_t
Type	Struct
Description	SysTick Timer configuration

Table 0.128 SysTick Types 2

ii. APIs

Name	SysTick_Init
Syntax	`void SysTick_Init(SysTick_Config_t *pConfig).`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	pConfig
Parameters (out)	None
Return Value	Void
Description	Enable The External Interrupt.

Table 0.129 SysTick APIs 1

Name	SysTick_GetTick
Syntax	`u32 SysTick_GetTick(void)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	U32
Description	gets the current tick count.

Table 0.130 SysTick APIs 2

Name	SysTick_IncTick
Syntax	`void SysTick_IncTick(void).`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	Void
Description	Increments a local static variable by 1.

Table 0.131 SysTick APIs 3

Name	SysTick_Delay
Syntax	`void SysTick_Delay(u32 Copy_DelayMs)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Copy_DelayMs
Parameters (out)	None
Return Value	Void
Description	blocking delay in ms.

Table 0.132 SysTick APIs 4

2. HAL

a. ESP

i. Types

Name	Esp_UsartNum
Type	Enum
Description	The USART Number ESP Connected to.

Table 0.133 ESP Types 1

Name	Esp_Recv_t
Type	Enum
Description	Receiving from ESP Type Sync/Asynch

Table 0.134 ESP Types 2

ii. APIs

Name	Esp_Init
Syntax	`ErrorState_t Esp_Init(Esp_UsartNum Copy_UsartNum)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_UsartNum
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the ESP module.

Table 0.135 ESP APIs 1

Name	Esp_ConnectWI-FI
Syntax	`ErrorState_t Esp_ConnectWI-FI(Esp_UsartNum Copy_UsartNum, char* Copy_Username, char* Copy_Password)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_Username, Copy_Password`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Connect ESP to Wi-Fi.

Table 0.136 ESP APIs 2

Name	Esp_ConnectServer
Syntax	`ErrorState_t Esp_ConnectServer(Esp_UsartNum Copy_UsartNum, char* Copy_ServerIp, char* Copy_ConnectionType, u16 Copy_PortNum)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_ServerIp, Copy_ConnectionType, Copy_PortNum`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Connect the ESP module to Server.

Table 0.137 ESP APIs 3

Name	Esp_SendData
Syntax	`ErrorState_t Esp_SendData(Esp_UsartNum Copy_UsartNum, u8* Copy_Data, u16 Copy_DataLength)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_Data, Copy_DataLength`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Send Data to the server.

Table 0.138 ESP APIs 4

Name	Esp_ReceiveData
Syntax	`ErrorState_t Esp_ReceiveData(Esp_UsartNum Copy_UsartNum, u8* Copy_Data, u16 Copy_DataLength, Esp_Recv_t Copy_RecvType, void (*Copy_NotificationFunc)(void))`
Sync/Async	Synchronous / Asynchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_DataLength, Copy_RecvType, Copy_NotificationFunc`
Parameters (out)	`Copy_Data`
Return Value	`ErrorState_t`
Description	Receive data from the server.

Table 0.139 ESP APIs 5

b. OV7670

i. Types

Name	Ov7670_I2cId_t
Type	Enum
Description	The I2C Number Camera Connected to.

Table 0.140 OV7670 Types 1

ii. APIs

Name	Ov7670_Init
Syntax	`ErrorState_t Ov7670_Init(I2c_Id_t Copy_I2cId)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_I2cId`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Initialization of the I2c and DCMI interfaces used by the camera and Initialization of the OV7670 internal registers.

Table 0.141 OV7670 APIs 1

Name	OV7670_SnapShot
Syntax	`ErrorState_t OV7670_SnapShot(void (*Frame_Cplt)(void))`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Frame_Cplt`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Capture an image via the DCMI interface and transfer the received frame to a buffer in SRAM via DMA.

Table 0.142 OV7670 APIs2

c. AHT21B

i. APIs

Name	Aht21b_Init
Syntax	`ErrorState_t Aht21b_Init(void)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Initialize the AHT21B Module.

Table 0.143 AHT21B APIs 1

Name	Aht21b_ReadTemperatureInCelsius
Syntax	`ErrorState_t Aht21b_ReadTemperatureInCelsius(s8 *Copy_Temperature)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	Copy_Temperature
Return Value	ErrorState_t
Description	Get the environment temperature in degree Celsius (from -40 to 80 C).

Table 0.144 AHT21B APIs 2

Name	Aht21b_ReadRelativeHumidity
Syntax	`ErrorState_t Aht21b_ReadRelativeHumidity(u8 *Copy_Humidity)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	Copy_Humidity
Return Value	ErrorState_t
Description	Get the environment relative humidity (from 0 to 100 %RH).

Table 0.145 AHT21B APIs 3

d. Pump

i. Types

Name	Pump_State_t
Type	Enum
Description	Pump state.

Table 0.146 Pump Types 1

Name	Pump_Activation_t
Type	Enum
Description	Pump Activation State.

Table 0.147 Pump Types 2

Name	Pump_Config_t
Type	Struct
Description	Pump configuration structure.

Table 0.148 Pump Types 3

ii. APIs

Name	Pump_TurnOn
Syntax	`ErrorState_t Pump_TurnOn(Pump_Config_t *Copy_PumpConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PumpConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Turn the mini water pump on.

Table 0.149 Pump APIs 1

Name	Pump_TurnOff
Syntax	`ErrorState_t Pump_TurnOff(Pump_Config_t *Copy_PumpConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PumpConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Turn the mini water pump off.

Table 0.150 Pump APIs 2

Name	Pump_GetState
Syntax	`ErrorState_t Pump_GetState(Pump_Config_t *Copy_PumpConfig, u8 *Copy_PumpState)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_PumpConfig
Parameters (out)	Copy_PumpState
Return Value	ErrorState_t
Description	Get the Pump current state (opened or closed).

Table 0.151 Pump APIs 3

e. Valve

i. Types

Name	Valve_State_t
Type	Enum
Description	Valve state.

Table 0.152 Valve Types 1

Name	Valve_Activation_t
Type	Enum
Description	Valve Activation State.

Table 0.153 Valve Types 2

Name	Valve_Config_t
Type	Struct
Description	Valve configuration structure.

Table 0.154 Valve Types 3

ii. APIs

Name	Valve_Open
Syntax	`ErrorState_t Valve_Open(Valve_Config_t *Copy_ValveConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ValveConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Open the valve to let the fluid go through.

Table 0.155 Valve APIs 1

Name	Valve_Close
Syntax	`ErrorState_t Valve_Close(Valve_Config_t *Copy_ValveConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ValveConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Close the valve to stop the fluid from going through.

Table 0.156 Valve APIs 2

Name	Valve_GetState
Syntax	`ErrorState_t Valve_GetState(Valve_Config_t *Copy_ValveConfig, u8 *Copy_ValveState)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_ValveConfig
Parameters (out)	Copy_ValveState
Return Value	ErrorState_t
Description	Get the valve current state (opened or closed).

Table 0.157 Valve APIs 3

f. Water Level Sensor

i. Types

Name	Switch_Type_t
Type	Enum
Description	Switch Type.

Table 0.158 Water Level Types 1

Name	Switch_Pull_t
Type	Enum
Description	Switch Pull Type.

Table 0.159 Water Level Types 2

Name	Switch_State_t
Type	Enum
Description	Switch State.

Table 0.160 Water Level Types 3

Name	Switch_Config_t
Type	Struct
Description	Switch Configuration structure.

Table 0.161 Water Level Types 4

Name	Switch_IntEnable_t
Type	Enum
Description	Pin Interrupt state.

Table 0.162 Water Level Switch Types 5

ii. APIs

Name	Switch_Init
Syntax	`ErrorState_t switch_Init(Switch_Config_t *Copy_SwitchConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_SwitchConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the Switch module.

Table 0.163 Water Level APIs 1

Name	Switch_GetStateDebounce
Syntax	`ErrorState_t Switch_GetStateDebounce(Switch_Config_t *Copy_SwitchConfig, Switch_State_t *Copy_SwitchState, Switch_Callback Delay, double Copy_DelayAmount)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_SwitchConfig, Delay, Copy_DelayAmount
Parameters (out)	Copy_SwitchState
Return Value	ErrorState_t
Description	Get the switch current state(pressed or not pressed) after debouncing.

Table 0.164 Water Level APIs 2

Name	Switch_GetState
Syntax	`ErrorState_t Switch_GetState(Switch_Config_t *Copy_SwitchConfig, Switch_State_t *Copy_SwitchState)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_SwitchConfig
Parameters (out)	Copy_SwitchState
Return Value	ErrorState_t
Description	Get the switch current state (pressed or not pressed.

Table 0.165 Water Level APIs 3

g. FC-28

i. Types

Name	Fc28_MoistureState_t
Type	Enum
Description	Sensor State.

Table 0.166 FC-28 Types 1

Name	Fc28_Config_t
Type	Struct
Description	Sensor configuration structure.

Table 0.167 FC-28 Types 2

Name	Fc28_DigPinIntEnable_t
Type	Enum
Description	Digital Pin Interrupt state.

Table 0.168 FC-28 Types

ii. APIs

Name	Fc28_Init
Syntax	`ErrorState_t Fc28_Init(Fc28_Config_t *Copy_Fc28Config)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Fc28Config
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the FC-28 module.

Table 0.169 FC-28 Types 1

Name	Fc28_GetSoilMoisturePercentage
Syntax	`ErrorState_t Fc28_GetSoilMoisturePercentage(Fc28_Config_t *Copy_Fc28Config, u8 *Copy_MoisturePercentage)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Fc28Config
Parameters (out)	Copy_MoisturePercentage
Return Value	ErrorState_t
Description	Get the soil moisture current percentage.

Table 0.170 FC-28 APIs 2

Name	Fc28_GetSoilMoistureStateDig
Syntax	`ErrorState_t Fc28_GetSoilMoistureStateDig(Fc28_Config_t *Copy_Fc28Config, Fc28_MoistureState_t *Copy_MoistureState)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_Fc28Config
Parameters (out)	Copy_MoistureState
Return Value	ErrorState_t
Description	Get the soil moisture current state using digital pins.

Table 0.171 FC-28 APIs 3

h. LDR

i. Types

Name	Ldr_IlluminationState_t
Type	Enum
Description	Sensor State.

Table 0.172 LDR Types 1

Name	Ldr_Config_t
Type	Struct
Description	Sensor configuration structure.

Table 0.173 LDR Types 2

Name	Ldr_DigPinIntEnable_t
Type	Struct
Description	Digital Pin Interrupt state.

Table 0.174 LDR Types

ii. APIs

Name	Ldr_Init
Syntax	`ErrorState_t Ldr_Init(Ldr_Config_t *Copy_LdrConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_LdrConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the LDR module.

Table 0.175 LDR Types 1

Name	Ldr_GetIlluminationPercentage
Syntax	`ErrorState_t Ldr_GetIlluminationPercentage(Fc28_Config_t *Copy_LdrConfig, u8 *Copy_ Copy_IlluminationPercentage)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_LdrConfig
Parameters (out)	Copy_IlluminationPercentage
Return Value	ErrorState_t
Description	Get the illumination current percentage.

Table 0.176 LDR APIs 2

Name	Ldr_GetIlluminationStateDig
Syntax	`ErrorState_t Ldr_GetIlluminationStateDig(Ldr_Config_t *Copy_LdrConfig, Ldr_IlluminationState_t *Copy_IlluminationState)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_LdrConfig
Parameters (out)	Copy_IlluminationState
Return Value	ErrorState_t
Description	Get the illumination current state using digital pins.

Table 0.177 LDR APIs 3

i. LED Grid

i. Types

Name	LedGrid_State_t
Type	Enum
Description	Led Grid state.

Table 0.178 LED Grid Types 1

Name	LedGrid_Activation_t
Type	Enum
Description	LED Grid Activation Type.

Table 0.179 LED Grid Types 2

Name	LedGrid_Config_t
Type	Enum
Description	Led Grid configuration structure.

Table 0.180 LED Grid Types 3

ii. APIs

Name	LedGrid_TurnOn
Syntax	`ErrorState_t LedGrid_TurnOn(LedGrid_Config_t *Copy_LedGridConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_LedGridConfig`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Turn the LED grid on.

Table 0.181 LED Grid APIs 1

Name	LedGrid_TurnOff
Syntax	`ErrorState_t LedGrid_TurnOff(LedGrid_Config_t *Copy_LedGridConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_LedGridConfig`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Turn the LED grid off.

Table 0.182 LED Grid APIs 2

Name	LedGrid_GetState
Syntax	`ErrorState_t LedGrid_GetState(LedGrid_Config_t *Copy_LedGridConfig, u8 *Copy_LedGridState)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_LedGridConfig
Parameters (out)	Copy_LedGridState
Return Value	ErrorState_t
Description	Get the LED grid current state (on or off).

Table 0.183 LED Grid APIs 3

3. **Service**
    a. **MQTT**
        i. **Types**

Name	Mqtt_UsartNum
Type	Enum
Description	The USART Number ESP Connected to.

Table 0.184 MQTT Types 1

Name	Mqtt_Qos_t
Type	Enum
Description	QOS of Message publish / Subscribe.

Table 0.185 MQTT Types 2

Name	Mqtt_Connect_t
Type	Struct
Description	ESP Connection configuration.

Table 0.186 MQTT Types 3

Name	Mqtt_EspConnection
Type	Struct
Description	ESP Connected to MCU configuration.

Table 0.187 MQTT Types 4

Name	Mqtt_Publish_t
Type	Struct
Description	The publish message configuration.

Table 0.188 MQTT Types 5

ii. APIs

Name	Mqtt_Connect
Syntax	`ErrorState_t Mqtt_Connect(Mqtt_UsartNum Copy_UsartNum ,Mqtt_Connect_t* Copy_Connect, Mqtt_EspConnection* Copy_EspConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_Connect, Copy_EspConfig`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Function initializes the whole Connection and Sends the Connect Packet to broker.

Table 0.189 MQTT APIs 1

Name	Mqtt_Publish
Syntax	`ErrorState_t Mqtt_Publish(Mqtt_UsartNum Copy_UsartNum, Mqtt_Publish_t* Copy_PubPacket)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_PubPacket`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Function Sends the publish Packet to broker.

Table 0.190 MQTT APIs 2

Name	Mqtt_Subscribe
Syntax	`ErrorState_t Mqtt_Subscribe(Mqtt_UsartNum Copy_UsartNum, char* Copy_TopicName, Mqtt_Qos_t Copy_MaxQos)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	`Copy_UsartNum, Copy_TopicName, Copy_MaxQos`
Parameters (out)	None
Return Value	`ErrorState_t`
Description	Function Sends the subscribe Packet to broker.

Table 0.191 MQTT APIs 3

b. WTR Tank

i. Types

Name	WtrTank_SwitchInitialState_t
Type	Enum
Description	Switch Initial State.

Table 0.192 WTR Tank Types 1

Name	WtrTank_State_t
Type	Enum
Description	Water Tank State.

Table 0.193 WTR Tank Types 2

Name	WtrTank_SwitchIntEnable_t
Type	Enum
Description	Switch Interrupt Enable.

Table 0.194 WTR Tank Types 3

Name	WtrTank_Config_t
Type	Struct
Description	Water tank configuration structure.

Table 0.195 WTR Tank Types 4

ii. APIs

Name	WtrTank_Init
Syntax	`ErrorState_t WtrTank_Init(WtrTank_Config_t Copy_WtrTankConfig)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_WtrTankConfig
Parameters (out)	None
Return Value	ErrorState_t
Description	Initialize the water tank module.

Table 0.196 WTR Tank APIs 1

Name	WtrTank_SetTankEmptyCallback
Syntax	`void WtrTank_SetTankEmptyCallback(void (*TankEmptyCallbackFunc)(void))`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	TankEmptyCallbackFunc
Parameters (out)	None
Return Value	Void
Description	Set callback function for empty water tank event.

Table 0.197 WTR Tank APIs 2

Name	WtrTank_SetTankFullCallback
Syntax	`void WtrTank_SetTankFullCallback(void (*TankFullCallbackFunc)(void))`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	TankFullCallbackFunc
Parameters (out)	None
Return Value	Void
Description	Set callback function for Full water tank event.

Table 0.198 WTR Tank APIs 3

Name	WtrTank_GetState
Syntax	`ErrorState_t WtrTank_GetState(WtrTank_Config_t *Copy_WaterTankConfig, WtrTank_State_t *Copy_WaterTankState)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	Copy_WaterTankConfig
Parameters (out)	Copy_WaterTankState
Return Value	ErrorState_t
Description	Get the water tank current state (full or empty).

Table 0.199 WTR Tank APIs 4

4. LIB

a. Platform Types

Name	u8 \ s8
Type	Permeative
Description	Type definition of unsigned \ signed char.

Table 0.200 Platform Types 1

Name	u16 \ s16
Type	Permeative
Description	Type definition of unsigned \ signed short.

Table 0.201 Platform Types 2

Name	U32 \ s32
Type	Permeative
Description	Type definition of unsigned \ signed long.

Table 0.202 Platform Types 3

Name	f32
Type	Permeative
Description	Type definition of float.

Table 0.203 Platform Types 4

Name	f64
Type	Permeative
Description	Type definition of double.

Table 0.204 Platform Types 5

b. STD Types

Name	Peripheral_State
Type	Enum
Description	Contains all states of peripherals.

Table 0.205 STD Types 1

Name	ISR_Src
Type	Enum
Description	Contains the possible ISR sources in the peripherals.

Table 0.206 STD Types 2

Name	Bool
Type	Enum
Description	Type definition for the Boolean.

Table 0.207 STD Types 3

Name	ErrorState_t
Type	Enum
Description	Contains all possible error states that can happen in peripherals.

Table 0.208 STD Types 4

b- Dynamic Design

Dynamic design, also known as behavioral design, is a type of software design that focuses on the dynamic behavior of a software system. It involves defining the behavior of the system in response to various events or stimuli, such as user input, external data, or system events.

The primary goal of dynamic design is to create a software system that executes correctly and efficiently in response to various scenarios and inputs. This is achieved by defining the actions and interactions of the different components of the system, including classes, objects, and modules, as well as the flow of data and control between them.

Some of the key principles and techniques used in dynamic design include:

I. State Machine Diagram

These are graphical representations of the possible states of a system and the transitions between them, which help to define the behavior of the system in response to different events. And will be done for two scopes.

1. ECU Components

a. MCAL

i. RCC

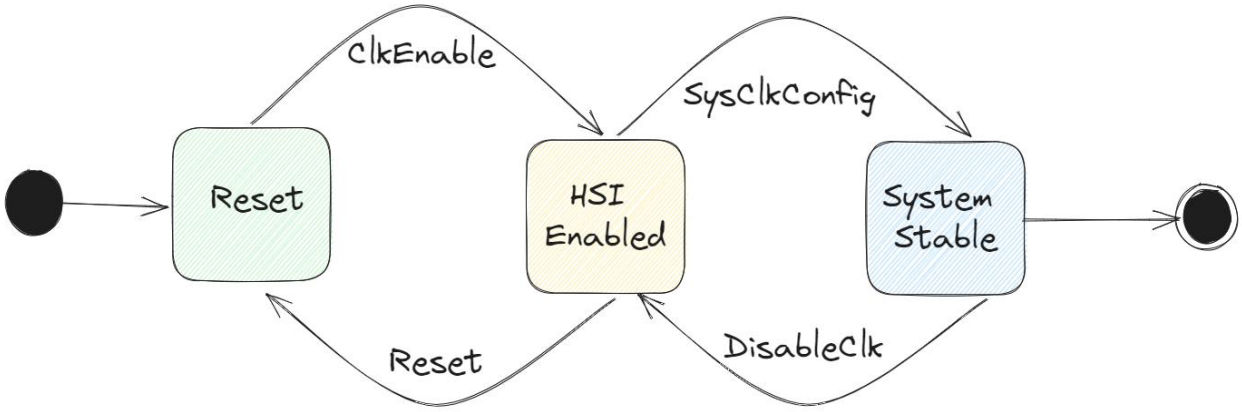

Figure 0.16 RCC State Machine

ii. GPIO

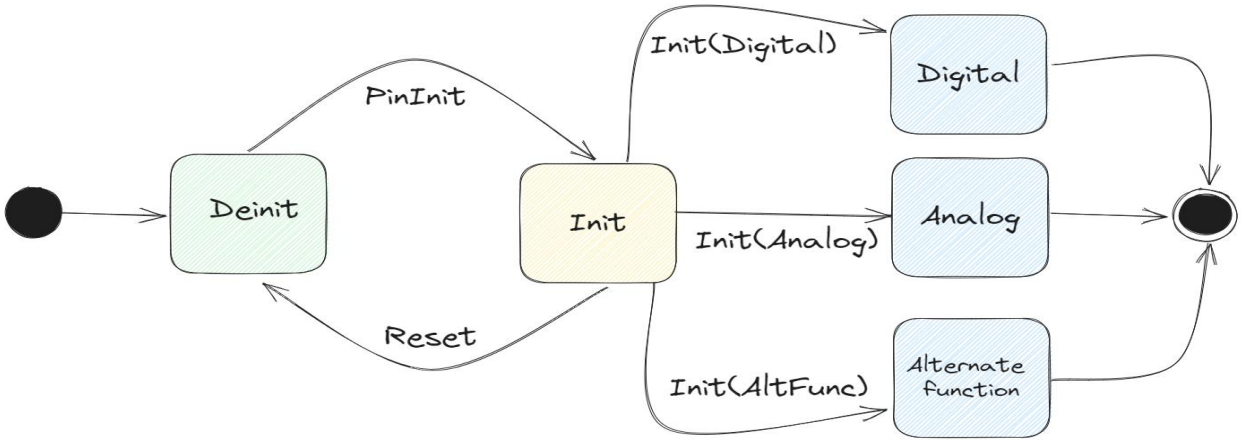

Figure 0.17 GPIO State Machine

iii. NVIC

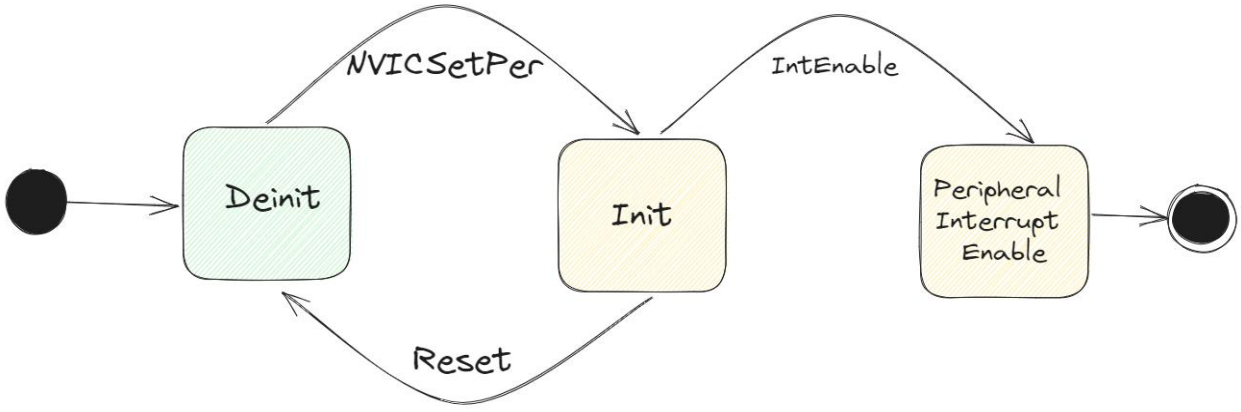

Figure 0.18 NVIC State Machine

iv. DMA

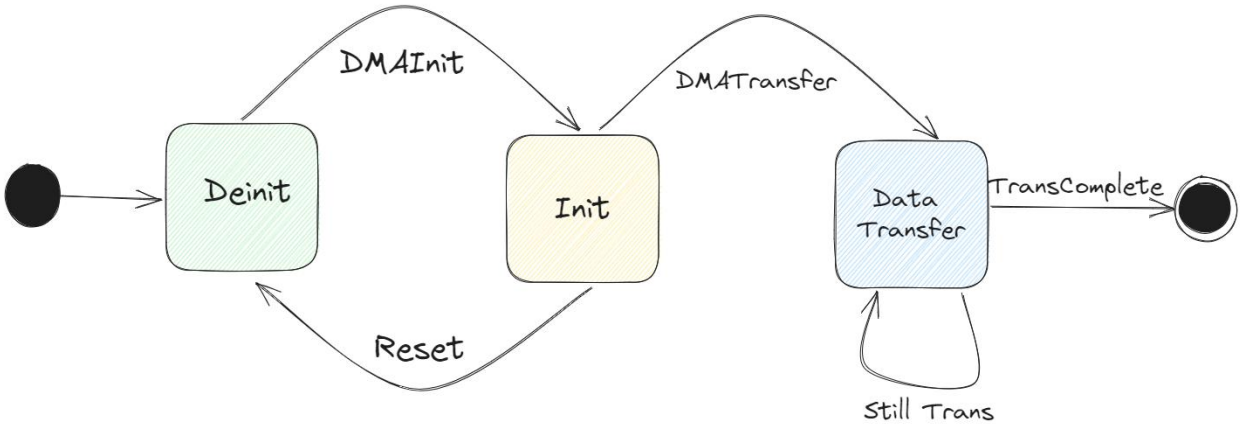

Figure 0.19 DMA State Machine

v. ADC

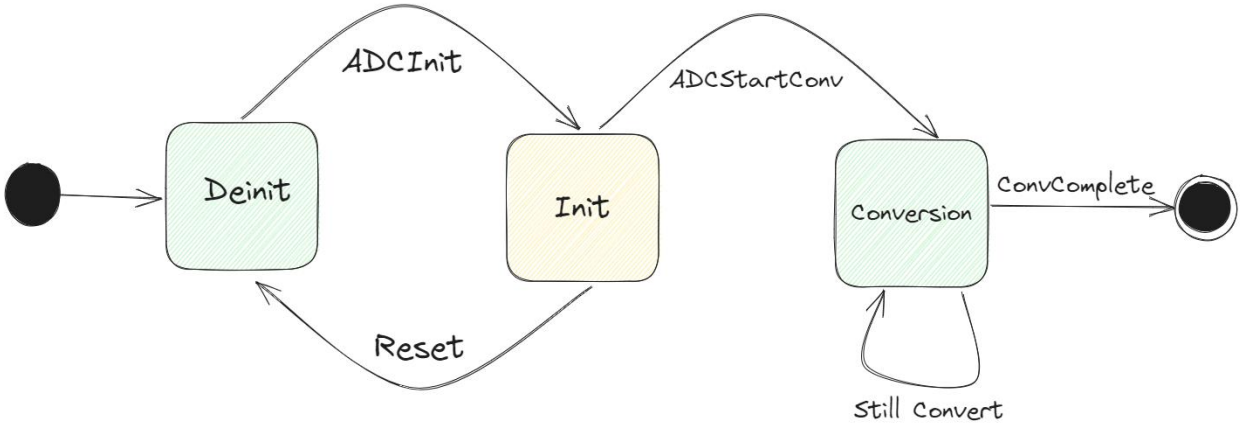

Figure 0.20 ADC State Machine

vi. DCMI

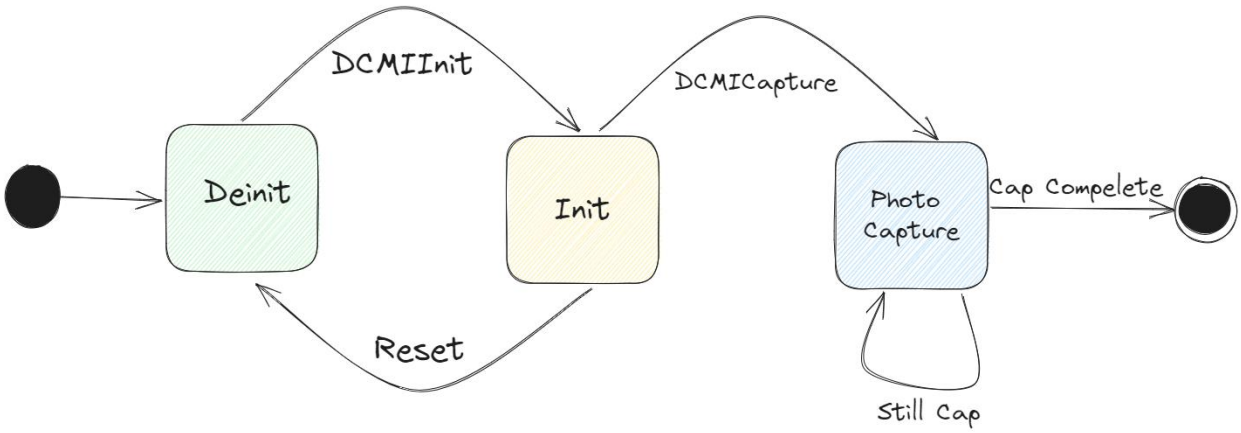

Figure 0.21 DCMI State Machine

vii. EXTI

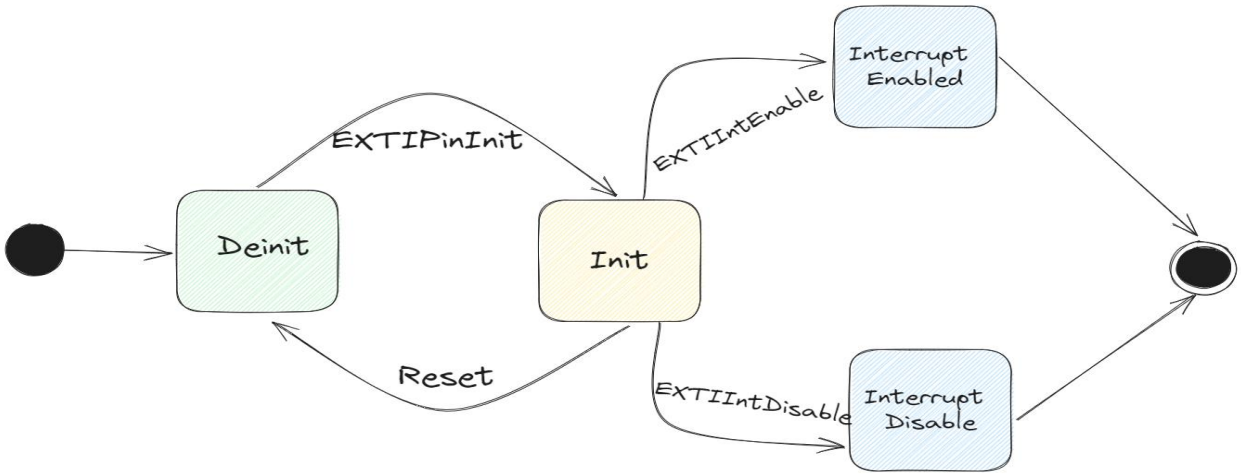

Figure 0.22 EXTI State Machine

viii. USART

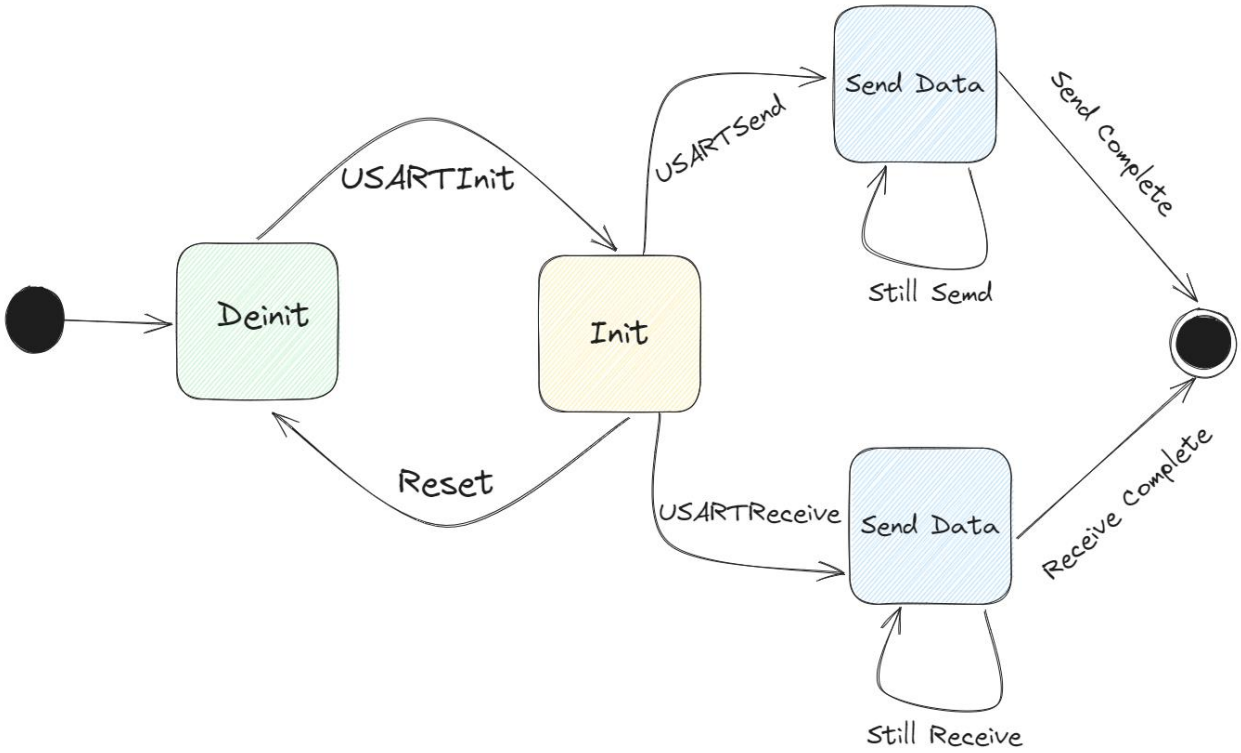

Figure 0.23 USART State Machine

ix. I2C

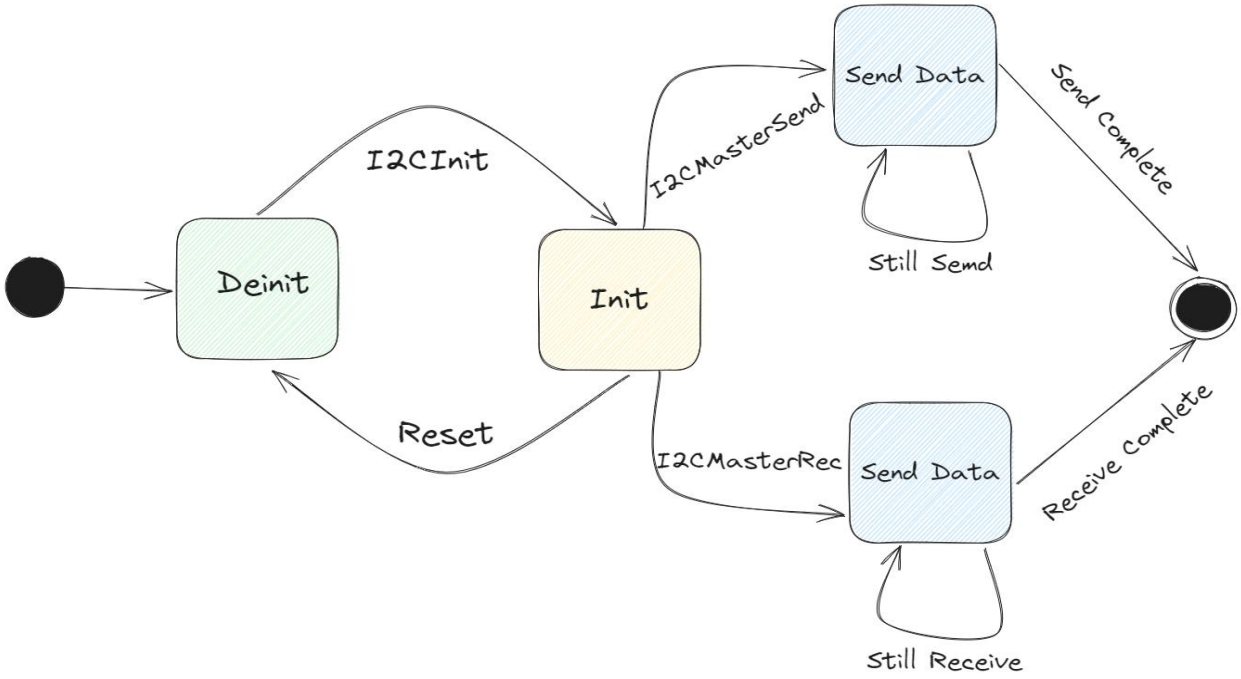

Figure 0.24 I2C State Machine

b. HAL

i. ESP

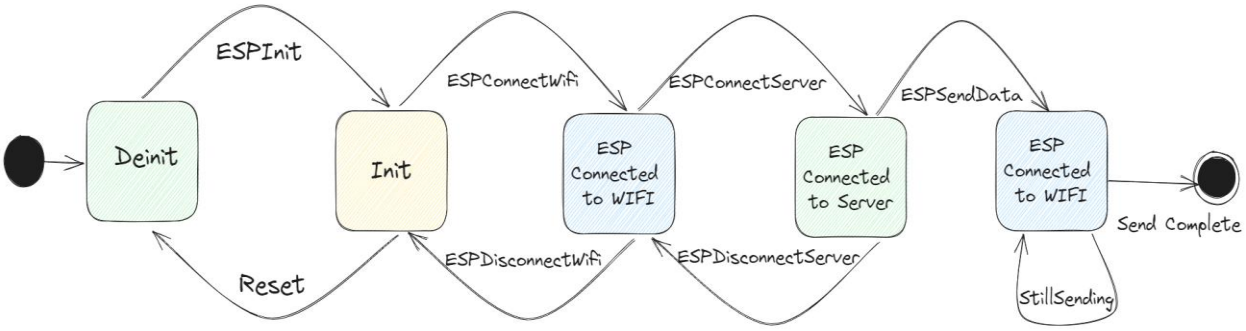

Figure 0.25 ESP State Machine

ii. OV7670

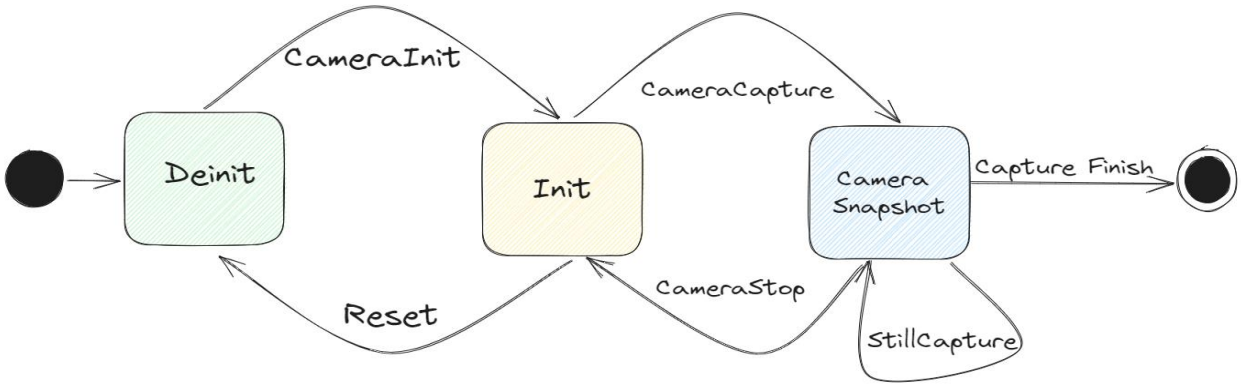

Figure 0.26 OV7670 State Machine

iii. AH21B

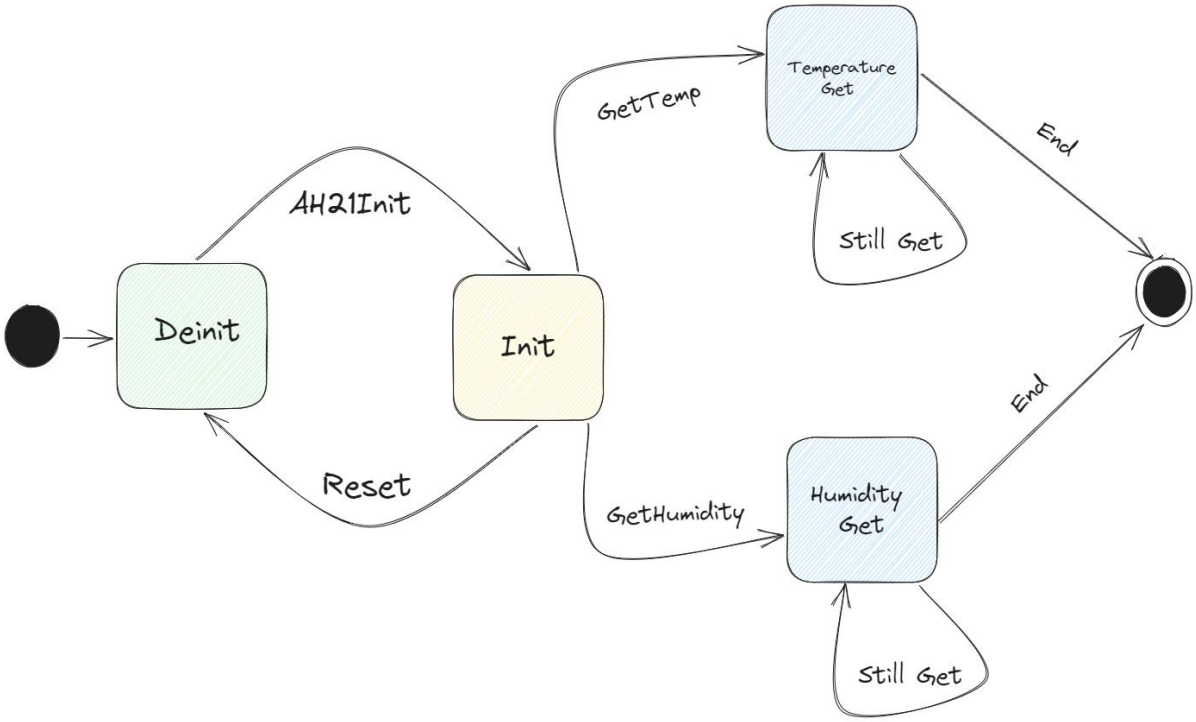

Figure 0.27 Ah21B State Machine

iv. Pump

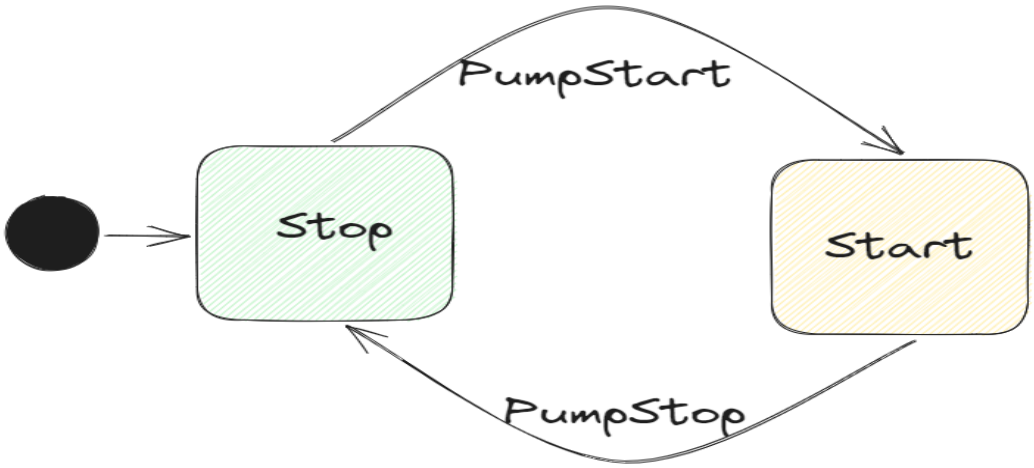

Figure 0.28 Pump State Machine

v. Valve

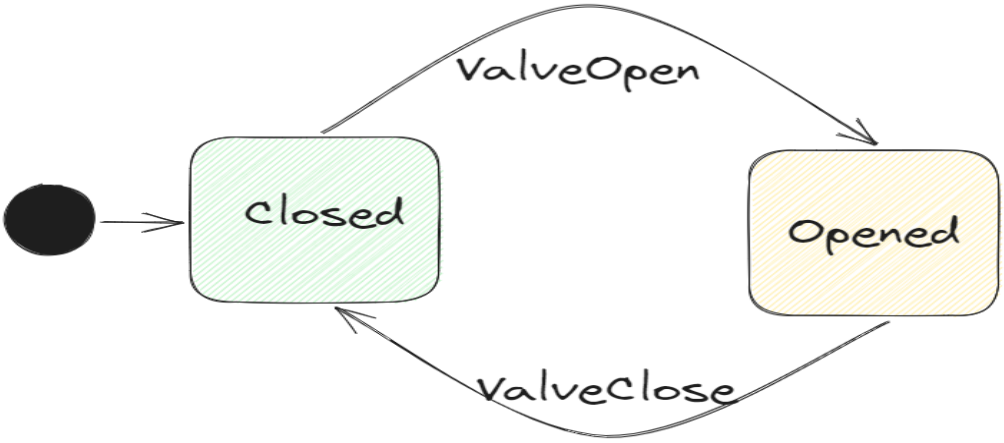

Figure 0.29 Valve State Machine

vi. Water Level switch

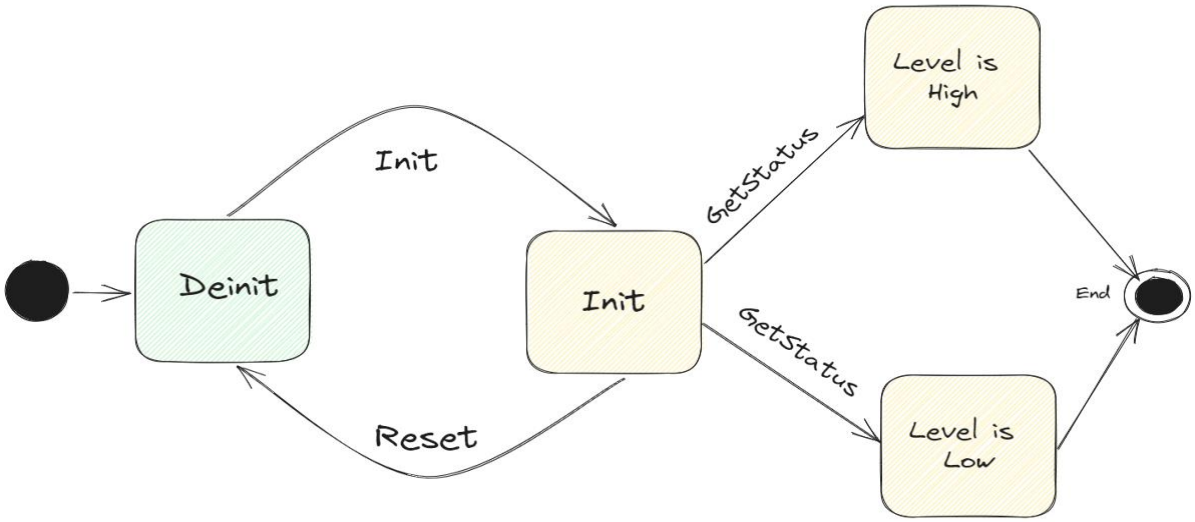

Figure 0.30 Water Level Switch State Machine

vii. FC-28

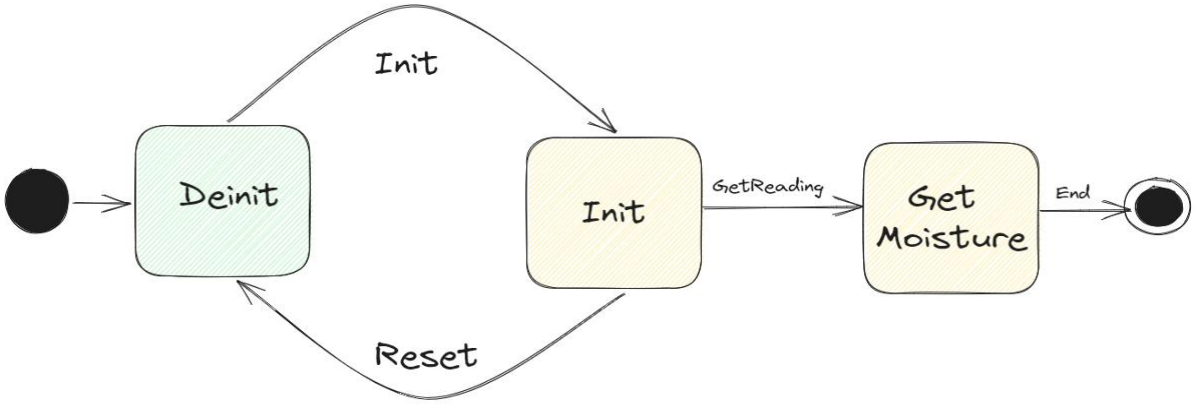

Figure 0.31 FC-28 State Machine

viii. LDR

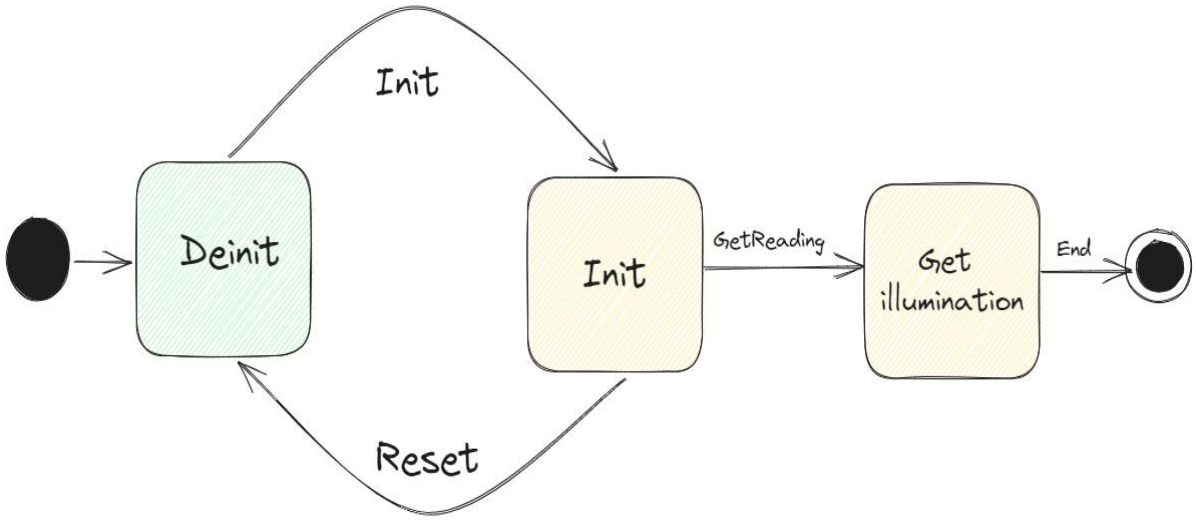

Figure 0.32 LDR State Machine

ix. LED Grid

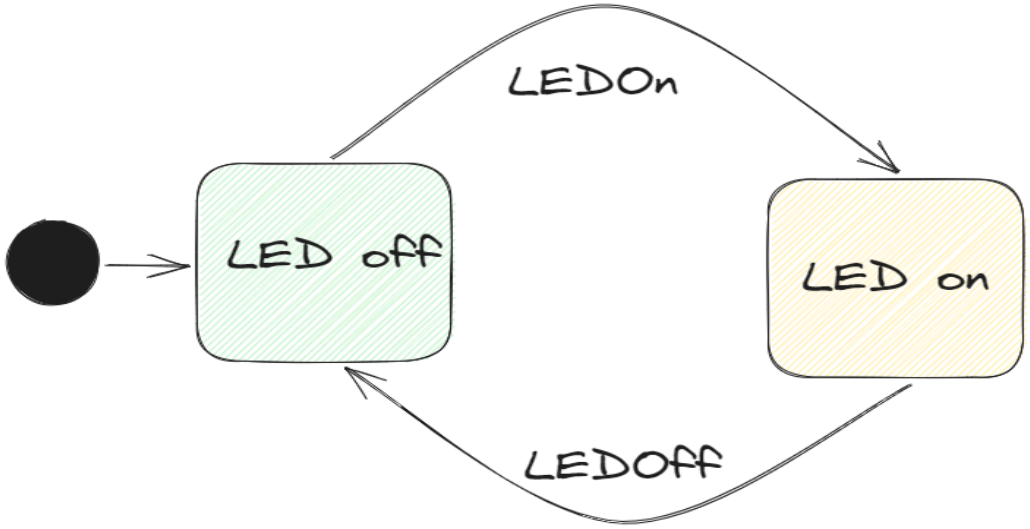

Figure 0.33 LED Grid State Machine

c. Service

i. MQTT

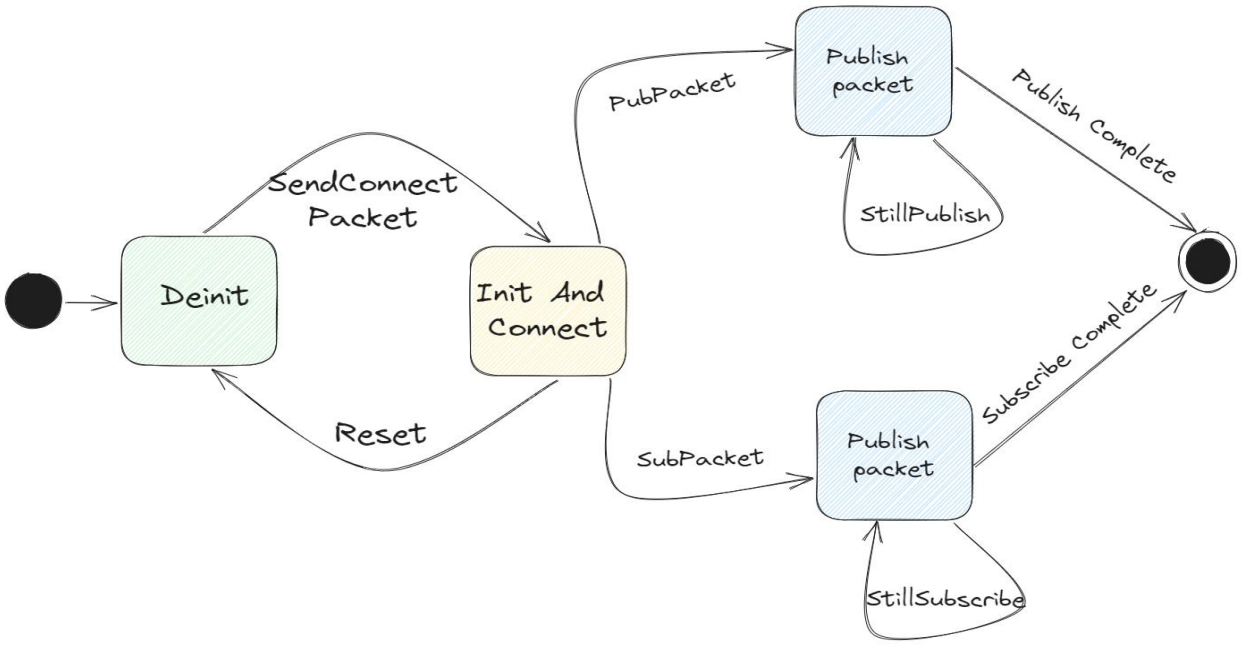

Figure 0.34 MQTT State Machine

2. ECU Operation

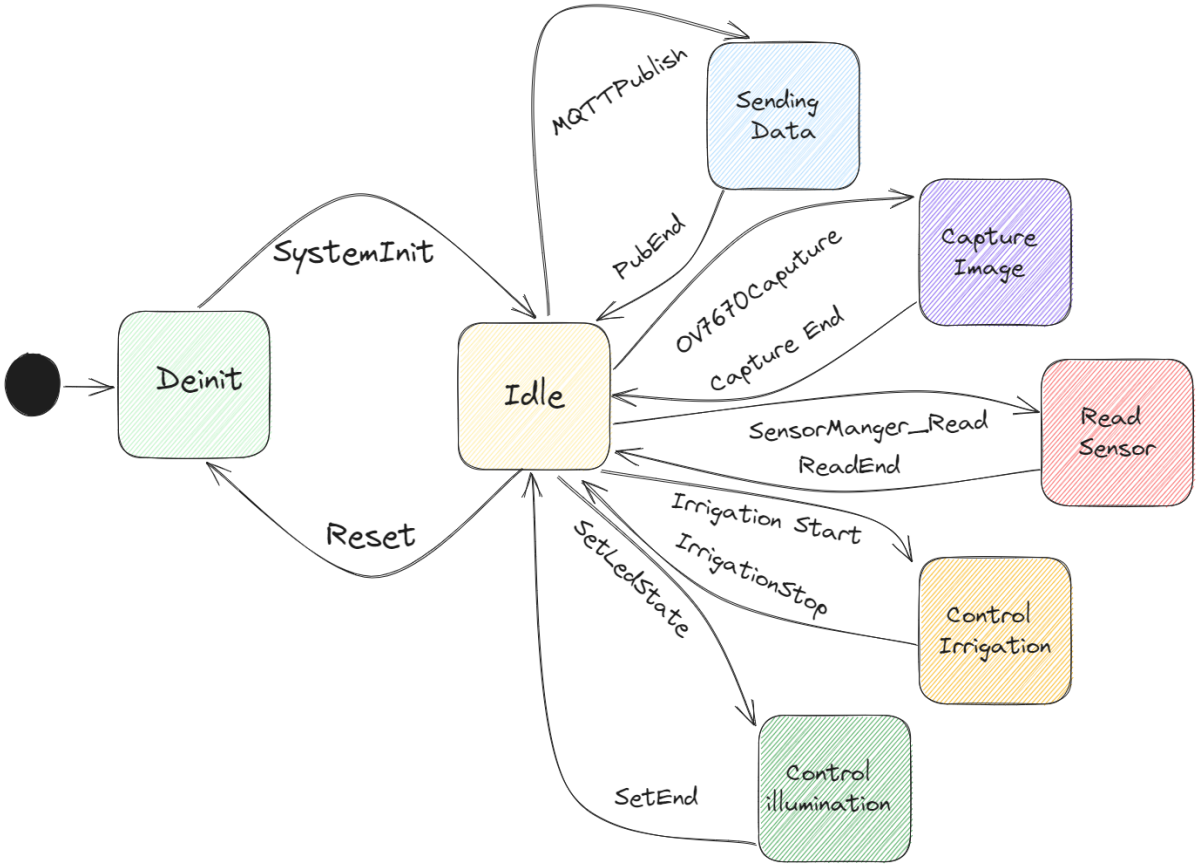

Figure 0.35 ECU Operation State Machine

Conclusion

In conclusion, both software and hardware design are critical components of any technology solution, and both play important roles in ensuring the success of a project.

Software design involves the creation of software applications, systems, and programs that enable devices to perform specific functions and interact with users. It involves the development of algorithms, coding, testing, and debugging to ensure that the software operates as intended and meets the desired requirements.

Hardware design, on the other hand, is the process of creating physical components and systems that enable devices to function. It involves the design of circuits, components, and devices, as well as the selection of materials, manufacturing processes, and testing procedures to ensure that the hardware performs as intended and meets the desired specifications.

Both software and hardware design are closely intertwined, as software relies on hardware to function, and hardware requires software to control and operate. Therefore, it is essential to consider both software and hardware design together when developing a technology solution.

Ultimately, successful software and hardware design requires a multidisciplinary approach, with collaboration between software engineers, hardware engineers, and other stakeholders to ensure that the solution meets the desired specifications and performs as intended.

Chapter 3: System Implementation

Intro

Now, it's time to take the software design and development artifacts and configuring them on the target hardware and software environment, installing and configuring any necessary third-party software, and testing the system to ensure that it meets the required specifications and quality standards.

The implementation phase typically follows the completion of the design and development phases of the software development life cycle (SDLC). During the implementation phase, the software is deployed and made available to users, and any necessary training and documentation are provided to support the users.

We will show our work in implementation of the system step by step by showing work in every peripheral by showing the peripherals, onboard devices, and services configurations used to make our system run. So, let's start.

1. **RCC**

   is responsible for managing the clock system of the device. The RCC is primarily used to set up the clock sources, configure the PLL (Phase Locked Loop), and configure the clock distribution to the various peripherals and modules in the microcontroller.

   The RCC provides a range of features, including:

   a. **Clock source selection:**
      The RCC allows you to select the clock source for the microcontroller. The available clock sources include the internal HSI (High-Speed Internal), external HSE (High-Speed External), and low-speed LSE (Low-Speed External) oscillators.

   b. **PLL configuration:**

      The RCC allows you to configure the PLL, which is used to generate a high-frequency clock from the input clock source. The PLL can be used to generate a clock frequency up to 180 MHz, which is the maximum frequency supported by the STM32F429ZI microcontroller.

   c. **Clock distribution:**

      The RCC allows you to configure the clock distribution to the various peripherals and modules in the microcontroller. This includes configuring the clock frequency for the CPU, peripherals, timers, and other modules.

   Overall, the RCC is an essential peripheral in the STM32F429ZI microcontroller that enables you to configure and manage the clock system of the device.

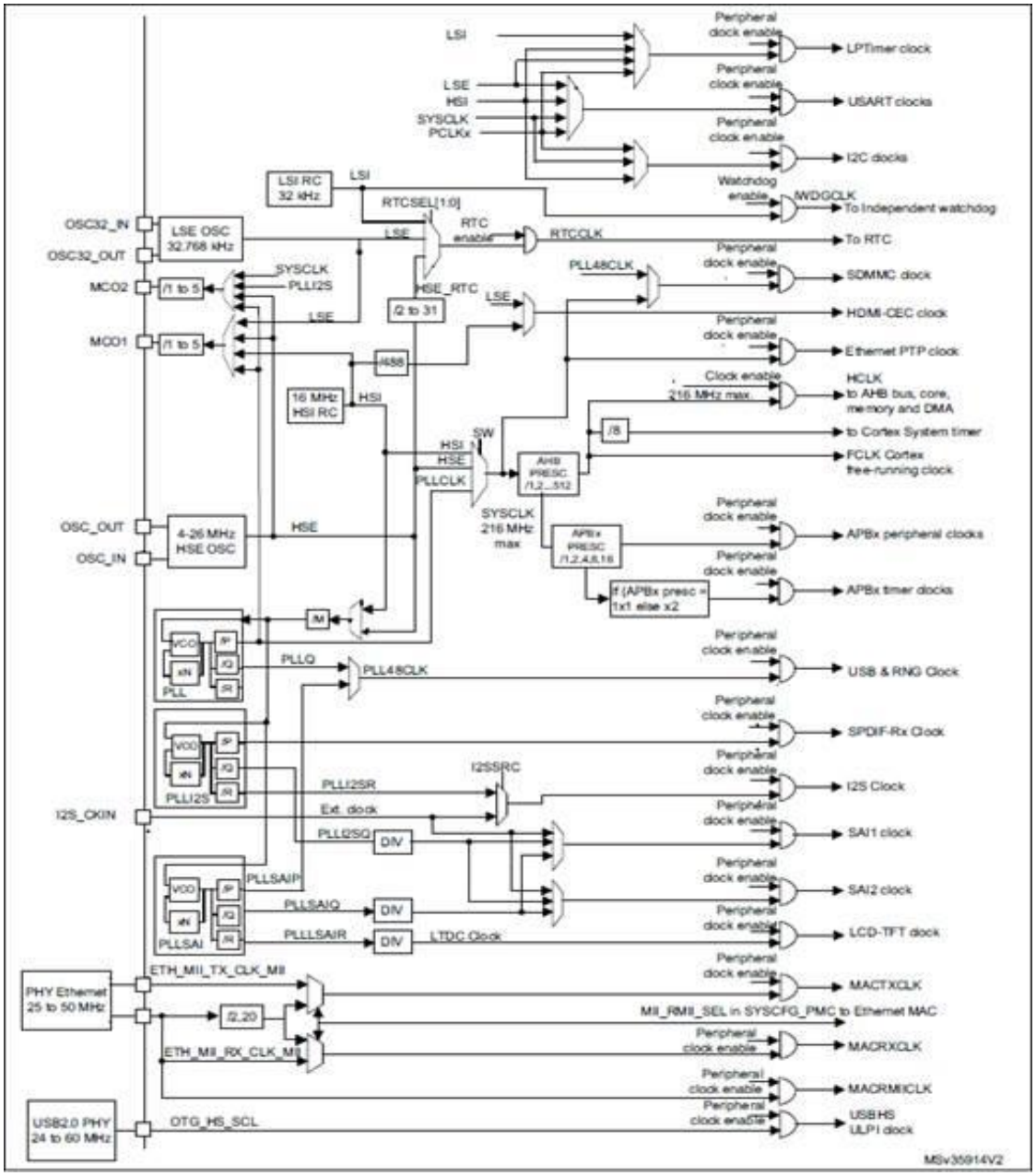

Figure 0.1 STM32 Clock Tree

2. GPIO:

Here are some of the key features of the GPIO:

I. Pin count and configuration: The STM32F4xx has up to 168 GPIO pins, which can be configured as either input or output pins. The pins can also be configured with alternate functions, such as SPI, I2C, UART, and PWM.

II. Voltage levels: The GPIO pins can operate at different voltage levels, including 3.3V, 5V, and 1.8V. This provides flexibility in interfacing with different peripherals and sensors.

III. Interrupts: The GPIO pins on the STM32F4xx support interrupts, which can be triggered by various events, such as rising or falling edges, high or low levels, and pulse widths.

IV. Pull-up and pull-down resistors: The GPIO pins on the STM32F4xx support internal pull-up and pull-down resistors, which can be used to ensure stable input levels when the pins are configured as inputs.

V. Output modes: The GPIO pins can be configured with different output modes, such as push-pull, open-drain, and open-drain with pull-up/pull-down. This provides flexibility in controlling external devices and circuits.

VI. Speed: The GPIO pins on the STM32F4xx can operate at high speeds, up to 100 MHz, making them suitable for use in high-speed applications.

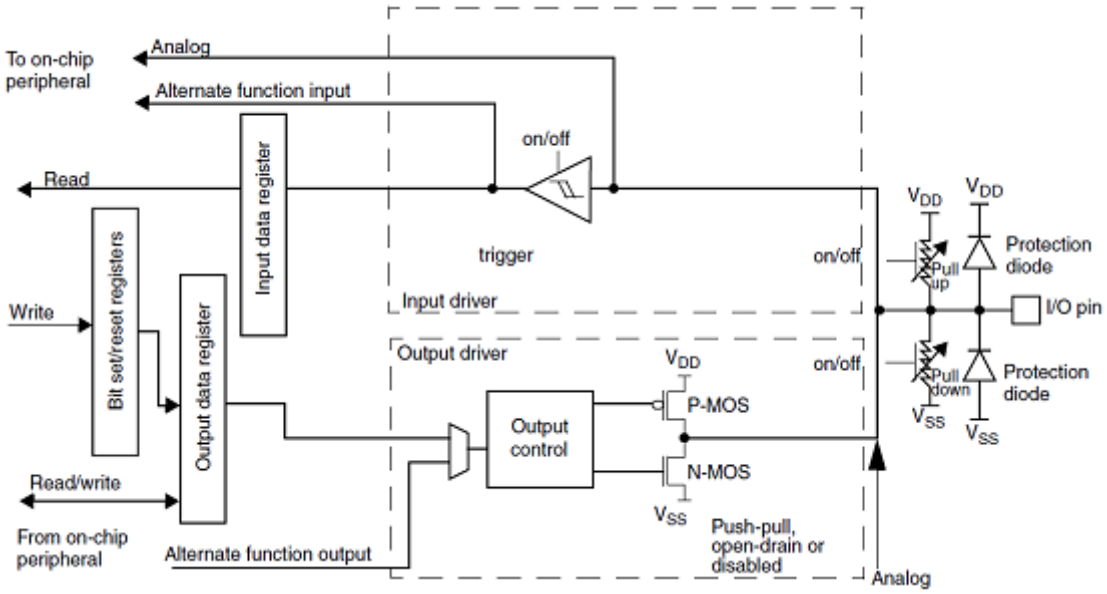

Figure 0.2 GPIO Block Diagram

To configure the GPIO pins, the following steps can be taken:

I. Enable the GPIO clock:
II. Configure the pin mode
III. Configure the pin type:
IV. Configure the pin speed:
V. Configure the pull-up/pull-down resistors
VI. Configure the alternate function

Overall, the GPIO pins on the STM32F4xx microcontroller family are versatile and powerful, offering a wide range of configuration options and features. The STM32F4xx microcontroller family provides a rich set of features and resources, including development tools and documentation, to assist in the GPIO configuration process.

3. **NVIC:**

manages the priority and handling of interrupts. It is responsible for managing and distributing interrupt requests to the appropriate interrupt service routines (ISRs) in the event of an interrupt.

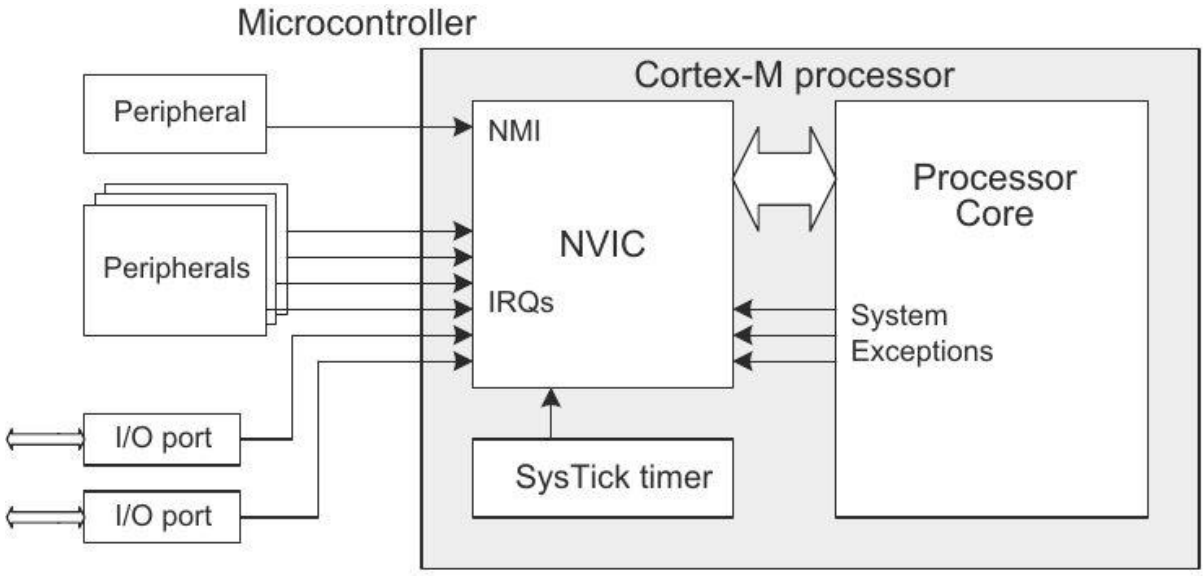

Figure 0.3 NVIC Block Diagram

Here are the steps to configure the NVIC:

a. Enable the interrupt in the peripheral you want to use by setting the appropriate bit in its interrupt enable register.
b. Set the priority level for the interrupt using the NVIC priority registers. The NVIC can handle up to 82 interrupts, each with its own priority level. Lower priority levels are assigned to higher priority interrupts, with priority level 0 being the highest priority.
c. Enable the interrupt in the NVIC by setting the appropriate bit in the NVIC enable register.
d. Write the ISR (Interrupt Service Routine) for the interrupt. This is the code that will be executed when the interrupt is triggered.
e. In the main program, enable interrupts by setting the global interrupt enable bit in the control register.

4. **DMA:**

Direct Memory Access controller that allows for high-speed data transfer between peripherals and memory without CPU intervention. The DMA controller is capable of transferring data in both directions, from memory to peripheral and from peripheral to memory. We have two DMA peripherals that have these features:

- DMA has priority over the processor in memory access
- Work as slave when programmed and as a master when start transfer data.
- Every DMA has eight streams

- Every stream has eight channels/requests.
- Every stream has a four-word FIFO.
- Every DMA has arbiter that used when there is more than stream want to work at the same time and it supports four level of SW priority. And when two streams have the same SW priority it returns to HW priority that is assigned in an inverse proportion with the stream number i.e., stream zero has priority over stream one.
- Can do up to 65533 transactions.
- Support peripheral control of transactions.
- In FIFO mode the source and destination widths are independent.

Figure 32. DMA block diagram

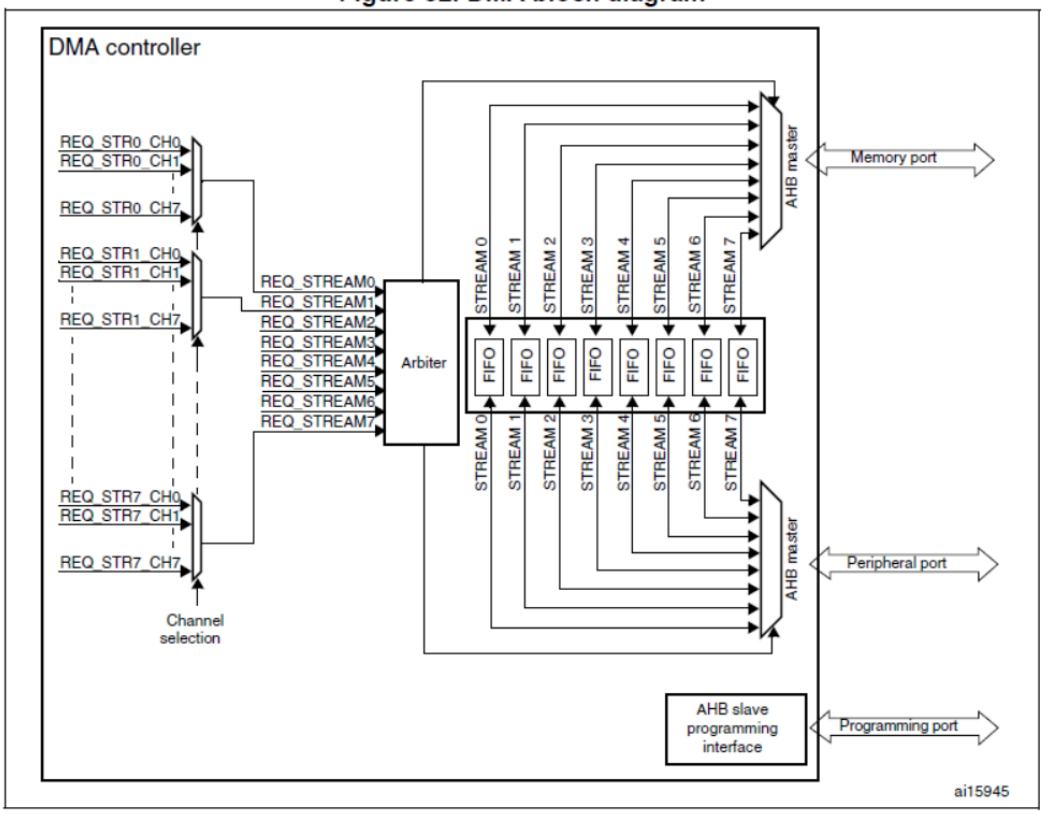

Figure 0.4 DMA Block Diagram

DMA transfer consists of three operations:

1- Loading from the source (mem or peripheral).
2- Load the data to destination.
3- Post increment of number of transactions.

Here are the steps to configure the DMA:

a. Disable the stream by resetting EN bit
b. Polling in the EN bit to make sure it has been 0
c. Configure the total number of transactions to be transferred in case of DMA controller

d. Select the DMA channel
e. Configure the stream priority, FIFO usage, Data trans direction, and mode
f. Activate the DMA

Here are the best scenarios for the transactions based on the data direction

Data Direction	Scenario
Memory to Peripheral	The best case is direct mode to make sure no data is lost
Peripheral to Memory	It's advisable to use FIFO with burst
Memory to Memory	You must work in FIFO mode

Table 0.1 DMA Scenarios

5. ADC

The STM32F4xx microcontroller family from STMicroelectronics features a high-performance analog-to-digital converter (ADC) peripheral.

Here are some of the key features of the ADC on the STM32F4xx:

a. Resolution: The ADC on the STM32F4xx can have a resolution of up to 12 bits. This provides high accuracy and precision for analog-to-digital conversions.
b. Sampling rate: The ADC can operate at a maximum sampling rate of 5 MSPS (mega-samples per second) in fast interleaved mode and up to 2.4 MSPS in regular mode. This enables fast and efficient conversion of analog signals.
c. Input voltage range: The ADC can operate with a wide input voltage range of up to 3.6V. This makes it suitable for use with a variety of sensors and input signals.
d. Sampling time: The ADC allows for adjustable sampling times, which can be set to different values depending on the application requirements.
e. Conversion modes: The ADC supports various conversion modes, including single, continuous, and scan modes. These modes can be selected based on the specific application requirements.
f. Trigger sources: The ADC allows for various trigger sources, including software, external events, and timers. This provides flexibility in controlling the conversion process.
g. DMA support: The ADC supports direct memory access (DMA), which enables efficient and fast transfer of conversion results to memory.
h. Temperature sensor: The STM32F4xx microcontroller family includes a temperature sensor that can be used with the ADC.

Overall, the ADC on the STM32F4xx microcontroller family is a powerful peripheral that provides high accuracy and precision for analog-to-digital conversions. It includes several features and options for configuring and

controlling the conversion process and is suitable for a wide range of applications, such as data acquisition, instrumentation, and control systems.

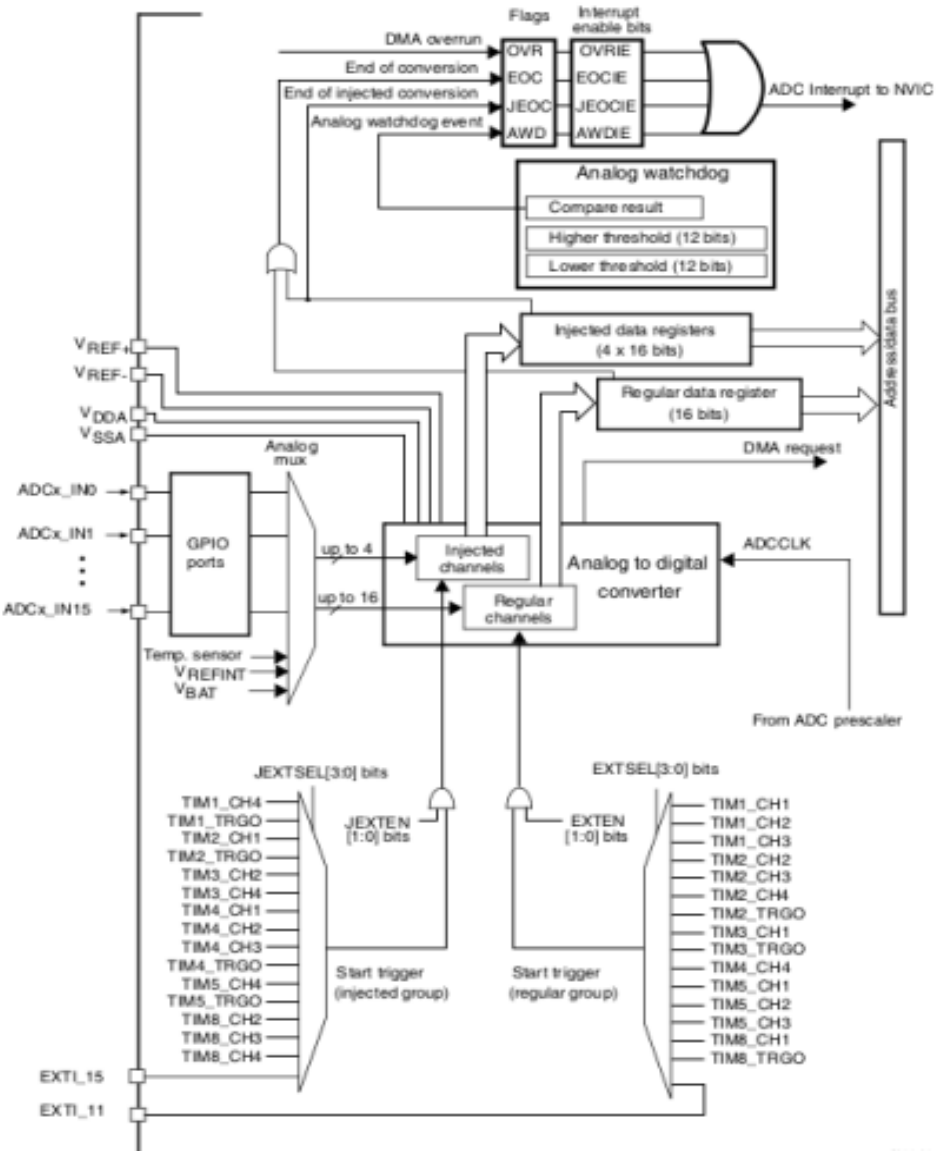

Figure 0.5 ADC Block Diagram

Here are the steps to configure the ADC:

I. Enable the ADC clock
II. Configure the ADC pins
III. Configure the ADC registers
IV. Configure the sampling time
V. Configure the conversion resolution
VI. Configure the conversion sequence
VII. Start the conversion
VIII. Read the conversion result

6. DCMI

The DCMI is used to interface with digital image sensors and is designed to handle image data streams efficiently.

The STM32F429ZI microcontroller features a high-speed DCMI interface that supports multiple image formats, including YCbCr 4:2:2, RGB565, and JPEG. The DCMI interface can operate in both master and slave modes, depending on the application requirements.

Some of the key features of the DCMI interface on the STM32F429ZI include:

a. Support for multiple image formats: The DCMI interface supports several image formats, including YCbCr 4:2:2, RGB565, and JPEG.
b. High-speed data transfer: The DCMI interface supports high-speed data transfer, with a maximum data rate of up to 54 Mbps.
c. Flexible configuration: The DCMI interface can be configured to operate in both master and slave modes, with various options for controlling the image data stream.
d. DMA support: The DCMI interface on the STM32F429ZI supports direct memory access (DMA), which allows for efficient and fast transfer of image data.
e. Interrupts and events: The DCMI interface support interrupts and events, which can be used to trigger actions based on specific events or data transfers.

Overall, the DCMI interface is a powerful peripheral that can be used to interface with digital image sensors efficiently. The STM32F429ZI microcontroller provides a high-speed DCMI interface with several features and options for configuring and controlling the image data stream, making it suitable for a wide range of applications.

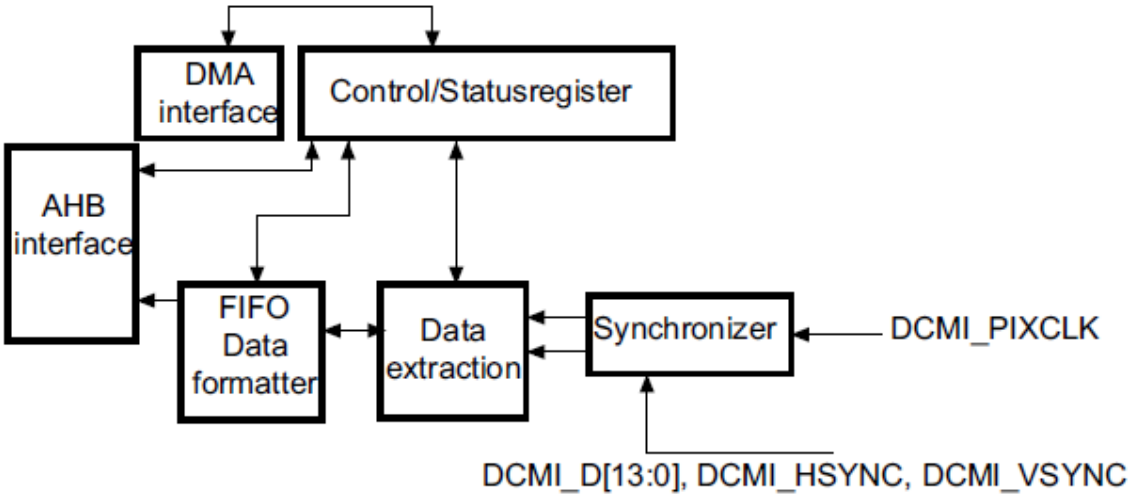

Figure 0.6 DCMI Block Diagram

Configuring the DCMI peripheral on an STM32 microcontroller like the STM32F429ZI involves several steps. Here is a general overview of the process:

1- Enable the DCMI peripheral clock
2- Configure the GPIO pins
3- Configure the DCMI peripheral
4- Configure DMA
5- Enable interrupts

7. EXTI

The STM32F4xx microcontroller family from STMicroelectronics features an external interrupt (EXTI) peripheral that allows the microcontroller to respond to external events.

Here are some of the key features of the EXTI on the STM32F4xx:

I. Interrupt sources: The EXTI on the STM32F4xx can be triggered by various sources, including GPIO pins, timers, and external signals.
II. Trigger modes: The EXTI can be configured to trigger on rising edges, falling edges, or both rising and falling edges.
III. Interrupt priority: The EXTI has a configurable interrupt priority, allowing for efficient handling of multiple interrupts.
IV. Wake-up capability: The EXTI can be used to wake up the microcontroller from low-power modes, allowing for efficient power management.

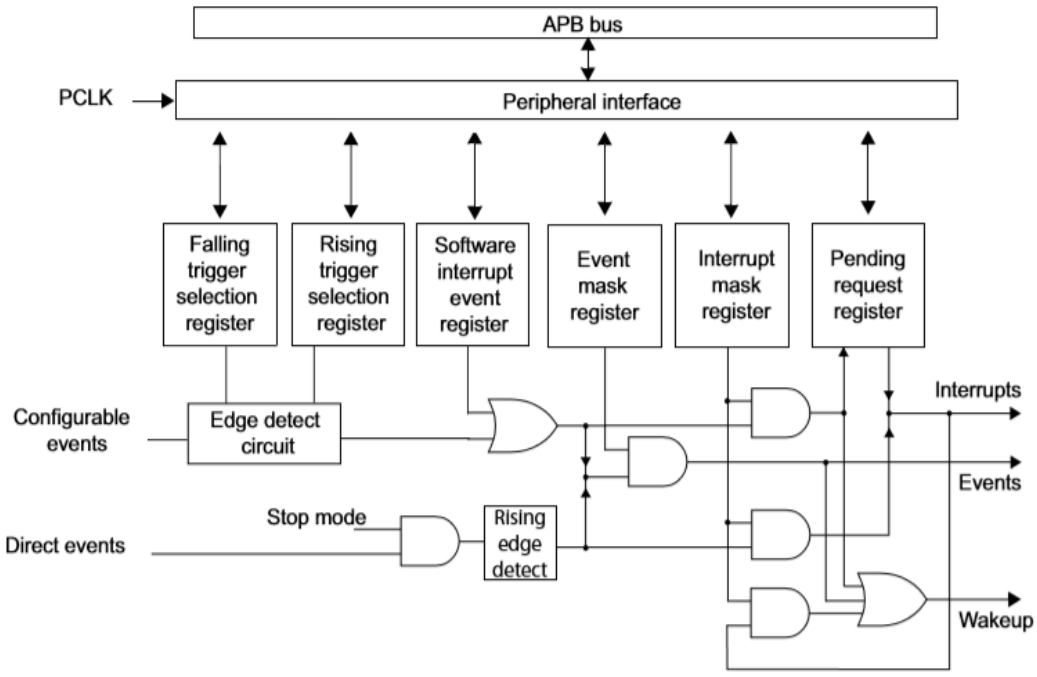

Figure 0.7 EXTI Block Diagram

To configure the EXTI, the following steps can be taken:

I. Enable the EXTI clock
II. Configure the interrupt source
III. Configure the trigger mode
IV. Enable the interrupt
V. Set the interrupt priority
VI. Handle the interrupt

8. USART

peripheral that provides serial communication capabilities.

Here are some of the key features of the USART:

I. Communication modes: The USART on the STM32F4xx supports several communication modes, including asynchronous, synchronous, and smart card modes.
II. Data format: The USART supports various data formats, including 7 or 8-bit data, parity, and stop bits.
III. Baud rate: The USART can operate at various baud rates, ranging from a few hundred bits per second up to several megabits per second, depending on the specific requirements of the application.
IV. Interrupts: The USART supports interrupts for efficient handling of communication events, such as data received or transmitted.
V. DMA support: The USART supports direct memory access (DMA), which enables efficient and fast transfer of data to and from memory.

To configure the USART, the following steps can be taken:

I. Enable the USART clock
II. Configure the GPIO pins
III. Configure the USART registers
IV. Configure the interrupt or DMA
V. Enable the USART
VI. Transmit and receive data

Overall, the USART on the STM32F4xx microcontroller family provides a powerful way to communicate with external devices and peripherals.

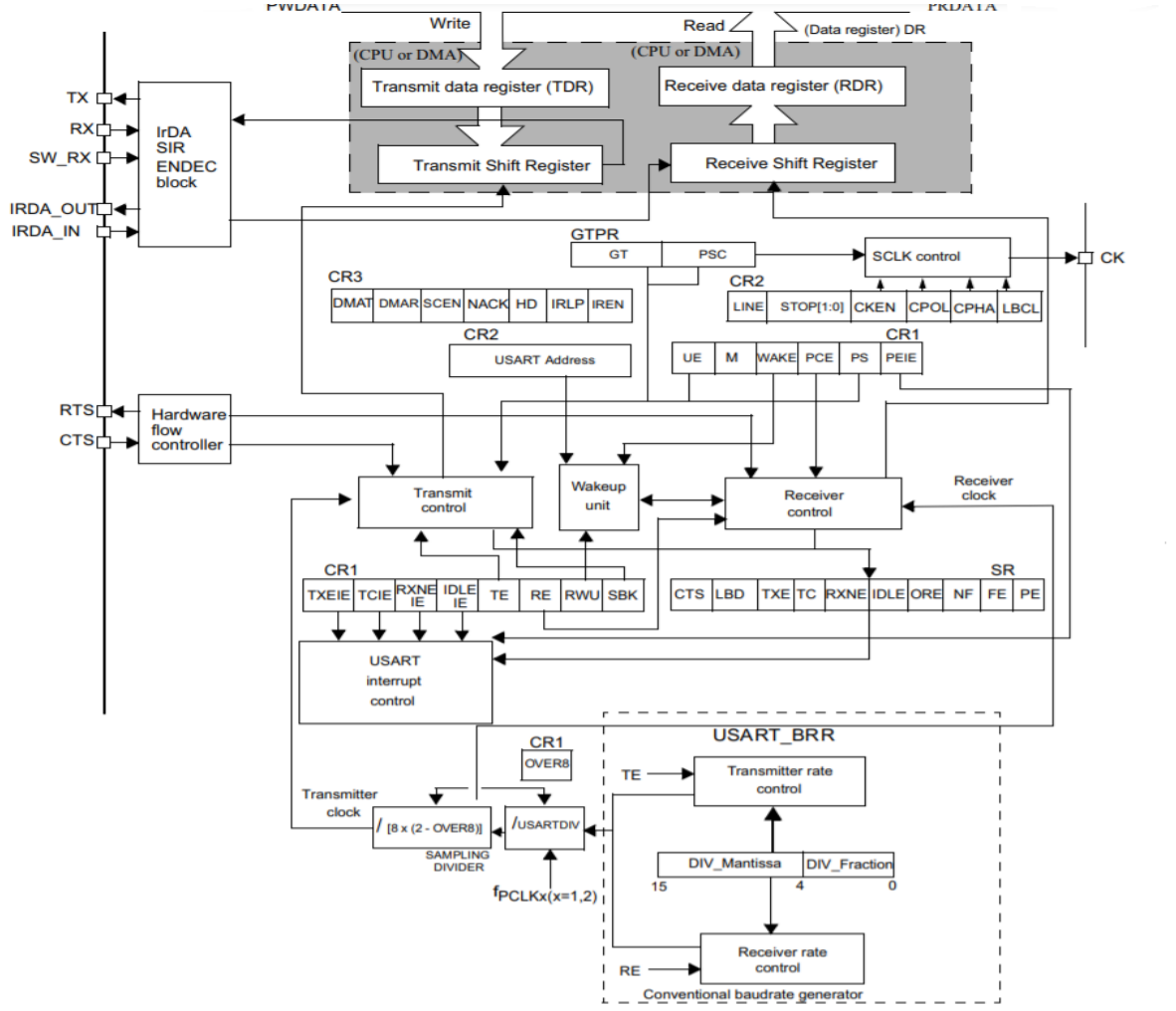

Figure 0.8 USART Block Diagram

9. I2C

I2C peripheral that provides serial communication capabilities for connecting to sensors, EEPROMs, and other devices.

Here are some of the key features of the I2C on the STM32F4xx:

I. Communication modes: The I2C on the STM32F4xx supports both master and slave modes of operation.

II. Data format: The I2C supports various data formats, including 7 or 10-bit addressing, and multiple data bytes.

III. Clock speed: The I2C can operate at various clock speeds, up to 400 kHz in standard mode and up to 1 MHz in fast mode.

IV. Interrupts: The I2C supports interrupts for efficient handling of communication events, such as data received or transmitted.

V. DMA support: The I2C supports direct memory access (DMA), which enables efficient and fast transfer of data to and from memory

To configure the I2C, the following steps can be taken:

I. Enable the I2C clock
II. Configure the GPIO pins
III. Configure the I2C registers
IV. Configure the interrupt or DMA
V. Enable the I2C
VI. Transmit and receive data

Overall, the I2C on the STM32F4xx microcontroller family provides a powerful way to communicate with external sensors and other devices.

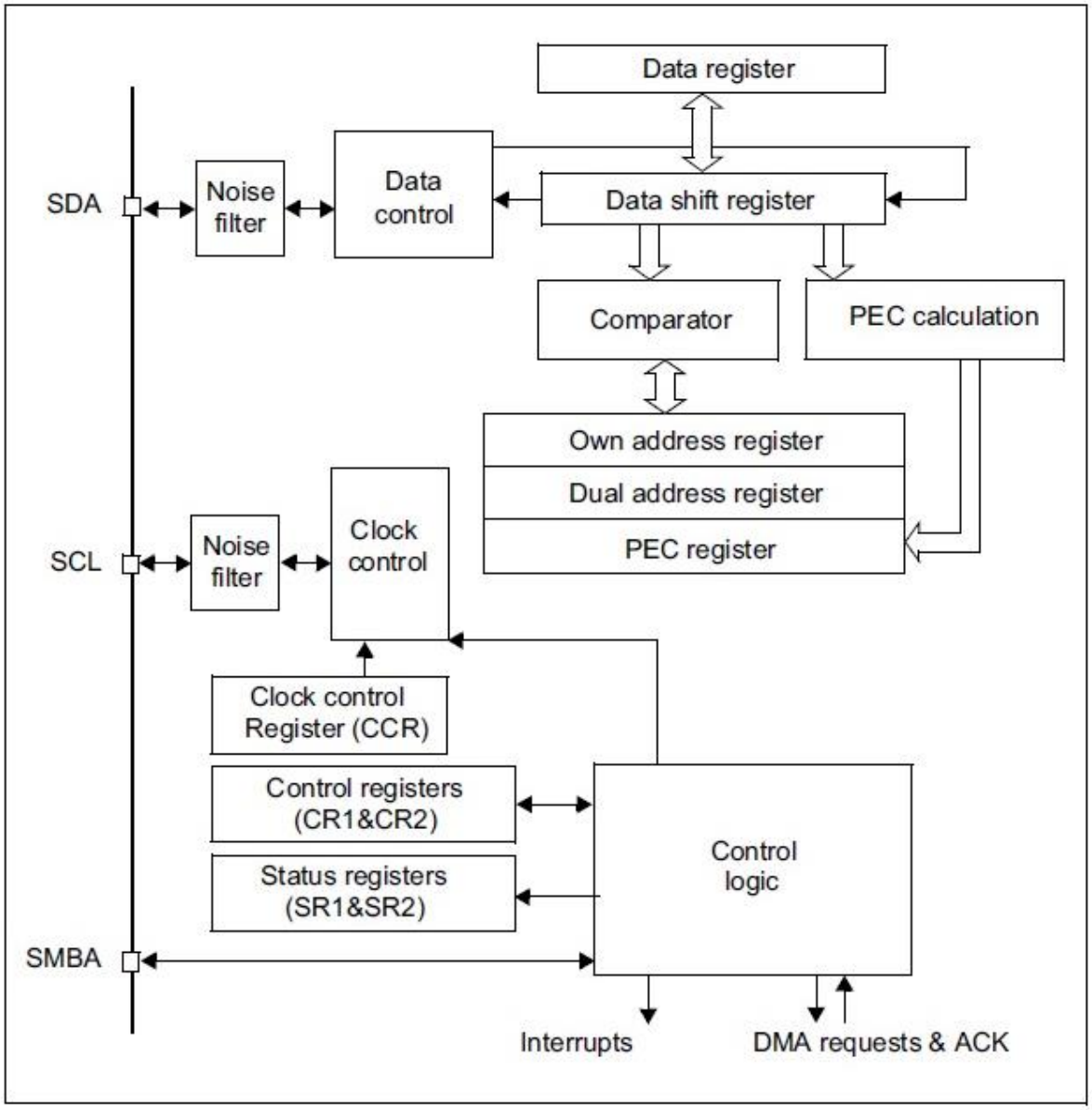

Figure 0.9 I2C Block Diagram

10. SysTick

The SysTick counter runs on the processor clock. If this clock signal is stopped for low power mode, the SysTick counter stops.

Ensure software uses aligned word accesses to access the SysTick registers.

The SysTick counter reload and current value are undefined at reset, the correct initialization sequence for the SysTick counter is:

I. Program reload value.
II. Clear current value.
III. Program Control and Status register.
IV. Set the interrupt priority
V. Enable the SysTick interrupt
VI. Handle the SysTick interrupt

Overall, the SysTick on the STM32F4xx microcontroller family provides a simple and flexible way to generate interrupts at a fixed rate.

11. ESP

As we mentioned ESP is our port to talk over Ethernet so, it's one of the most important on-board devices that will be configured.

The first step is to know how to talk to ESP after searching we found that ESP can communicate over USART using what named AT commands so, let's know more about AT commands.

Command	Usage
AT	Test the module connectivity
ATE0	Close the echo
ATE1	open the Echo again
AT+RST	Reset the configuration
AT+CWMODE=1	Configure The ESP as station
AT+CWJAP_CUR="Wi-Fi Name","Wi-Fi pass"	Connect to Wi-Fi
AT+CWQAP	Disconnect from WI-FI
AT+CIPSTART="Mode","IP or DNS",PortNum	Connect to server
AT+CIPSEND= # of data to be sent	Send Data to server

Table 0.2 AT Commands

To configure the ESP there are some steps that must be done to make sure that ESP works in the right way:

1. Start by Reset the ESP to have the default configuration.
2. Send AT to make sure that the ESP is connected and can response then check that the response is "OK".
3. Close the echo to receive the responses only as the default configuration of ESP is to echo every char received by it

4. Set the mode as station to make it connect to any WI-FI needed then validate its response is “OK”
5. Connect to WI-FI then validate Connection using ESP response “WIFI Connected\r\n WIFI GOT IP\r\nOK”
6. Connect to the Server you want to communicate with and validate Connection using ESP response “CONNECT\r\nOK”
7. The Last step is to exchange data between MCU and server

12. OV7670

The MCU controller interfaces with the OV7670 camera module through two different interfaces: I2c and DCMI.

The MCU receives the image from the camera module through the DCMI interface where it takes load off the processor. Where the DCMI peripheral is the one that waits for the VSYNC and HREF signal from the camera module. And when it sends a valid data the DCMI receives this pixel data and sends a request to DMA to transfer the pixel data from its data register to a place in SRAM where we can later send it through the Wi-Fi module to the server.

The MCU programs the registers inside of the OV7670 module with the configured values through the I2c interface. Each register is responsible for a particular functionality. The following is a breakdown of all the settings we programmed inside of the module.

1. Format

The camera module supports different formats (YUV, RGB, RAW BAYER). The format we use in our system is YUV 4:2:2 where each pixel is represented using 2 bytes. YUV 4:2:2 is YUV 4:4:4 sampled down by 2 horizontally resulting in neighboring pixels sharing chroma information. To set the YUV format we do the following:

Register	Address	Value	Comments
COM7	0x12	0x00	Select the YUV format
RGB444	0x8c	0x00	Disable RGB444 format
COM1	0x04	0x00	Disable CCIR656 format
COM15	X40	0xc0	Output full range (0x00 -> 0xff)

Table 0.3 Format Registers Values

2. Image scaling and resolution

The OV7670 has an image scaling circuitry inside of it consisting of two blocks as shown in the figure below.

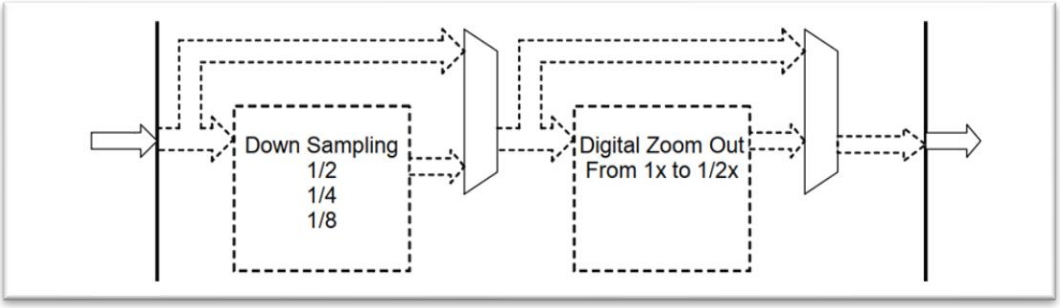

Table 0.4 Image scaling circuitry

2.1. Down sampling

As mentioned before in embedded system we deal with limited resources in our case a 256KB SRAM. So, the image needs to fit inside of the SRAM leaving enough space for other memory sections for the application to run.

Each pixel in the YUV 4:2:2 format takes 2 bytes so I can't store a full VGA 640x480 YUV image. The down sampling can be by 2, 4, or 8 vertically and horizontally. In our case, we down sample it by 2 horizontally and by 2 vertically resulting in an image of 150KB in size. To set the down sampling we do the following.

Register	Address	Value	Comments
COM3[2]	0x0c	1	Enable down sampling circuitry.
SCALING_DWCTR	0x72	0x11	Down sample horizontally by 2 and vertically by 2

Table 0.5 Down sampling

2.2. Zoom out.

The zoom out block gives us fractional scaling where we can achieve any resolution we want below VGA. But note that due to the limited buffer of this block the image coming out of the down sampling block needs to be at most CIF in resolution. So, we can't feed a full VGA 640x480 directly to the digital zoom out block.

In our case, we get the resolution we want only using the down sampling block. So, there's no need to enable digital zoom out block.

Register	Address	Value	Comments
COM3[3]	0x0c	0	Disable digital zoom our block
SCALING_XSC [6:0]	0x70	0x20	Indicates horizontal scaling ratio in zoom out circuit (0x20 -> No scaling)
SCALING_YSC [6:0]	0x71	0x20	Indicates vertical scaling ratio in zoom out circuit (0x20 -> No scaling)

Table 0.6 Zoom Out

2.3. Adjust pixel clock

Since we've altered the image resolution and sampled it by 2 vertically and horizontally, now we need to adjust the pixel clock accordingly to achieve correct timing for the new image size.

In case the new image size is not a multiple of the original size the pixel clock should be delayed achieving correct timings. Use the following formula to calculate the value needed to program inside of the SCALING_PCLK_DELAY [3:0] register bits.

Register	Address	Value	Comments
SCALING_PCLK_DIV	0x73	0x01	Set pixel clock divider to 2 to adjust timing for new image size down sampled by 2.
SCALING_PCLK_DELAY [3:0]	0xa2	0x00	No delay is required since the new image size is a multiple of the original image.

Table 0.7 Adjust pixel clock

$$\frac{originalImageHorizontalPixels}{pixelClockDivider} - newImageHorizontalPixels$$

Equation 0.1 Scaling PCLCK Delay

3. Hardware windowing

The camera module provides a feature where we can essentially crop the image. Where we can specify a region of the original image where the HREF will be asserted when the active region of the image is being output.

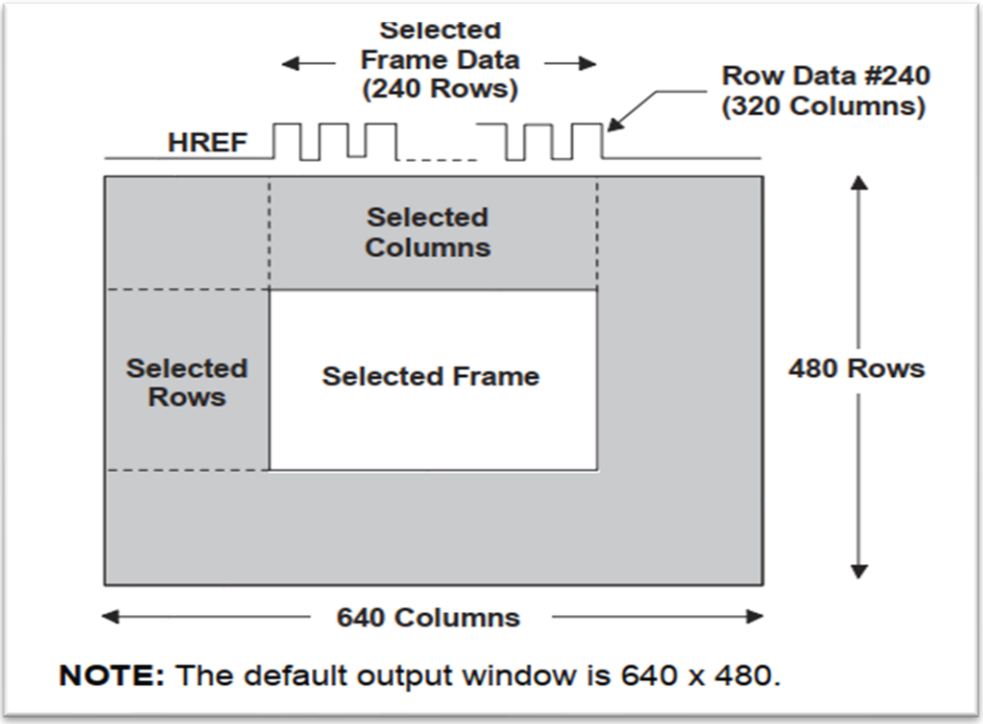

Figure 0.10 Hardware windowing

In our case, since we need the full frame and not just a part of it, we use the HREF and VSYNC signal where it syncs with the full frame.

4. Gamma curve

The gamma curve is essentially a mathematical function that maps the input intensity values (typically in the range of 0-255) to output values. It adjusts the distribution of brightness levels to compensate for the nonlinear response of the display system. The gamma curve allows for fine-tuning the display of shadows, midtones, and highlights, improving the overall appearance and contrast of the image.

The OV7670 gamma curve is composed of approximately 16 linear lines as shown in the following figure.

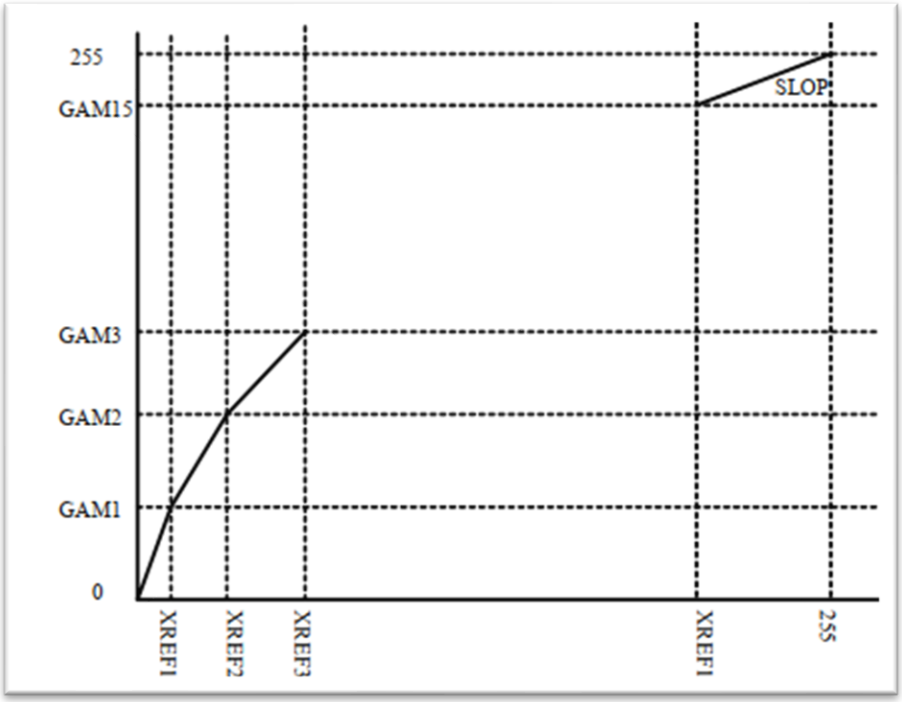

Figure 0.11 Gamma Curve

We use gamma curve 2.2. To use this curve, we do the following.

Register	Address	Value	Register	Address	Value
GAM1	0x7b	39	GAM9	0x83	144
GAM2	0x7c	53	GAM10	0x84	151
GAM3	0x7d	73	GAM11	0x85	164
GAM4	0x7e	99	GAM12	0x86	176
GAM5	0x7f	110	GAM13	0x87	197
GAM6	0x80	120	GAM14	0x88	216
GAM7	0x81	128	GAM15	0x89	232
GAM8	0x82	136	SLOP	0x7a	32

Figure 0.12 Gamma Configuration

5. **Color matrix**

If we use the YUV image format or any other formats derived from it then we have to deal with what's called color matrix. The OV7670 camera module has this 3x3 matrix inside of it where it does the following:

a. **Color correction.**
b. **RGB to YUV/YCbCr conversion.**
c. **Color saturation.**
d. **Hue control.**

To program the appropriate matrix, we need we first calculate the combined 3x3 matrix and after calculating this combined matrix take the last two rows as our final color matrix that needs to be programmed inside the module where,

$$combinedMatrix = saturationMatrix * hueMatix * coversionMatrix * correctionMatrix$$

Equation 0.2 combined Matrix

Register	Address	Value
MTX1	0x4f	0x80
MTX2	0x50	0x80
MTX3	0x51	0x00
MTX4	0x52	0x22
MTX5	0x53	0x5e
MTX6	0x54	0x80
MTXS	0x58	0x9e

Table 0.8 Color Matrix Configuration

6. **Enable additional features**

As mentioned before, the OV7670 camera module supports features like: AEC, AGC, AWB, ABF, ABLC and Saturation level auto adjust. To enable some of them we do the following. (AGC, AWB and AEC are enabled by default)

Register	Address	Value	Comments
COM13	0x7a	0xc0	Enable gamma and automatic saturation.
COM16	0x41	0x38	Enable edge-enhancement, de-noise.

Table 0.9 Additional Features

13. Water pump

The MCU interface with the water pumps through a single interface: GPIO peripheral.

Since the MCU can't drive the Water pump directly, it sends its control signals to electromechanical switches called "relays". Where it switches the 5V

Water pump without damaging the MCU pins. The relays we use can handle up to 10A of current so there will be no problem feeding enough power to the pump without any damage.

The MCU sends the control signal to turn on and off the pump depending on the readings of the water tank state. If the water tank is empty, the control signal will be high and the water pump will turn on. If the water tank is full, the control signal will be low and the water pump will turn off.

14. Valve

a. Solenoid Valve ½"

The MCU interface with the solenoid valve ½" through a single interface: GPIO peripheral.

Since the MCU can't drive the solenoid valve ½" directly, it sends its control signals to electromechanical switches called "relays". Where it switches the 12V solenoid valve ½" without damaging the MCU pins. The relays we use can handle up to 10A of current so there will be no problem feeding enough power to the solenoid valve ½" without any damage.

The solenoid valve ½" is normally closed.

The MCU sends the control signal to open and close the solenoid valve ½" depending on the readings of the water tank state. If the water tank is empty, the control signal will be high and solenoid valve ½" will open and water will flow to the tank after the water pump is turned on. If the water tank is full, the control signal will be low and solenoid valve ½" will close and water will stop.

b. Solenoid Valve ¼"

The MCU interface with the solenoid valve ¼" through a single interface: GPIO peripheral.

Since the MCU can't drive the solenoid valve ¼" directly, it sends its control signals to electromechanical switches called "relays". Where it switches the 12V solenoid valve ¼" without damaging the MCU pins. The relays we use can handle up to 10A of current so there will be no problem feeding enough power to the solenoid valve ¼" without any damage.

The solenoid valve ¼" is normally closed.

The MCU sends the control signal to open and close the solenoid valve ¼" depending on the readings of the FC-28 module (Soil Moisture Sensor Module). If the soil moisture sensor reads that the soil is dry, the control signal will be high and solenoid valve ¼" will open and

water will flow from the water tank through the hoses to irrigate the specific plant. If the soil moisture sensor reads that the soil is humid, the control signal will be low and solenoid valve ¼" will close and no water will stop.

15. Water Level Switch

The MCU interface with two water level switches through a single interface: GPIO peripheral.

The software module is responsible for checking the current state of the two water level sensors to determine whether the water tank is empty or not. In the API, where it reads the current state, it supports software debouncing to solve the bouncing issue.

The software module is configured in such a way that it reads the current state of the switches correctly regardless of whether it is pulled-up or down. It does so through a configuration type definition called "Switch_Pull_t".

The software module provides us interrupt capabilities where the interrupt is fired when the switch changes state. It also provides us with a way to set the callback function that will be called when this event happens.

16. Water Tank

We use this software module to implement the water tank module in the layer above. Where in our case, the water tank has inside of it two water level switches. One of them is mounted at a height of 80% of the total water tank height. And the other one is mounted at a height of 20% of it. These two water level sensors result in 4 different possibilities of mounting them. The one we choose is that the high-level switch is normally open, and the low-level switch is normally closed. The tank software module is implemented in such a way that it handles the 4 different possibilities.

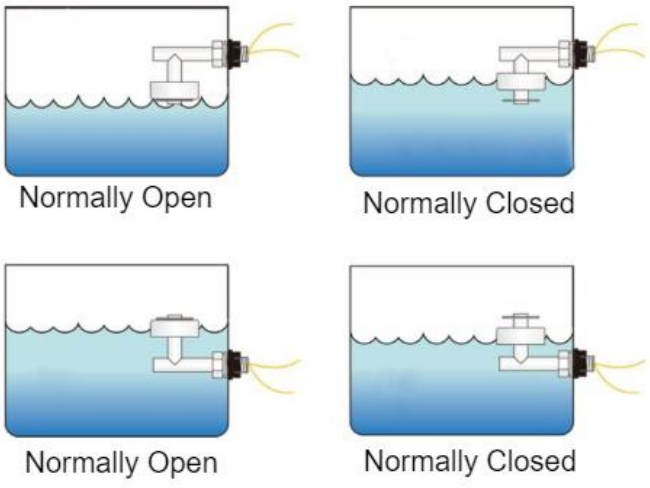

Figure 0.13 Tank States

The tank software module gets the current state of the tank regardless of the chosen method to mount the two water level switches. The tank has only three states. It can only be in one state at a time. The states are if it's empty, standby, or full.

17. FC-28

The MCU interface with the FC-28 (Moisture sensor) through two different interfaces: GPIO peripheral and ADC peripheral.

The FC-28 software module provides us with two methods to read the state of the soil moisture.

The first one is by using the ADC peripheral where the MCU is connected to the FC-28 via an analog pin (A0). This method provides us with an accurate read of the current soil moisture percentage after of course converting the digital reading of the ADC to the correct value.

The second method is using the GPIO peripheral where the MCU is connected to the FC-28 via a digital pin (D0). The FC-28 IC has a comparator IC LM393 that compares the current moisture level of the soil with an already preset threshold. The threshold is adjusted via a mechanical potentiometer on the FC-28 IC. This method provides us with the state of the soil whether it's dry or humid with reference to the already set threshold.

The software module provides us interrupt capabilities where the interrupt is fired when the reading of the digital pin of IC module changes state. According to the module datasheet, when the module digital output is low that means that the value of the soil moisture exceeds the set threshold.

There are two possible events that could happen. The first one is a rising edge on the digital pin of the module where this indicates that the soil is dry with reference to the threshold. The second event is a falling edge on the digital pin of the module where this indicates that the soil is wet with reference to the threshold.

18. LDR

The MCU interface with the LDR (Light sensor) through two different interfaces: GPIO peripheral and ADC peripheral.

The LDR software module provides us with two methods to read the state of the ambient light.

The first one is by using the ADC peripheral where the MCU is connected to the LDR via an analog pin (A0). This method provides us with an accurate read of the current ambient light percentage after of course converting the digital reading of the ADC to the correct value.

The second method is using the GPIO peripheral where the MCU is connected to the LDR via a digital pin (D0). The LDR IC has a comparator IC LM393 that compares the current moisture level of the soil with an already preset threshold. The threshold is adjusted via a mechanical potentiometer on the LDR IC. This method provides us with the state of the ambient light whether it's dark or bright with reference to the already set threshold.

The software module provides us interrupt capabilities where the interrupt is fired when the reading of the digital pin of IC module changes state. According to the module datasheet, when the module digital output is low that means that the value of the ambient light exceeds the set threshold.

19. LED grid

The MCU interface with the LDR (Light sensor) through a single interface: GPIO peripheral.

The MCU sends the control signal to turn on and off the LED grid depending on the readings of the LDR sensor. If the LDR senses the surroundings are dark, the control signal will be high, and the LED grid will turn on. If the LDR senses the surroundings are bright, the control signal will be low, and the LED grid will turn off.

Since the MCU can't drive the LED grid directly, it sends its control signals to electromechanical switches called "relays". Where it switches the 12V LED grid without damaging the MCU pins. The relays we use can handle up to 10A of current so there will be no problem feeding enough power to the LED grid without any damage.

20. AHT21B

The MCU controller interfaces with the AHT21B (Temperature and humidity module) through a single interface: I2c.

To interface with the module, we do the following steps:

1. After power-on, wait for 100-500ms. Before reading the temperature and humidity value, get a byte of status word by sending 0x71. If the status word and 0x18 are not equal to 0x18, initialize the 0x1B, 0x1C, 0x1E registers.

Bits	Significance	Description
Bit[7]	Busy indicator	1: Equipment is busy, in measurement mode. 0: Equipment is idle, in hibernation state.
Bit[6:4]	Retain	Retain
Bit[3]	CAL enable	1: Calibrated. 0: Uncalibrated.
Bit[2:0]	Retain	Retain

Table 0.10 AHT21B Indicators

2. Initialize the register by sending the slave address followed by the address of each register followed by two bytes of zeros.
3. Delay for 5ms then Read 3 data bytes from the module.
4. Delay for 10ms then write on the (register address | 0xB0) the second and third received bytes from previous step.
5. Wait 10ms to send the 0xAC command (trigger measurement). This command parameter has two bytes, the first byte is 0x33, and the second byte is 0x00.
6. Wait more than 80ms for the measurement to be completed, if the read status word Bit [7] is 0, it means the measurement is completed, and then six bytes can be read continuously; otherwise, continue to wait for configure timeout amount.
7. After receiving six bytes, the next byte is the CRC check data. The user can read it out as needed. If the receiving end needs CRC check, an ACK will be sent after the sixth byte is received. Reply, otherwise, send NACK to end, the initial value of CRC is 0XFF, and the CRC8 check polynomial is:

$$CRC\ [7:0] = 1 + X^4 + X^5 + X^8$$

Equation 0.3 CRC Generator

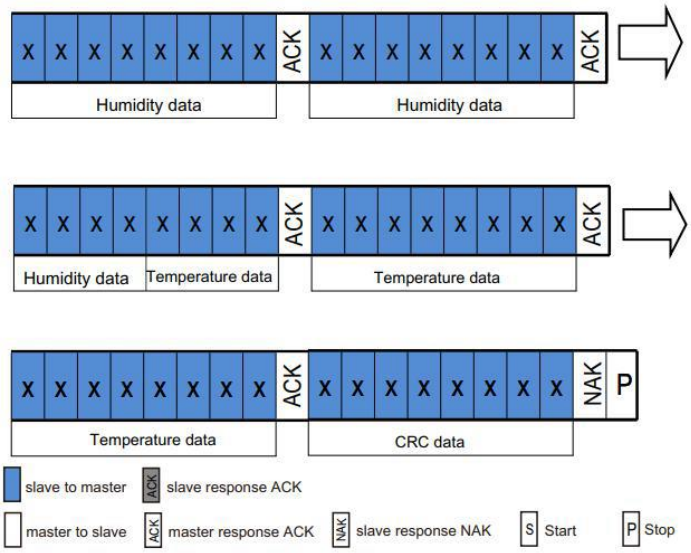

Figure 0.14 CRC

8. Calculate the temperature and humidity value.

Note: The calibration status check in the first step only needs to be checked when the power is turned on. No operation is required during the acquisition process.

a. Relative Humidity Conversion

The relative humidity RH can be calculated according to the relative humidity signal S_{RH} output by SDA through the following formula (the result is expressed in %RH)

$$RH[\%] = \left(\frac{S_{RH}}{2^{20}}\right) * 100\ \%$$

Equation 0.4 Relative Humidity Conversion

b. Temperature Conversion

The temperature T can be calculated by substituting the temperature output signal S_T into the following formula (The result is expressed in temperature ℃):

$$T[°C] = \left(\frac{S_T}{2^{20}}\right) * 200 - 50$$

Equation 0.5 Temperature Conversion

The software module implements the CRC8/MAXIM algorithm with the previously mentioned polynomial to check the received CRC value from the sensor and make sure that there are no errors in the bytes received. It gives you a configuration parameter to enable or disable CRC checking.

Conclusion

The implementation process can involve a variety of tasks, including installation and configuration of hardware and software, data migration, system testing, user acceptance testing, and training and documentation. It is important to carefully plan and execute the implementation phase to minimize the risk of disruption to existing business processes and to ensure that users can effectively use the system to achieve their goals.

Chapter 4: Make Bare Metal Talking to The World Using MQTT

Intro

In This chapter we will take a dive in IoT world to learn more about how to talk to servers and websites, what is MQTT, why using it, and how to configure it in our MCU. First, we will talk about IoT and what is it, why IoT and what we need from it.

IoT

The Internet of Things (IoT) is a network of physical devices, vehicles, buildings, and other objects that are embedded with sensors, software, and connectivity, allowing them to collect and exchange data with other devices and systems over the internet. The IoT is transforming the way we live and work, enabling new levels of automation, efficiency, and convenience across a wide range of industries and applications. There are several key components of the IoT ecosystem, including:

1. **Devices:**

   IoT devices can be anything from simple sensors to complex machines, and they are usually designed to collect data, monitor conditions, or perform specific tasks. Examples of IoT devices include smart thermostats, fitness trackers, industrial sensors, and autonomous vehicles.

2. **Connectivity:**

   IoT devices rely on various forms of connectivity to transmit data to other devices and systems. This can include wireless networks such as Wi-Fi, cellular networks, and low-power wide-area networks (LPWANs).

3. **Cloud Platforms:**

   IoT data is often processed and analyzed in cloud-based platforms that provide scalable and secure storage and computing resources. These platforms can be used to store, analyze, and visualize IoT data, as well as to manage and control IoT devices remotely.

4. **Applications:**

   IoT applications are software programs that are designed to interact with IoT devices and data. These applications can be used to control devices, monitor performance, automate processes, and provide insights and alerts based on IoT data.

Some of the key benefits of the IoT include:

1. **Increased Efficiency:**

   IoT devices and applications can automate and optimize many processes, reducing waste, improving productivity, and lowering costs.

2. **Improved Safety and Security:**

   IoT devices can monitor and alert users to potential safety hazards or security breaches, enabling them to take action quickly and prevent accidents or losses.

3. **Better Decision Making:**

   IoT data can provide valuable insights into customer behavior, product performance, and operational efficiency, enabling businesses to make more informed decisions.

4. **Enhanced User Experience:**

   IoT devices can provide personalized and convenient experiences for users, such as smart home devices that adjust lighting and temperature based on user preferences.

However, there are also several challenges associated with the IoT, including security and privacy concerns, interoperability issues, and the need for reliable connectivity and data management. As the IoT continues to evolve and mature, these challenges will need to be addressed to ensure that the IoT can deliver on its promise of a more connected and efficient world. We have lots of IoT protocols that can be used to connect our MCU to the internet. After searching we found two protocols that can be used in our case MQTT and COAP that are Application Layer protocols in internet stack and we will compare between them to choose one of Them.

Protocol Name	Network Layer	Transport Layer Protocol
COAP	APP	UDP
MQTT	APP	TCP

Table 0.1 Compare Between MQTT and COAP

So, the main factor that we will choose the protocol based on it is the Transport Layer Protocol that used to send the Data. So, we need to know more about TCP and UDP.

1. **TCP:**

   Transmission Control Protocol is a widely used transport layer protocol in computer networking, providing reliable, ordered, and error-checked delivery of data between applications running on different hosts. TCP is a connection-oriented protocol, which means that it establishes a virtual connection between two endpoints before transmitting data. When two hosts want to communicate using TCP, they first establish a connection using a three-way handshake.

2. **UDP:**

   User Datagram Protocol is a transport layer protocol in computer networking that provides lightweight, low-overhead communication between applications running on different hosts. Unlike TCP, UDP is a connectionless protocol, meaning that

it does not establish a virtual connection between two endpoints before transmitting data.

As TCP is a reliable and efficient protocol that is widely used in a variety of applications and industries. Its reliability and error-checking mechanisms make it suitable for applications that require accurate data transmission, while its flow control and congestion control mechanisms make it suitable for applications that require efficient use of network resources. We will choose MQTT to Send Data to server. So let's know more about MQTT and its configurations.

MQTT

Message Queuing Telemetry Transport is a lightweight and efficient messaging protocol that is widely used in the Internet of Things (IoT) and other low-power networked devices. So, we can say that MQTT connecting anything anywhere anytime. To make a bare metal system talk to the world using MQTT, you will need to implement the following steps:

1. **Choose an MQTT broker:**

   An MQTT broker is the central messaging server that is responsible for receiving and distributing messages between the bare metal system and other devices on the network. There are many public and private MQTT brokers available, such as Mosquitto, AWS IoT Core, Thingspeak, and Azure IoT Hub.

2. **Choose an MQTT client library:**

   An MQTT client library provides the necessary functions and APIs to send and receive MQTT messages over the network. There are many MQTT client libraries available for different programming languages and platforms, such as Paho MQTT for C/C++ and Eclipse for Python.

3. **Implement the MQTT client code:**

   Once you have chosen an MQTT client library, you can implement the MQTT client code in your bare metal system. This code will typically include the following steps:

   a. Initialize the MQTT client library and connect to the MQTT broker
   b. Subscribe to specific MQTT topics to receive messages from other devices
   c. Publish MQTT messages to specific topics to send data to other devices
   d. Handle incoming MQTT messages and perform the necessary actions based on the message content.

4. **Integrate with the bare metal system:**

   To make the MQTT client code work with the bare metal system, you will need to integrate it with the other system components, such as the sensors, actuators, and other peripherals. This integration will typically involve reading data from the sensors, processing the data, and sending it to other devices using MQTT messages.

5. **Test and deploy the system:**

Once you have implemented and integrated the MQTT client code with the bare metal system, you can test the system by sending and receiving MQTT messages and verifying that the system is working as expected. You can then deploy the system in a real-world environment and monitor its performance and reliability over time.

Overall, making a bare metal system talk to the world using MQTT requires careful planning, implementation, and testing to ensure that the system is reliable and efficient. However, with the right tools and techniques, it is possible to build powerful and scalable IoT systems that can communicate seamlessly with other devices on the network.

How MQTT Works

We will Talk about the basic concepts needed to start working with MQTT and know how it works. We will start with the definitions of the most important concepts needed.

1. **Clients and Brokers:**

MQTT uses a client-server architecture, where clients are the devices or applications that send and receive messages, and brokers are the servers that receive and distribute messages between clients. Clients can be publishers (sending messages) or subscribers (receiving messages).

2. **Topics:**

MQTT messages are organized by topics, which are hierarchical strings that represent a specific category or type of message. Topics are used to filter and route messages between clients.

3. **QoS Levels:**

MQTT supports three levels of Quality of Service (QoS), which determine the level of reliability and delivery guarantees for messages. The three levels are:

- **QoS 0 (At most once):**

Messages are delivered at most once, without any guarantee of delivery. This level is used for messages that are not critical or require real-time delivery.

- **QoS 1 (At least once):**

Messages are guaranteed to be delivered at least once, but may be delivered multiple times due to network errors or message loss. This level

is used for messages that require reliable delivery, but can tolerate duplicates.

- **QoS 2 (Exactly once):**

  Messages are guaranteed to be delivered exactly once, with no duplicates or losses. This level is used for messages that require the highest level of reliability and accuracy.

4. **Message Flow:**

   The typical flow of MQTT messages involves the following steps:

   - A client publishes a message to a specific topic, including the topic name, message payload, and QoS level.
   - The broker receives the message and stores it in a message queue for the specified topic.
   - The broker distributes the message to all subscribed clients for the topic, based on the QoS level and the availability of the clients.
   - The subscribed clients receive the message and process it according to their specific needs.

5. **Keep Alive:**

   MQTT includes a keep alive mechanism, where clients and brokers periodically exchange ping messages to ensure that their connection is still active and responsive.

   Overall, MQTT provides a simple and efficient way for devices and applications to communicate over a network, using lightweight and flexible messaging with a variety of QoS levels and topic-based routing.

In embedded system we use QoS 0 if the information isn't important and use QoS 1 if the information is important and the subscriber must tell the broker max supported QoS. In the below figure you can see the complete PUBSUB model of MQTT.

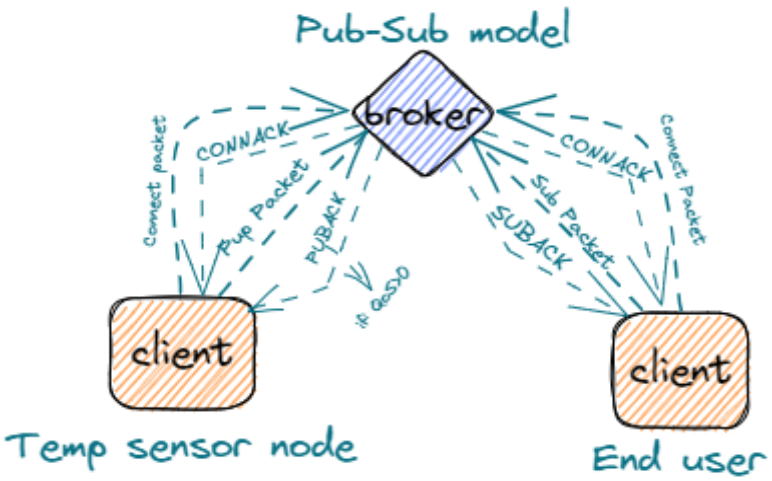

Figure 0.1 Pub-Sub Model

We know how MQTT works and the important concepts that we need. And now we will have a dive in the packets that we will need to send data but first we will talk about the frame format of the MQTT packets. So, we will talk about the MQTT packet below.

MQTT Packet:

Also called control packets, used to send commands and data to broker. So, we can say that it's the way of the communication between client and broker. So, what are these packets consist of what are the different packets send what is the usage of each one we will talk about that below. Show the figure below to know the content of Packets.

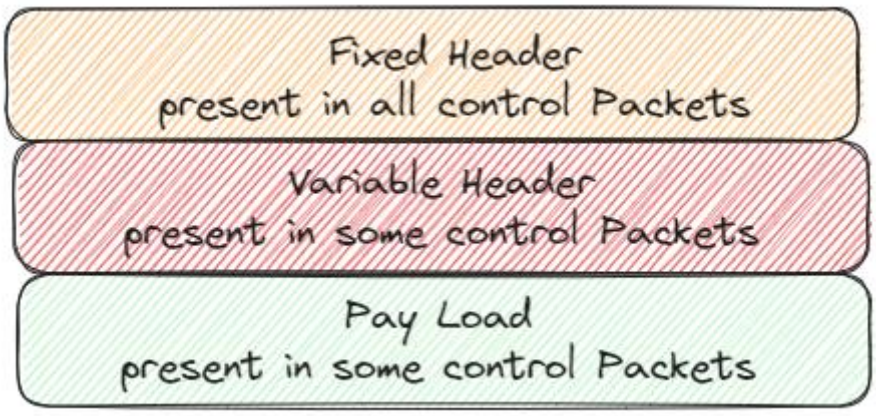

Figure 0.2 MQTT Control Packet

Content of Packet

The packet consists of from one to three sections based on the packet type and usage. The three sections of packets are:

1. **Fixed Header:**

   Consist of two to five bytes. All packets must have the fixed header as it contains:

   1. **Packet Type:**

      The packet type is the MSB of first byte that holds number express the sent packet type.

Name	Value	Direction of follow	Description
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Two Directions	Publish message
PUBACK	4	Two Directions	Publish acknowledgment
SUBSCRIBE	8	Client to Server	Client subscribes request
SUBACK	9	Server to Client	Subscribe acknowledgment

Table 0.2 Some MQTT Control Packet Types

2. **Flags:** The 4 LSB of first byte and it's specific to each control packet.
3. **Remaining Length:**

indicate the length of the variable header and payload. The Remaining Length field is a variable-length field that can range from 1 to 4 bytes, depending on the size of the packet. The format of the Remaining Length field is as follows:

- The first byte contains the least significant 7 bits of the Remaining Length value, with the most significant bit (bit 7) used as a continuation bit.
- If the continuation bit is set to 1, this indicates that there are additional bytes in the Remaining Length field. The next byte contains the next 7 bits of the value, with the continuation bit set to 1 if there are additional bytes.
- This process continues until the continuation bit is set to 0, indicating that the last byte of the Remaining Length field has been reached.

For example, if the Remaining Length field is 200, it would be encoded as follows:

- The first byte would be 0xC8 (11001000 in binary), indicating that the value is 200 and there are additional bytes in the field.
- The second byte would be 0x01 (00000001 in binary), indicating that the next 7 bits of the value are 1 and there are no additional bytes in the field.

The maximum value of the Remaining Length field is 268,435,455 (0xFFFFFF), which requires 4 bytes to encode. However, in practice, most MQTT packets have much smaller Remaining Length values, typically ranging from a few bytes to a few kilobytes. The complete fixed header is shown below.

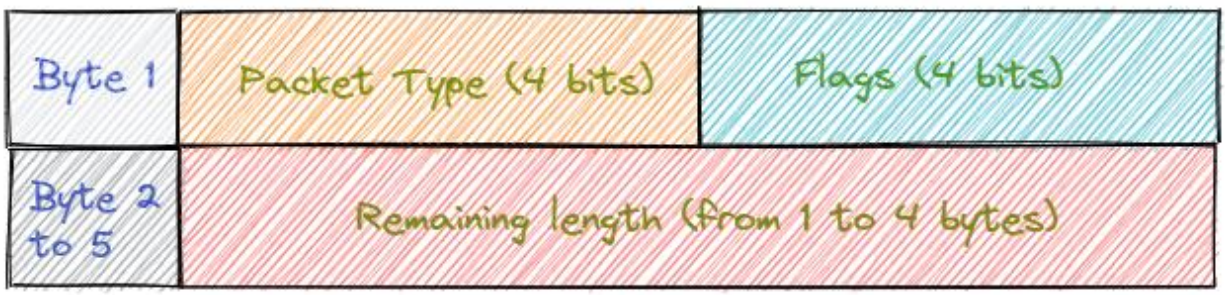

Figure 0.3 MQTT Fixed Header

2. **Variable Header:**

contains additional information about the MQTT packet beyond the fixed header. The structure of the variable header depends on the type of MQTT packet, but typically includes information such as the message ID, QoS level, topic name, and connection flags.

The most popular usage of variable header is message ID that used in publish and PUBACK packet when QoS is more than zero. The figure shown below show the ID message bytes

Figure 0.4 MQTT Packet ID

3. **Payload**

is the actual data that is being transmitted in the MQTT message. The payload is sent as part of the PUBLISH packet, which is used to send a message from a publisher client to the broker, or from the broker to a subscriber client.

We know about frame format of the packets and now we will talk about the control packets that we will use in our scope. So, let's start our journey with our control packets.

Control Packets

1. **CONNECT Packet**

The first packet sent by the client and it's used by a client to establish a connection with the broker after TCP connection is established. The CONNECT packet sent one time and if sent more than one time the connection is closed. The CONNECT packet has a fixed header, a variable header, which contains fields that specify various connection parameters, and payload. Below the frame format of connect packet in details.

1.1. Fixed Header

Bytes	Value	Description
Byte 1	0x10	The type is 1 and the flags are reserved
Bytes 2:5	X	The Remaining Length

Table 0.3 Connect Packet Fixed Header

The remaining length can be calculated from the below equation

$$Rem\ Length\ =\ 10\ +\ 2\ +\ Client\ length\ +(2+username\ length\ if\ username)\\ +(2+Password\ Length\ if\ Password)$$

Equation 0.1 Remaining Length of Connect Packet

1.2. Variable Header

Bytes	Value	Description
Bytes 1:2	0x04	The Protocol Name Size
Bytes 3:6	MQTT	The Protocol Name
Byte 7	0x04	The Protocol Level
Byte 8	X	Connect Flags
Bytes 9:10	X	KEEP ALIVE

Table 0.4 Connect Packet Variable Header

- **Connect Flags:** set of flags that control the behavior of the connection. The below table Show usage of them

Bits	Usage	Description
Bit 0	Reserved	Must be 0
Bit 1	Clean Session	Clean the last messages sent when I was offline
Bit 2	Will Flag	Set the flag when there is a will message
Bits 3:4	QoS Retain	Send the will message with QoS x
Bit 5	Will Retain	Save the will message
Bit 6	Username flag	Set when Broker has username
Bit 7	Password Flag	Set when Broker has Password

Table 0.5 Connect Flags

1.3. **Payload:** The Payload consist of UTF-8 Encoded strings

Bytes	Value	Description
Bytes 1:2	X	The Client ID Length
Bytes 3: n	X	The Client ID as String
Bytes n: n+1	X	The Username Length
Bytes n+2: k	X	The Username as String
Bytes k: k+1	X	The Password Length
Bytes k+2: L	X	The Password as String

Table 0.6 Connect Packet Payload

2. CONNACK Packet

used to acknowledge the receipt of a CONNECT packet from a client to a broker. When a client sends a CONNECT packet to a broker to establish a connection, the broker sends a CONNACK packet to the

client indicating whether the connection request has been accepted or rejected.

It consists of fixed and variable headers as follow

2.1. Fixed Header

Bytes	Value	Description
Byte 1	0x20	The type is 2 and the flags are reserved
Byte 2	0x02	The Remaining Length, the length of variable header

Table 0.7 Connect Ack Fixed Header

2.2. Variable Header

Bytes	Value	Description
Byte 1	0x00	Reserved
Byte 2	X	Connect return code that indicates connection state

Table 0.8 Connect Ack Variable Header

❖ Connect return code

Value	Connection State
0x00	Connection Accepted
0x01	Connection Refused, unacceptable protocol version
0x02	Connection Refused, identifier rejected
0x03	Connection Refused, server unavailable
0x04	Connection Refused, bad user name or password

Table 0.9 Connect Ack Return code

3. PUBLISH Packet

used in the MQTT protocol to publish a message from a client to a broker or from a broker to a client. It is the most commonly used MQTT packet type and is used to send and receive messages in the MQTT network. It consists of fixed header, variable header, and payload.

3.1. Fixed Header

Bytes	Value	Description
Byte 1	0x3X	The type is 3 and the flags can be used
Bytes 2:5	X	The Remaining Length

Table 0.10 Publish Packet Fixed Header

❖ **The flags**

Bit	Set	Reset
DUP	re-delivery of the pub packet	The first trail to send the packet
QoS Level	QoS 0	QoS 1 or 2
RETAIN	The server saves the packet until the subscriber need it	The server saves the last recently sent packet before it

Table 0.11 Publish Packet Fixed Header Flags

$$RemainingLength = 2 + Topic\ Length + Message\ Length + (2\ if\ QoS > 0)$$

Equation 0.2 Publish Packet Remaining Length

3.2. Variable Header

Bytes	Value	Description
Bytes 1:2	X	Topic Length
Bytes 3: n	X	Topic Name
Bytes n+1: n+2	X	Packet ID used when $\boldsymbol{QoS > 0}$

Table 0.12 Publish Packet Variable Header

3.3. **Payload:** actual data that is being transmitted in message.

4. PUBACK Packet

used to acknowledge the receipt of a PUBLISH packet from a client to a broker. When a client sends a PUBLISH packet to a broker, the broker sends a PUBACK packet to the client to acknowledge that the message has been received and processed successfully. Sent when $\boldsymbol{QoS > 0}$.

4.1. Fixed Header

Bytes	Value	Description
Byte 1	0x40	The type is 4 and the flags are reserved
Byte 2	0x02	The Remaining Length, the length of variable header

Table 0.13 Publish Ack Packet Fixed Header

4.2. Variable Header: The Packet ID being Acknowledged.

5. SUBSCRIBE Packet

used in the MQTT protocol to subscribe a client to one or more topics on the broker. When a client sends a SUBSCRIBE packet to the broker, it requests to receive messages published to the specified topics.

5.1. Fixed Header

Bytes	Value	Description
Byte 1	0x82	The type is 8 and the flags are reserved
Byte 2	X	The Remaining Length

Table 0.14 Subscribe Packet Fixed Header

$$RemainingLength = 2 + (3 + Topic\ Length) * Number\ of\ Topics$$

Equation 0.3 Subscribe Packet Remaining Length

5.2. Variable Header: The Packet ID used in SUBACK

5.3. Payload:

contains one or more topic filters and their corresponding QoS levels. The topic filters are strings that identify the topics to which the client wants to

subscribe, and the QoS levels are the desired quality of service levels for the subscriptions. The figure below shows the payload.

Figure 0.5 Subscribe Packet Payload

6. SUBACK:

used to acknowledge the receipt of a SUBSCRIBE packet from a client to a broker. When a client sends a SUBSCRIBE packet to the broker to subscribe to one or more topics, the broker sends a SUBACK packet to the client indicating the status of the subscription request.

6.1. Fixed Header:

Bytes	Value	Description
Byte 1	0x90	The type is 9 and the flags are reserved
Byte 2	X	The Remaining Length, the length of variable header

Table 0.15 Subscribe ACK Packet Fixed Header

$$RemaingLength = 2 + Number\ of\ topic\ Filters\ being\ Acked$$

Equation 0.4 Subscribe ACK Remaining Length

6.2. Variable Header: The Packet ID of SUBSCRIBE Packet

6.3. Payload: Contains the Return codes of SUB topic filters

- **Return Codes:**

Value	Connection State
0x00	Success - Maximum QoS 0
0x01	Success - Maximum QoS 1
0x02	Success - Maximum QoS 2
0x80	Failure

Table 0.16 Subscribe ACK Return Code

Conclusion

In conclusion, MQTT is a powerful messaging protocol that can enable communication between bare metal devices and the outside world. By utilizing a lightweight and efficient approach to message transmission, MQTT can facilitate real-time communication and data exchange between devices, even in low-bandwidth or unstable network environments.

To make bare metal devices talk to the world using MQTT, you would need to first establish a connection to the MQTT broker server. This can be achieved through a variety of means, such as using an MQTT client library or implementing the protocol directly in your device's firmware.

Once connected, your device can subscribe to specific topics on the broker to receive messages or publish messages to specific topics to communicate with other devices or applications. This can enable various use cases, such as IoT applications, sensor networks, and machine-to-machine communication.

Overall, MQTT offers a flexible and scalable solution for bare metal devices to communicate with the world, and its popularity continues to grow as more and more applications require real-time communication and data exchange.

Part 2: Web Module

Chapter 5: Hosting, AEBs, and Thingsboard:

Introduction

The Internet of Things (IoT) transforms industries and how we interact with technology. As connected devices grow, businesses seek efficient ways to manage their IoT deployments. It is where Application Enablement Platforms (AEPs) come in. AEPs provide a comprehensive suite of tools and services that enable businesses to develop, deploy and manage IoT applications quickly.

What is an IoT Application Enablement Platform?

An IoT Application Enablement Platform (AEP) is a middleware platform that connects IoT devices, applications, and services. It provides tools and services that enable businesses to develop, deploy and manage IoT applications. The platform abstracts the complexity of IoT deployments and provides an easy-to-use interface for developers, allowing them to focus on building applications instead of addressing the infrastructure.

Architecture of an AEP

The architecture of an AEP typically consists of the following layers:

- **Edge Layer:** The edge layer consists of physical devices and sensors that collect data from the industrial environment. This layer may include gateways and edge devices that preprocess data and perform local analytics.
- **Connectivity Layer:** The connectivity layer provides the mechanisms for transmitting data from the edge layer to the cloud. This layer may include protocols such as MQTT, HTTP, and CoAP, as well as transport mechanisms such as 4G/LTE, Wi-Fi, and Ethernet.
- **Cloud Layer:** The cloud layer provides the backend infrastructure for processing, storing, and analyzing data from the edge layer. This layer may include cloud services such as storage, databases, analytics, and machine learning.
- **Application Layer:** The application layer provides the tools and services for developing custom IoT applications. This layer may include APIs, SDKs, development tools for creating custom applications, and pre-built application templates for common use cases.
- **Security Layer:** The security layer provides mechanisms for securing the IIoT system from threats and vulnerabilities. This layer may include encryption, authentication, access control, intrusion detection, and prevention mechanisms.
- **Management Layer:** The management layer provides tools and services for managing the IoT system, including device management, software updates, and

system monitoring. This layer may include dashboards, analytics, and reporting tools for monitoring and managing the system.

The architectural complexity of an enterprise IoT and A-IoT platform goes through multiple stages. Six lenses under IT Management for an ideal industry fit include an enterprise ecosystem, a built-in could-agnostic infrastructure, a system secured by design, business continuity through DevOps, performance scaling with edge compatibility, and multi-tenancy via IAM or Role-based Access Management (RBAC). Further, these layers constitute field devices, application lifecycle management, device/asset monitoring (including OEM's custom SDK), application experience layer with tenant/asset onboarding functions, and device cloud connectivity (AWS, Azure, & Google).

Overall, the architecture of an IIoT AEP should be designed for scalability, reliability, and security and should provide a comprehensive platform for developing, deploying, and managing IIoT applications in an industrial environment.

Key Features of an AEP

Industrial IoT Application Enablement Platform (AEP) offers various key features that enable businesses to develop and deploy IoT solutions. These features include device management, data analytics, and integration with third-party services. However, more than having an AEP alone is needed to overcome IoT development challenges.

Every Industrial OEM requires an IoT Solutions partner who can help them scale their IoT project from MVP to a fully functional solution and customize it to their unique business needs. A reliable development partner can also help businesses overcome common hurdles in IoT development, such as security, compatibility, and reliability issues, and ensure that the IoT solution is built for success.

An ideal AEP must be:

- A build-to-deployment and cloud-agnostic tool that is proven, reliable, foundationally secured, and highly scalable
- A platform that offers an expanding array of Web2 Application Development services to accelerate the development of Enterprise-Grade Applications on your IoT and A-IoT digital platform journey, powered by advanced Middleware technology.
- An enterprise-grade multi-tenant multi-Asset performance application with life cycle management, monitoring, operations, insights, and model-building features
- An intuitive Low/Pro Code tools and workflow for advanced data handling, expanding multi-asset and multi-tenant application functionalities

- Built using industry-best secure development and deployment processes that fulfill enterprise requirements for security, scalability, disaster recovery, and multiple deployment flexibilities

The key features of an AEP must include the following:

- Device management: An AEP should provide a user-friendly interface to manage IoT devices, including provisioning, configuration, and monitoring. It should allow users to easily add, remove, and update devices and track their status and performance.
- Data Ingestion: It should be able to collect data from various IoT devices and sensors, process it, and store it in a secure and scalable database.
- Integration: It should support integration with third-party systems and services, such as cloud platforms, data lakes, and APIs.
- Security: It should provide robust security features, such as access control, authentication, encryption, and threat detection.
- Scalability: It should be able to handle large volumes of IoT data and support a growing number of IoT devices and users.
- Customization: It should provide a flexible and extensible platform that can be customized to meet the unique business needs of the user.
- Application Development: It should enable users to develop and deploy IoT applications quickly and easily using pre-built templates, libraries, and tools.

ThingsBoard

ThingsBoard is an open-source IoT platform that enables rapid development, management, and scaling of IoT projects. Our goal is to provide the out-of-the-box IoT cloud or on-premises solution that will enable server-side infrastructure for your IoT applications. That has some features that make you able to:

- Provision devices, assets and customers, and define relations between them.
- Collect and visualize data from devices and assets.
- Analyze incoming telemetry and trigger alarms with complex event processing.
- Control your devices using remote procedure calls (RPC).
- Build work-flows based on a device life-cycle event, REST API event, RPC request, etc.
- Design dynamic and responsive dashboards and present device or asset telemetry and insights to your customers.
- Push device data to other systems.

Architecture

ThingsBoard is designed to be:

- **Scalable**: horizontally scalable platform, build using leading open-source technologies.
- **Fault-tolerant**: no single-point-of-failure, every node in the cluster is identical.
- **Robust and efficient**: single server node can handle tens, hundreds, or even thousands of devices depending on use-case. ThingsBoard cluster can handle millions of devices.
- **Durable**: never lose your data. ThingsBoard supports various queue implementations to provide extremely high message durability.
- **Customizable**: adding new functionality is easy with customizable widgets and rule engine nodes.

The diagram below shows key system components and interfaces they provide.

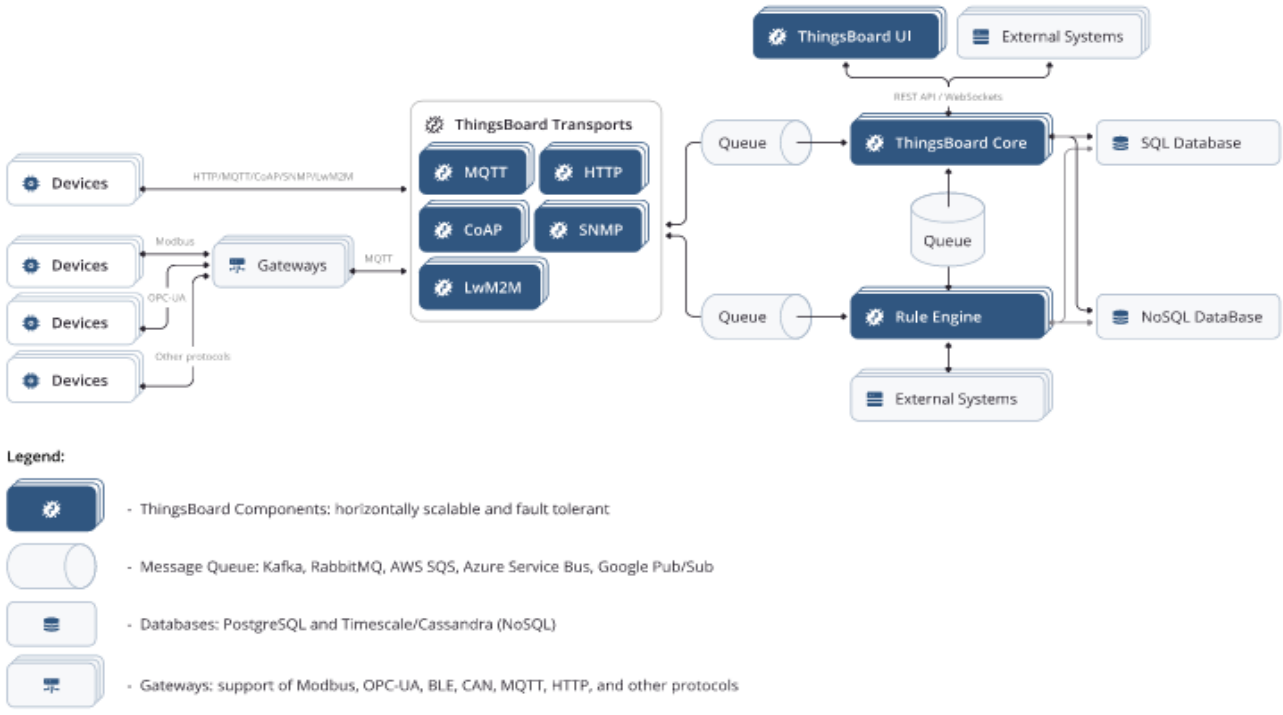

Figure 0.1 ThingsBoard Components and Interfaces

ThingsBoard Transports

ThingsBoard provides MQTT, HTTP, CoAP and LwM2M based APIs that are available for your device applications/firmware. Each of the protocol APIs are

provided by a separate server component and is part of ThingsBoard “Transport Layer”. MQTT Transport also provides Gateway APIs to be used by gateways that represent multiple connected devices and/or sensors.

Once the Transport receives the message from device, it is parsed and pushed to durable Message Queue. The message delivery is acknowledged to device only after corresponding message is acknowledged by the message queue.

ThingsBoard Core

ThingsBoard Core is responsible for handling REST API calls and WebSocket subscriptions. It is also responsible for storing up to date information about active device sessions and monitoring device connectivity state. ThingsBoard Core uses Actor System under the hood to implement actors for main entities: tenants and devices. Platform nodes can join the cluster, where each node is responsible for certain partitions of the incoming messages.

ThingsBoard Rule Engine

ThingsBoard Rule Engine is the heart of the system and is responsible for processing incoming messages. Rule Engine uses Actor System under the hood to implement actors for main entities: rule chains and rule nodes. Rule Engine nodes can join the cluster, where each node is responsible for certain partitions of the incoming messages.

Rule Engine subscribes to incoming data feed from queue(s) and acknowledge the message only once it is processed. There are multiple strategies available that control the order or message processing and the criteria of message acknowledgement.

ThingsBoard Rule Engine may operate in two modes: shared and isolated. In shared mode, rule engine process messages that belong to multiple tenants. In isolated mode Rule Engine may be configured to process messages for specific tenant only

ThingsBoard Web UI

ThingsBoard provides a lightweight component written using Express.js framework to host static web ui content. Those components are completely stateless and no much configuration available. The static web UI contains application bundle. Once it is loaded, the application starts using the REST API and WebSockets API provided by ThingsBoard Core.

Message Queues

ThingsBoard supports multiple message queue implementations: Kafka, RabbitMQ, AWS SQS, etc. Using durable and scalable queues allow ThingsBoard to implement back-pressure and load balancing. Back-pressure is extremely important in case of peak loads.

On-premise vs cloud deployments

ThingsBoard supports both on-premise and cloud deployments. With more than 5000 ThingsBoard servers running all over the world, ThingsBoard is running in production on AWS, Azure, GCE and private data centers. It is possible to launch ThingsBoard in the private network with no internet access at all.

Standalone vs cluster mode

Platform is designed to be horizontally scalable and supports automatic discovery of new ThingsBoard servers (nodes). All ThingsBoard nodes inside cluster are identical and are sharing the load. Since all nodes are identical there is no “master” or “coordinator” processes and there is no single point of failure. The load balancer of your choice may forward request from devices, applications and users to all ThingsBoard nodes.

Monolithic vs microservices architecture

It is possible to run the platform as a monolithic application or as a set of microservices. Supporting both options requires some additional programming efforts, however, is critical due to back-ward compatibility with variety of existing installations.

Approximately 80% of the platform installations are still using monolithic mode due to minimum support efforts, knowledge and hardware resources to do the setup and low maintenance efforts.

However, if you do need high availability or would like to scale to millions of devices, then microservices is a way to go. There are also some challenges that are solved with microservices architecture and applicable for more complex deployments and usage scenarios. For example, running a multi-tenant deployment where one need more granular isolation to protect from:

- unpredictable load spikes
- unpredictable rule chain misconfiguration.
- single devices opening 1000s of concurrent connections due to firmware bugs.
- And many other cases.

SQL vs Hybrid database approach:

ThingsBoard uses database to store entities (devices, assets, customers, dashboards, etc.) and telemetry data (attributes, timeseries sensor readings, statistics, events). Platform supports three database options at the moment:

- SQL - Stores all entities and telemetry in SQL database. ThingsBoard authors recommend to use PostgreSQL and this is the main SQL database that ThingsBoard supports. It is possible to use HSQLDB for local

development purposes. We do not recommend to use HSQLDB for anything except running tests and launching dev instance that has minimum possible load.

- Hybrid (PostgreSQL + Cassandra) - Stores all entities in PostgreSQL database and timeseries data in Cassandra database.

Key Concepts:

Entities

Overview: ThingsBoard provides the user interface and REST APIs to provision and manage multiple entity types and their relations in your IoT application. Supported entities are:

- Tenants - you can treat the tenant as a separate business-entity: it's an individual or an organization who owns or produce devices and assets. Tenant may have multiple tenant administrator users and millions of customers, devices and assets.
- Customers - the customer is also a separate business-entity: individual or organization who purchase or uses tenant devices and/or assets. Customer may have multiple users and millions of devices and/or assets.
- Users - users are able to browse dashboards and manage entities.
- Devices - basic IoT entities that may produce telemetry data and handle RPC commands. For example, sensors, actuators, switches.
- Assets - abstract IoT entities that may be related to other devices and assets. For example, factory, field, vehicle.
- Entity Views - useful if you like to share only part of device or asset data to the customers.
- Alarms - events that identify issues with your assets, devices, or other entities.
- Dashboards - visualization of your IoT data and ability to control particular devices through the user interface.
- Rule Node - processing units for incoming messages, entity lifecycle events, etc.
- Rule Chain - defines the flow of the processing in the Rule Engine. May contain many rule nodes and links to other rule chains

Each entity supports:

- Attributes - static and semi-static key-value pairs associated with entities. For example, serial number, model, firmware version.
- Time-series data - time-series data points available for storage, querying and visualization. For example, temperature, humidity, battery level.
- Relations - directed connections to other entities. For example, contains, manages, owns, and produces.

Some entities support profiles:

- Tenant Profiles - contains common settings for multiple tenants: entity, API and rate limits, etc. Each Tenant has the one and only profile at a single point in time.
- Device Profiles - contains common settings for multiple devices: processing and transport configuration, etc. Each Device has the one and only profile at a single point in time.
- Asset Profiles - contains common settings for multiple assets: processing configuration, etc. Each Asset has the one and only profile at a single point in time.

Relations:

Entity relation defines connection between two ThingsBoard entities that belong to the same Tenant. The relation has an arbitrary type: Contains, Manages, Supports, etc. The relation is also directional. You may treat ThingsBoard relations as a Has-a relationship from object-oriented programming.

Relations help to model physical world objects in ThingsBoard. The easiest way to understand them is using the example. Let's assume we want to build an application that collects data from soil moisture and temperature sensors, visualize this data on the dashboard, detect issues, raise alarms and control the irrigation. Let's also assume we want to support multiple fields with hundreds of sensors. Fields may be also grouped into the Geo regions.

The following diagram explains how those entities are configured and stored in ThingsBoard:

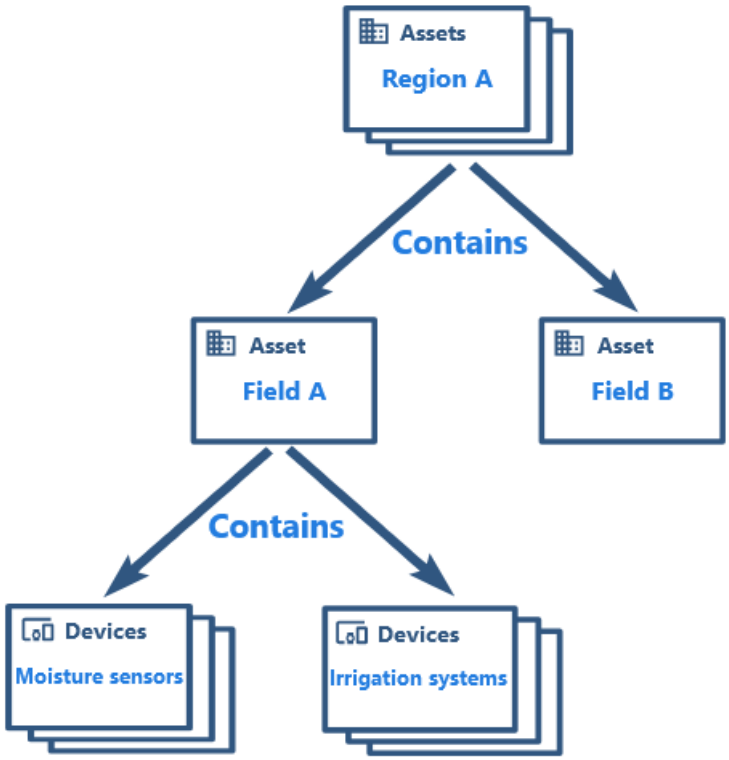

Figure 0.2 ThingsBoard Entities and Configurations

Device Profiles

A tenant administrator is able to configure common settings for multiple devices using Device Profiles. Each Device has one and only profile at a single point in time.

Device Profile settings:

By default, the Root Rule Chain processes all incoming messages and events for any device. However, the more different device types you have, the more complex your Root Rule Chain may become. Many platform users create their Root Rule Chain for the sole purpose of sending messages to specific rule chains depending on the device type.

To avoid this painful and mundane activity, you can specify a custom Rule Chain for your devices. The new Rule Chain will receive all telemetry, device activity (Active/Inactive), and device lifecycle (Created/Updated/Deleted) events. This setting is available in the Device Profile wizard and in the Device Profile details.

Queue Name:

By default, the Main queue will be used to store all incoming messages and events from any device. The transport layer will submit messages to this queue and Rule Engine will poll the queue for new messages. However, for multiple use cases, you might want to use different queues for different devices. For example, you

might want to isolate data processing for Fire Alarm/Smoke Detector sensors and other devices. This way, even if your system has a peak load produced by millions of water meters, whenever the Fire Alarm is reported, it will be processed without delay. Separation of the queues also allows you to customize different submit and processing strategies.

Default transport type

The Default transport type is intended for backward compatibility with previous releases. With the Default transport type, you can continue to use the platform's default MQTT, HTTP, CoAP and LwM2M APIs to connect your devices. There is no specific configuration setting for the default transport type. The main focus is on MQTT and HTTP for this project which shall be explained later on.

- **MQTT transport type**

  The MQTT transport type enables advanced MQTT transport settings. Now you are able to specify custom MQTT topics filters for time-series data and attribute updates that correspond to the telemetry upload API and attribute update API, respectively.

- **MQTT device topic filters**

  Custom MQTT topic filters support single '+' and multi-level '#' wildcards and allow you to connect to almost any MQTT based device that sends a payload using JSON.

Alarm Rules

Platform users can use Rule Engine to configure alarms. Rule Engine is a quite powerful feature, but it requires some programming skills. Alarm Rules have been introduced to simplify the process of configuring the most popular alarm types. Now you don't need to be the Rule Engine guru to configure your processing logic. Under the hood, Rule Engine evaluates Alarm Rules using the "Device Profile" rule node. Alarm Rule consists of the following properties:

- **Alarm Type** - a type of Alarm. Alarm type must be unique within the device profile alarm rules.
- **Create Conditions** - defines the criteria when the Alarm will be created/updated. The condition consists of the following properties:
  - Severity - will be used to create/update an alarm. ThingsBoard verifies Create Conditions in the descending order of the severity. For example, if a condition with Critical severity is true, the platform will raise alarm with Critical severity, and "Major", "Minor" or "Warning" conditions will not be evaluated. Severity must be unique per alarm rule (e.g., two conditions created within the same alarm rule can't have the same severity).

- Key Filters - list of logical expressions against attributes or telemetry values. For example, *"(temperature < 0 OR temperature > 20) AND softwareVersion = '2.5.5'"*.

- Condition Type - either simple, duration, or repeating. For example, *3 times in a row* or *during 5 minutes*. The simple condition will raise an alarm once the first matching event occurrs.

- Schedule - defines the time interval during which the rule is active. Either "active all the time", "active at specific time" or "custom".

- Details - the alarm details template supports substitution of the telemetry and/or attribute values using ${attributeName} syntax.

- **Clear condition** - defines criteria when the Alarm will be cleared.
- **Advanced settings** - defines alarm propagation to related assets, customers, tenant, or other entities.

Device Attributes

ThingsBoard provides the ability to assign custom attributes to your entities and manage these attributes. Those attributes are stored in the database and may be used for data visualization and data processing.

Attribute types

There are three types of attributes. Let's review them with examples:

- **Server-side attributes**

  This type of attribute is supported by almost any platform entity: Device, Asset, Customer, Tenant, User, etc. Server-side attributes are the ones that you may configure via Administration UI or REST API. The device firmware can't access the server-side attribute.

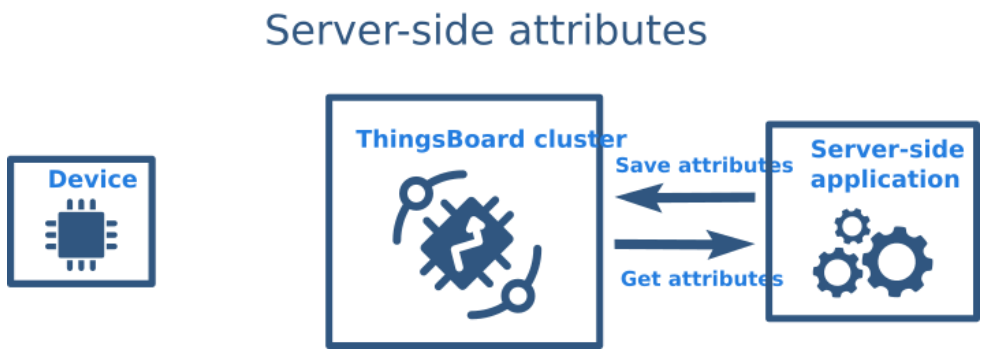

Figure 0.3 Server Side attributes

For example, with thermostat devices, The *maxTemperatureThreshold* and *temperatureAlarmEnabled* may be used to configure and enable/disable alarms for a certain device or asset.

- **Shared attributes**

  This type of attributes is available only for Devices. It is similar to the Server-side attributes but has one important difference. The device firmware/application may request the value of the shared attribute(s) or subscribe to the updates of the attribute(s). The devices which communicate over MQTT or other bi-directional communication protocols may subscribe to attribute updates and receive notifications in real-time. The devices which communicate over HTTP or other request-response communication protocols may periodically request the value of shared attribute.

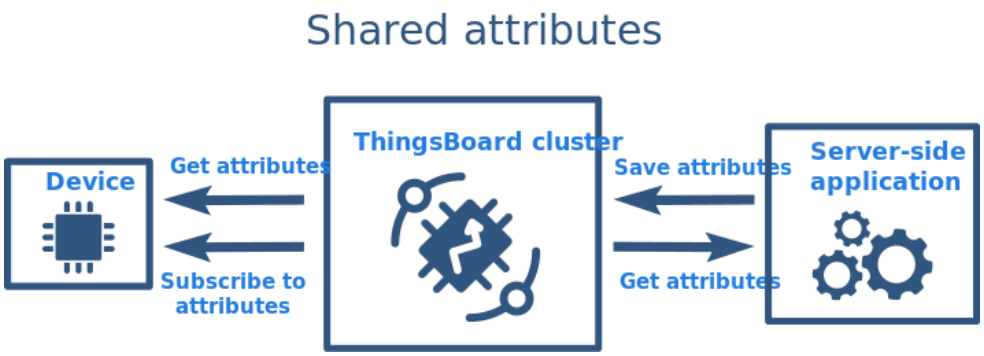

Figure 0.4 Shared Attribute

The most common use case of shared attributes is to store device settings.

- Client-side attributes

  This type of attributes is available only for Devices. It is used to report various semi-static data from Device (Client) to ThingsBoard (Server). It is similar to shared attributes, but has one important difference. The device firmware / application may send the value of the attributes from device to the platform.

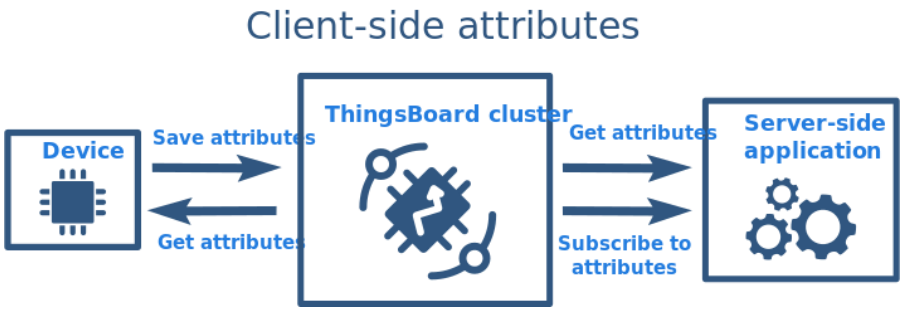

Figure 0.5 Client Side Attribute

The most common use case of client attributes is to report device state.

➢ Telemetry/Time-Series data

ThingsBoard provides a rich set of features related to time-series data:

- Collect data from devices using various protocols and integrations.
- Store time series data in SQL (PostgreSQL) or NoSQL (Cassandra or Timescale) databases.
- Query the latest time series data values or all data within the specified time range with flexible aggregation.
- Subscribe to data updates using WebSocket for visualization or real-time analytics
- Visualize time series data using configurable and highly customizable widgets and dashboards.
- Filter and analyze data using flexible Rule Engine.
- Generate alarms based on collected data.

➢ Data Points

ThingsBoard internally treats time-series data as timestamped key-value pairs. We call single timestamped key-value pair a data point. Flexibility and simplicity of the key-value format allow easy and seamless integration with almost any IoT device on the market. Key is always a string and is basically a data point key name, while the value can be either string, Boolean, double, integer or JSON.

➢ Alarms

ThingsBoard provides the ability to create and manage alarms related to your entities: devices, assets, customers, etc. For example, you may configure ThingsBoard to automatically create an alarm when the temperature sensor reading is above a certain threshold. Of course, this is a very simplified case, and real scenarios can be much more complex

➢ Main concepts

- **Originator:** Alarm originator is an entity that causes the alarm. For instance, Device A is an originator of the alarm if ThingsBoard receives a temperature reading from it and raise the “High Temperature” alarm because the reading exceeds the threshold.
- **Type:** Alarm type helps to identify the root cause of the alarm. For example, “HighTemperature” and “LowHumidity” are two different alarms.
- **Severity:** Each alarm has severity which is either Critical, Major, Minor, Warning, or Indeterminate (sorted by priority in descending order).
- **Lifecycle:** Alarm may be active or cleared. When ThingsBoard creates an alarm, it persists the start and end time of the alarm. By default, the start time and the end time are the same. If the alarm trigger condition repeats, the platform updates the end time. ThingsBoard can automatically clear the alarm when an event occurs

that matches an alarm clearing condition. Alarm clear condition is optional. A user can clear the alarm manually.

Besides active and cleared alarm state, ThingsBoard also keeps track of whether someone has acknowledged the alarm. Alarm acknowledgment is possible via the dashboard widget, or an entity details tab.

To summarize, there are 4 possible values of the “status” field:

- Active unacknowledged (ACTIVE_UNACK) - alarm is not cleared and not acknowledged yet.
- Active acknowledged (ACTIVE_ACK) - alarm is not cleared, but already acknowledged.
- Cleared unacknowledged (CLEARED_UNACK) - alarm was already cleared, but not yet acknowledged.
- Cleared acknowledged (CLEARED_ACK) - alarm was already cleared and acknowledged.

Alarm uniqueness: ThingsBoard identifies alarm using a combination of originator, type, and start time. Thus, at a single point in time, there is only one active alarm with the same originator, type, and start time.

Dashboards

ThingsBoard provides the ability to create and manage dashboards. Each dashboard can contain plenty of widgets. Dashboards display data from many entities: devices, assets, etc. Dashboards can be assigned to Customers.

Overview

Title: Editable dashboard title appears in the top-left corner of the dashboard. Dashboard title is displayed in the application breadcrumbs to simplify the navigation. You can change and adjust a Title in the settings of the dashboard.

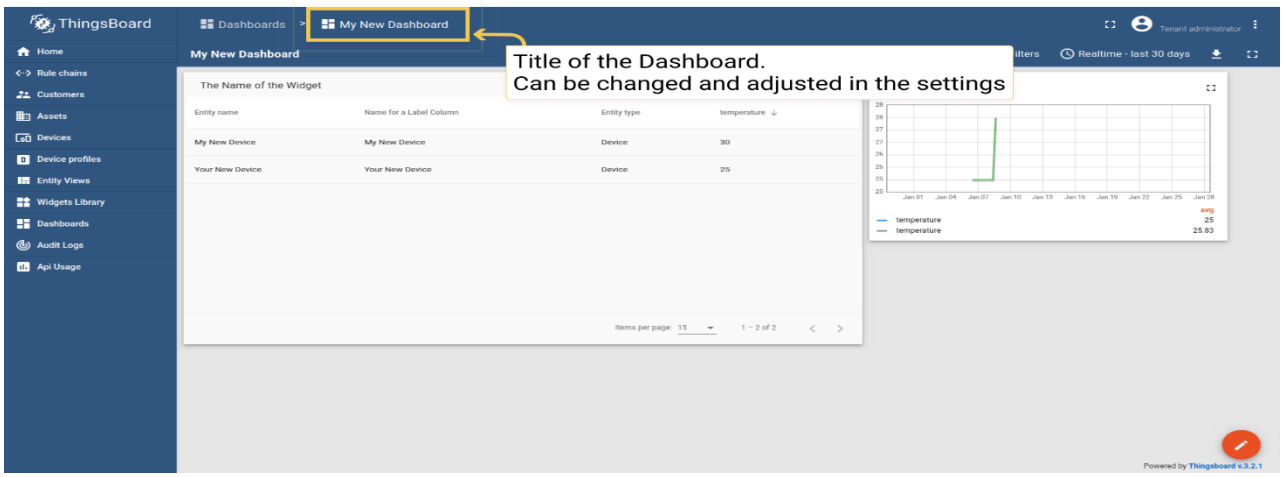

Figure 0.6 Dashboard

Edit mode: Use the “pencil” button in the bottom-right corner of the screen to enter dashboard edit mode. Once you switch to the edit mode, you can add new widgets and access controls in the dashboard toolbar. You can also use button “Apply changes” to save the dashboard or “Decline changes” to rollback all unsaved changes.

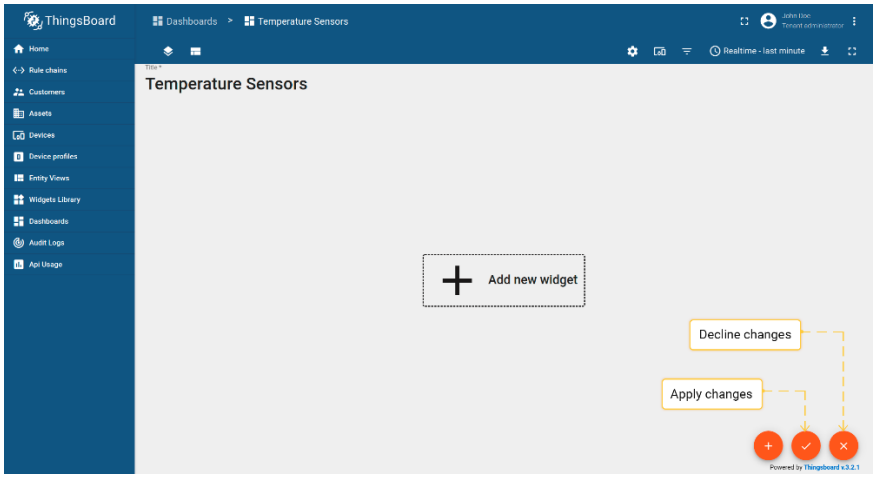

Figure 0.7 Dashboard Edit

Toolbar: Dashboard toolbar allows you to manage states, layouts, settings, aliases, filters, export dashboard in the needed format, version control and configure timewindow using the corresponding icons in the toolbar. Some of these icons (states, layout, settings, entity aliases, filters, version control) are visible only in the “edit” mode. All other icons are visible in both “view” and “edit” mode. You can hide these icons or configure the toolbar to be minimized by default using settings.

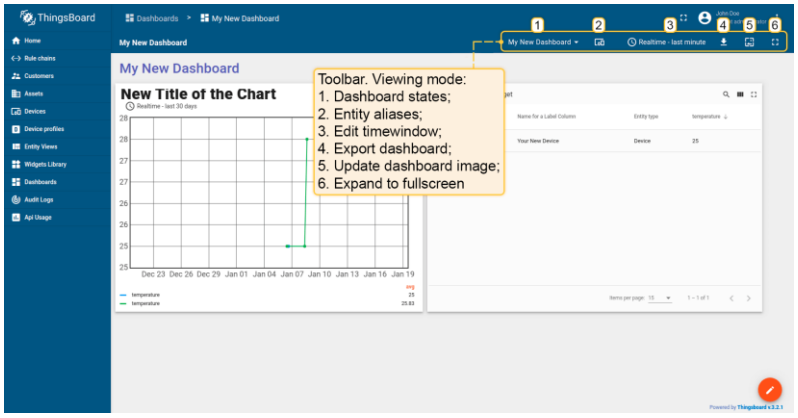

Figure 0.8 Dashboard Toolbar

➢ Entity Aliases

Entity Aliases determine which entities (devices, assets, etc.) you would like to display on the dashboard. You can treat alias as a reference to one or more devices. These references can be static or dynamic. An example of the static alias is the Single entity alias. An entity is configured once in the alias dialog box. All users see the same data if they have permission to access this device. An example of a dynamic alias is the

Device type alias, which displays all devices of a certain type (e.g., “Thermometer”). This alias is dynamic because the list of devices depends on the user using the dashboard.

➢ Widgets

▪ Widget types

All IoT dashboards are constructed using ThingsBoard widgets that are displayed in Widget library. Each widget provides end-user functions such as data visualization, remote device control, alarms management and displaying static custom html content. There are five types of widgets:

- Time series widgets display data for a specific time window. The time window may be real-time (e.g., for the last 24 hours) or historical (December 2020). Examples of time series widgets are chart widgets. Obviously, time series widgets are designed to display time series and not attributes.
- Latest values widgets display the latest values of particular attribute or time series keys. For example, device model or latest temperature reading.
- Control widgets allow you to send RPC commands to your devices. For example, control desired temperature on the thermostat device.
- Alarm widgets allow you to display alarms.
- Static widgets are designed to display static data. For example, floor plan or static company information.

ThingsBoard IoT Gateway

The ThingsBoard IoT Gateway is an open-source solution that allows you to integrate devices connected to legacy and third-party systems with ThingsBoard

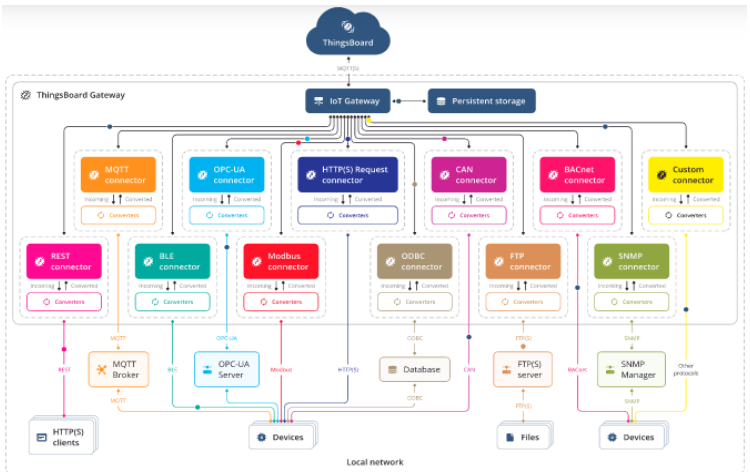

Figure 0.9 Things Board Gateway

- Gateway features

  ThingsBoard IoT Gateway provides following features including (and not limited to):

  - MQTT connector to control, configure and collect data from IoT devices that are connected to external MQTT brokers using existing protocols
  - REST connector to create endpoints and collect data from incoming HTTP requests Custom connector to collect data from IoT devices that are connected by different protocols. (You can create your own connector for the required protocol).
  - Persistence of collected data to guarantee data delivery in case of network or hardware failures.
  - Automatic reconnect to ThingsBoard cluster.
  - Simple yet powerful mapping of incoming data and messages to unified format.

- Architecture

  The IoT Gateway is a software component that is designed to run on a Linux based microcomputers that support Python 3.7+. Main components of ThingsBoard IoT Gateway are listed below.

  - **Connector:** The purpose of this component is to connect to external system (e.g., MQTT broker or OPC-UA server) or directly to devices (e.g. Modbus, BLE or CAN). Once connected, connector is either poll data from those systems or subscribe to updates. Poll vs subscribe depends on the protocol capabilities. For example, we use subscription model for MQTT connectors and polling for Modbus and CAN. Connector is also able to push updates to devices either directly or via external systems.
  - **Converter:** Converters are responsible for converting data from protocol specific format to/from ThingsBoard format. Converters are invoked by Connectors. Converters are often specific to protocol supported by Connector. There are uplink and downlink converters. Uplink converter is used to convert data from specific protocol to ThingsBoard format. Downlink converter is used to convert messages from ThingsBoard to specific protocol format.
  - **Event Storage:** Event Storage is used to temporary store the telemetry and other events produced by Connectors until they are delivered to ThingsBoard. Event Storage support two implementations: in-memory queue and persistent file storage. Both implementations make sure that your device data is eventually delivered in case of network outages. In-memory queue minimizes the IO operations but may lose message in case of gateway process restart. Persistent file storage survives the restart of the process but executes IO operations to the file system.
  - **ThingsBoard Client:** The Gateway communicates to ThingsBoard via MQTT protocol and uses MQTT Gateway API. ThingsBoard Client is a separate thread

that polls Event Storage and delivers messages once connection to ThingsBoard is active. ThingsBoard Client supports monitoring of the connectivity, batching the events for performance improvement and many other features.

- **Gateway Service:** The Gateway Service is responsible for bootstrap of the Connectors, Event Storage and ThingsBoard Client. This Service collects and periodically reports statistics to ThingsBoard about incoming messages and connected devices. Gateway Service persists list of connected devices to be able to re-subscribe to device configuration updates in case of the restart of the gateway.

➢ IoT Gateway Configuration

- General configuration file: The main configuration file that is used for connection to ThingsBoard platform instance and enable/disable connectors. This configuration points to ThingsBoard instance thingsboard.cloud (or host's IP address) and uses memory file storage configured to store maximum of 100,000 records. There are different connectors, we're only activating MQTT and REST connectors only here.

```
thingsboard:
  host: thingsboard.cloud
  port: 1883
  remoteShell: false
  remoteConfiguration: false
  statistics:
    enable: true
    statsSendPeriodInSeconds: 3600
  minPackSendDelayMS: 0
  checkConnectorsConfigurationInSeconds: 60
  handleDeviceRenaming: true
  checkingDeviceActivity:
    checkDeviceInactivity: false
    inactivityTimeoutSeconds: 120
    inactivityCheckPeriodSeconds: 10
  security:
    accessToken: PUT_YOUR_ACCESS_TOKEN_HERE
  qos: 1
storage:
  type: memory
  read_records_count: 100
  max_records_count: 100000
grpc:
  enabled: false
  serverPort: 9595
  keepaliveTimeMs: 10000
  keepaliveTimeoutMs: 5000
  keepalivePermitWithoutCalls: true
  maxPingsWithoutData: 0
  minTimeBetweenPingsMs: 10000
  minPingIntervalWithoutDataMs: 5000
connectors:
  -
    name: MQTT Broker Connector
    type: mqtt
    configuration: mqtt.json
  -
```

```
    name: Modbus Connector
    type: modbus
    configuration: modbus.json

  -
    name: Modbus Connector
    type: modbus
    configuration: modbus_serial.json

  -
    name: OPC-UA Connector
    type: opcua
    configuration: opcua.json

  -
    name: BLE Connector
    type: ble
    configuration: ble.json

  -
    name: CAN Connector
    type: can
    configuration: can.json

  -
    name: Custom Serial Connector
    type: serial
    configuration: custom_serial.json
    class: CustomSerialConnector
```

- Sections in config file:
  - **ThingsBoard – Configuration for connecting to ThingsBoard platform.**
  - **security – Configuration for type of encryption and authorization.**
  - **storage – Configuration for local storage of incoming data from devices.**
  - **connectors – Array of Connectors and their configuration to use.**
    - **Connection to ThingsBoard**

Parameter	Default value	Description
ThingsBoard		Configuration for connection to server.
Host	**thingsboard.cloud**	Hostname or ip address of ThingsBoard server.
Port	**1883**	Port of mqtt service on ThingsBoard server.

Table 0.1 MQTT Configuration Parameters

- **Subsection "statistics"**

This subsection uses to configure collecting statistics data and sending them to ThingsBoard Gateway device attributes.

Parameter	Default value	Description
Statistics		Configuration for enabling statistics collecting
… enable	True	A boolean value that is used to on/off collecting statistics
… statsSendPeriodInSeconds	3600	An integer value that is used to send data every period
… configuration	statistics.json	Name of a configuration file for additional user statistics data

Table 0.2 Subsection

- Subsection "checkingDeviceActivity"

This subsection is optional and used to monitor the activity of each connected device. If you define this section, the Gateway will check the activity of each device every n second that means, if the device is inactive for n seconds, it will disconnect it.

Parameter	**Default value**	**Description**
checkingDeviceActivity		Configuration for checking devices activity
… checkDeviceInactivity	**false**	Boolean value that is used to on/off checking device activity
… inactivityTimeoutSeconds	**120**	Inactivity device time after whose Gateway will disconnect it
… inactivityCheckPeriodSeconds	**10**	Periodicity of device activity check

Table 0.3 checking Device Activity

One type of security configuration is using clientId, username, and password, to get them you should login into your ThingsBoard platform instance, go to DEVICE tab, press the plus icon, fill the values and check the "Is gateway" option, press "Next: Credentials", check "Add credentials", select credentials type "MQTT Basic" and replace default with your value.

Parameter	**Default value**	**Description**
clientId	**clientId**	MQTT client id for the gateway form ThingsBoard server
username	**Admin**	MQTT username for the gateway form ThingsBoard server
password	**Admin**	MQTT password for the gateway form ThingsBoard server

Table 0.4 MQTT Parameters

- **Storage configuration**

Configs in storage subsection provides configuration for saving incoming data before it will be sent to ThingsBoard platform. There are 2 variants for this section: memory or file.

1. **Memory** storage - Received data saving to the RAM memory.
2. **File** storage - Received data saving to the hard drive.
3. **SQLite** storage - Received data saving to the .db file.

Parameter	Default value	Description
Type	**memory**	Storage type (Saving data to RAM, no save to hard drive).
read_records_count	**10**	Count of messages to get from storage and send to ThingsBoard.
max_records_count*	**100**	Maximum count of data in storage before send to ThingsBoard.

Table 0.5 Storage configuration

If receive data when storage has already counted, described in this parameter, new data will lose.

- **Connectors configuration**

Configs in connectors section configuration for connecting to devices by implemented protocols. Config for every connector in this section must have parameters as in table below:

Parameter	Default value	Description
useGRPC	**True**	**OPTIONAL** parameter that is used to on/off GRPC transport for default connector realisation
Name	**MQTT Broker Connector**	Name of connector to broker.
Type	**mqtt**	Type of connector, must be like a name of folder, contained configuration file.
Configuration	**mqtt.json**	Name of the file with configuration in config folder.

Table 0.6 Folder with this configuration file

- MQTT Connector Configuration

The purpose of this Connector is to connect to external MQTT broker and subscribe to data feed from devices. Connector is also able to push data to MQTT brokers based on the updates/commands from ThingsBoard. This Connector is useful when you have local MQTT broker in your facility or corporate network and you would like to push data from this broker to ThingsBoard.

- Connector configuration: mqtt.json

Connector configuration is a JSON file that contains information about how to connect to external MQTT broker, what topics to use when subscribing to data feed and how to process the data.

- Key Configuration Sections:
  - Section “broker”

Parameter	Default value	Description
name	Default Local Broker	Broker name for logs and saving to persistent devices.
host	Localhost	Mqtt broker hostname or ip address.
port	1883	Mqtt port on the broker.
clientId	ThingsBoard_gateway	This is the client ID. It must be unique for each session.
Version	5	MQTT protocol version.

Table 0.7 MQTT Parameters

- Subsection “security”

One type of security configuration is basic, for authorization will be used combination of username/password, provided in this section in config.

Parameter	Default value	Description
Type	**basic**	Type of authorization.
username	**username**	Username for authorization.
password	**password**	Password for authorization.

Table 0.8 MQTT Security

- Section “mapping”

This configuration section contains array of topics that the gateway will subscribe to after connecting to the broker and settings about processing incoming messages.

Parameter	Default value	Description
topicFilter	**/sensor/data**	Topic address for subscribing.

Table 0.9 Topic Mapping

The **topicFilter** supports special symbols: ‘#’ and ‘+’ to allow to subscribe to multiple topics. Let’s assume we would like to subscribe and process following data from Thermometer devices:

Example Name	Topic	Topic Filter	Payload	Comments
Example 1	/sensor/data	/sensor/data	{"serialNumber": "SN-001", "sensorType": "Thermometer", "sensorModel": "T1000", "temp": 42, "hum": 58}	Device Name is part of the payload
Example 2	/sensor/SN-001/data	/sensor/+/data	{"sensorType": "Thermometer", "sensorModel": "T1000", "temp": 42, "hum": 58}	Device Name is part of the topic

Table 0.10 MQTT Topic Filter Example

- **Subsection “converter”**

  This subsection contains configuration for processing incoming messages.

Types of mqtt converters:

1. json – Default converter
2. raw – Raw default converter
3. custom – Custom converter

Bytes converter is default converter, it looks for deviceName, deviceType, attributes and telemetry in the incoming message from the broker, with rules, described in this subsection:

Parameter	Default value	Description
Type	**bytes**	Provides information to connector that default converter to be used for converting data from topic.
deviceNameExpression	**[0:4]**	The expression that is used to find device name in the incoming message.
deviceTypeExpression	**[1:3]**	The expression that is used to find device type in the incoming message.
Timeout	**60000**	Timeout for triggering “Device Disconnected” event.
attributes:		This subsection contains parameters of the incoming message, to be interpreted as attributes for the device.
… type	**raw**	Type of incoming data for a current attribute.
… key	**temp**	Attribute name, to be sent to ThingsBoard instance.
… value	**[:]**	Final view of data that will be sent to ThingsBoard, [:] - will replace to device data using python slice rules.
timeseries:		This subsection contains parameters of the incoming message, to be interpreted as telemetry for the device.
… type	**raw**	Type of incoming data for a current telemetry.
… key	**rawData**	Telemetry name, to be sent to ThingsBoard instance.
… value	**[4:]**	Final view of data that will be sent to ThingsBoard, [:] - will replace to device data using python slice rules.

Table 0.11 Converter

- Section “connectRequests”

ThingsBoard allows sending RPC commands and notifications about device attribute updates to the device. But in order to send them, the platform needs to know if the target device is connected and what gateway or session is used to connect the device at the moment. If your device is constantly sending telemetry data - ThingsBoard already knows how to push notifications. If your device just connects to MQTT broker and waits for commands/updates, you need to send a message to the Gateway and inform that device is connected to the broker.

I. Name in a message from broker:

Parameter	Default value	Description
topicFilter	**/sensor/connect**	Topic address on the broker, where the broker sends information about new connected devices.
deviceNameJsonExpression	**${serialNumber}**	JSON-path expression, for looking the new device name.

Table 0.12 Name in Message

II. Name in topic address:

Parameter	Default value	Description
topicFilter	**/sensor/+/connect**	Topic address on the broker, where the broker sends information about new connected devices.
deviceNameTopicExpression	**(?<=sensor\/)(.*?)(?=\/connect)**	Regular expression for looking the device name in topic path.

Table 0.13 Name in Topic

Chapter 6: AI Module

Abstract:

Agriculture is the backbone of a country. It is important to note that without agriculture, there is no economic growth in the country. As technology continues to improve rapidly day by day, these technologies can be utilized in farming and agriculture so that there will be maximum utilization of crops and less wastage of crops. To achieve this, we need to come across a few challenges. Which crops can be grown depending on certain weather conditions? Identification of disease in crops so that we can prevent it and maximum yield of crops. Prevention is better than cure the famous quote says (1).

But their rapid identification remains difficult in many parts of the world due to the lack of the necessary infrastructure. The combination of increasing global smart phone penetration and recent advances in computer vision made possible by deep learning has paved the way for smart phone-assisted disease diagnosis (2).

Artificial Intelligence is one of the greatest inventions; using AI we can train the machine with images to detect disease in crops. The problem of the underutilization of crops can be addressed (1).

In this thesis, convolutional neural network models were developed to perform plant disease detection and diagnosis using simple leaves images of healthy and diseased plants, through deep learning methodologies. Training of the models was performed with the use of an open database of images, containing different plants of [plant, disease] combinations, including healthy plants. Several model architectures were trained. The significantly high success rate makes the model a very useful advisory or early warning tool, and an approach that could be further expanded to support an integrated plant disease identification system to operate in real cultivation conditions (3).

Introduction:

Agriculture is essential in the country's economic growth. The primary sector of an economy comprises agriculture and other activities which is significant to the Gross Domestic Product (GDP). For decades, agriculture has been associated with the productions of a maximum of crops, food, and other raw materials to the country. At present, agriculture includes farming, fruits, vegetables, dairy, mushrooms, etc.

Agriculture not only provides food it also provides employment opportunities to a very large percentage of the population of a country. Crops are living plants grown by farmers. The majority of the crops are food-related, such as fruit, vegetables, and grains. Some crops are grown for fabrics like cotton, pharmaceuticals like quinine, and other products like wood and rubber. Most of the crops are destroyed due to the rain or due to other climatic conditions. Most of the crops are grown and are affected by diseases, and these diseases are spread over the crops. If we overcome these challenges, we can achieve maximum yield and we can overcome the underutilization of crops (1).

Plant disease diagnosis through optical observation of the symptoms on plant leaves, incorporates a significantly high degree of complexity. Due to this complexity and to the large number of cultivated plants and their existing phytopathological problems, even experienced agronomists and plant pathologists often fail to successfully diagnose specific diseases, and are consequently led to mistaken conclusions and treatments.

The existence of an automated computational system for the detection and diagnosis of plant diseases, would offer a valuable assistance to the agronomist who is asked to perform such diagnoses through optical observation of leaves of infected plants (Mohanty et al., 2016; Yang and Guo, 2017).

Objectives:

The main objective of our model is to help farmers in the prevention of crop diseases at an earlier stage by using deep learning techniques such as image classification, object detection, and semantic segmentation of computer vision.

If we speak on classification, Farmers have been able to identify crop diseases with their naked eye since the beginning of time and continue to do so on a daily basis, which forces them to make difficult decisions about which fertilizers to employ. It necessitates a thorough understanding of disease kinds as well as a great deal of expertise to ensure accurate disease identification. Farmers are frequently perplexed by diseases that appear to be almost identical. By using my classification model, they can accurately diagnose crop diseases. And with the object detection model the occurrence of disease in a plant leaf is detected. All the affected area is identified using the bonded box. By doing this, farmers can apply necessary medicines to that area or remove that part of the plant before it affects the entire plan.

Background:

Plant disease detection and diagnosis is a very important research area in the field of machine vision. It is a technology that uses machine vision equipment to acquire images and judge whether there are diseases and pests in the collected plant images. At present, machine vision-based plant disease detection and diagnosis equipment has been initially applied in agriculture and has replaced traditional naked eye identification to some extent.

For traditional machine vision-based plant disease detection and diagnosis methods, conventional image processing algorithms or manual design of features plus classifiers are often used. This kind of method usually makes use of the different properties of plant diseases and pests to design the imaging scheme and chooses an appropriate light source and shooting angle, which is helpful in obtaining images with uniform illumination. Although carefully constructed imaging schemes can greatly reduce the difficulty of classical algorithm design, they also increase the application cost. At the same time, under natural environments, it is often unrealistic to expect classical algorithms designed to completely eliminate the impact of scene changes on recognition results. In real complex natural environments, plant disease and pest detection faces many challenges, such as small differences between the lesion area and the background, low contrast, large variations in the scale of the lesion area and various types, and a lot of noise in the lesion

image. Also, there are a lot of disturbances when collecting plant disease and pest images under natural light conditions. At this time, traditional classical methods often appear helpless and it is difficult to achieve better detection results.

In recent years, with the successful application of deep learning models represented by convolutional neural networks (CNNs) in many fields of computer vision (CV), for example traffic detection, medical image recognition, scenario text detection, expression recognition, face recognition, etc., several plant disease detection and diagnosis methods based on deep learning have been applied in real agricultural practice. Some domestic and foreign companies have developed a variety of deep learning-based plant disease detection and diagnosis systems. Therefore, plant disease and pest detection methods based on deep learning not only have important academic research value but also have a very broad market application prospect (4).

Scope and Limitations:

In this thesis, I aim to detect and diagnose plant diseases through the analysis of leaf images. My goal is to identify the presence of disease at the earliest stage of its development, preventing it from spreading throughout the entire plant or affecting the fruit. At one stage of my research, specifically level three, I incorporate the presence of pests into my model to improve its accuracy. In future stages, such as level four, I plan to expand the scope of my research to include the diagnosis of fruit diseases.

Despite my ambitious goals, my work was limited by several constraints. I had only 50 days to study, build a model, and write the book. Another limitation was the small number of resources available. I initially started training with my PC's CPU but discovered it was very slow. I then tried using my PC's GPU, but after a week of no progress and a destroyed GPU, I realized that was not a good idea. Finally, I turned to Kaggle, the world's largest data science community with powerful tools and resources to help achieve data science goals. Kaggle provided me with only 30 hours of GPU (P100) usage every 7 days and an input limit of about 20GB for datasets. I also used Colab for software development. The biggest challenge in plant disease detection and diagnosis is the lack of data. We often merge small datasets to create a larger one. My internet efficiency is low; it takes me 7 hours to upload 1.4GB. These are the limitations that I faced while conducting my research.

Definition of plant disease and pests' detection and diagnosis problem

Definition of plant diseases and pests:

Plant diseases and pests are types of natural disaster that affect the normal growth of plants and even cause plant death during the whole growth process of plants from seed development to seedling and to seedling growth. In machine vision tasks, plant diseases and pests tend to be the concepts of human experience rather than a purely mathematical definition.

Definition of plant disease and pests' detection and diagnosis:

Compared to the specific classification, detection and segmentation tasks in computer vision, the requirements for plant diseases and pests' detection and diagnosis are very general. In fact, its requirements can be divided into three different levels: what, where and how. In the first stage, "what" corresponds to the classification task in computer vision. As shown in Fig. 1, the label of the category to which it belongs is given. The task in this stage can be called classification and only gives the category information of the image. In the second stage, "where" corresponds to the location task in computer vision, and the positioning of this stage is the rigorous sense of detection. This stage not only identifies which types of diseases and pests exist in the image, but also gives their specific locations. As shown in Fig. 1, the affected area of gray mold is marked with a rectangular box. In the third stage, "how" corresponds to the segmentation task in computer vision. As shown in Fig. 1, the lesions of gray mold are distinguished from the background pixel by pixel, and a series of information such as the length, area, and location of the lesions of gray mold can be further obtained, which can assist the higher-level severity level evaluation of plant diseases and pests.

Classification describes the image globally through feature expression, and then determines whether there is a certain kind of object in the image by means of classification operation; while object detection focuses on local description, that is, answering which object exists in which position in an image, so in addition to feature expression, object structure is the most obvious feature that object detection differs from object classification. That is, feature expression is the main research line of object classification, while structure learning is the research focus of object detection. Although the function requirements and objectives of the three stages of plant diseases and pests' detection are different, in fact, the three stages are mutually inclusive (interconnected) and can be converted. For example, the "where" in the second stage contains the process of "what" in the first stage, and the "how" in the third stage can finish the task of "where" in the second stage. Also, the "what" in the first stage can achieve the goal of the second and the third stages through some methods. Therefore, the problem in this thesis is collectively referred to as plant diseases and pests detection and diagnosis as conventions in the following text, and the terminology differentiates only when different network structures and functions are adopted.

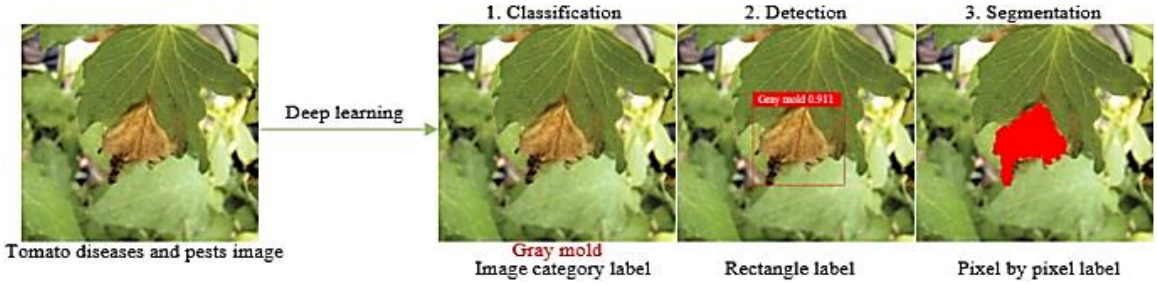

Figure 0.1 Definition of plant disease and pests detection and diagnosis problem

Comparison with traditional plant diseases and pests detection and diagnosis methods:

To better illustrate the characteristics of plant disease and pest detection and diagnosis methods based on deep learning, "a comparison is made with traditional plant disease and pest detection and diagnosis methods in four aspects: essence, method, required conditions and applicable scenarios. Detailed comparison results are shown in Table 6.1 (4).

Technology	Traditional image processing methods	Traditional image processing methods
Essence	Manual design features + classifiers (or rules)	Automatic learning of features from large amounts of data
Method	**Image segmentation method**: Threshold segmentation; Roberts, Prewitt, Sobel, Laplace and Kirsh edge detection; region segmentation **Feature extraction method**: SIFT, HOG, LBP, shape, color and texture feature extraction method **Classification method**: SVM, BP, Bayesian	CNN
Required conditions	Relatively harsh imaging environment requirements, high contrast between lesion and non-lesion areas, less noise	Adequate learning data and high-performance computing units
Applicable scenarios	It is often necessary to change the threshold or redesign the algorithm when imaging environment or plant diseases and pests class changes, which has poor recognition effect in real complex natural environment	It has ability to cope with certain real and complex natural environment changes

Table 0.1 Contrast between traditional image processing methods and deep learning methods

Overview of Machine Learning with a Focus on CNN and YOLO

In this section, we aim to establish a solid background on the machine learning concepts used in this project. We will start by defining machine learning and discussing the three main types: supervised, unsupervised, and semi-supervised learning. We will also cover artificial neural networks (ANNs) and delve into image recognition technology based on deep learning. This will include an introduction to deep learning theory and convolutional neural networks (CNNs), followed by a brief overview of classification, detection, and segmentation networks.

Machine Learning:

There are many definitions for machine learning. One definition state that Machine Learning is the science of programming computers so they can learn from data (5). Another more general definition states that Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed (6). A more engineering-oriented definition is: a computer program is said to learn from experience E with respect to some task T and some

performance measure P if its performance on T, as measured by P, improves with experience E (7). Another definition is: in other words, machine learning is defined as: The scientific study of algorithms and statistical models that computer systems use to perform a specific task without being explicitly programmed (8).

Machine learning has many applications, for example: analyzing images of products on a production line to automatically classify them, detecting tumors in brain scans, automatically classifying news articles, automatically flagging offensive comments on discussion forums, forecasting a company's revenue next year, detecting credit card fraud, recommending a product that a client may be interested in based on past purchases, building an intelligent bot for a game, and much more.

Supervised vs Unsupervised Learning vs Semi-supervised:

Machine Learning systems can be classified according to the amount and type of supervision they receive during training. The three main categories are supervised learning, unsupervised learning, and semi-supervised learning.

In supervised learning, the training set you feed to the algorithm includes the desired solutions, which are called labels. The two most typical supervised learning tasks are classification, where the system has to learn how to classify a particular input into one of a set of categories, and regression, where the task is to predict a target numeric value. Some important supervised learning algorithms include k-nearest neighbors, linear regression, logistic regression; support vector machines (SVMs), decision trees, random forests, and neural networks.

In unsupervised learning, the training data is unlabeled. Therefore, the system has to be able to learn patterns in the data without any kind of guidance. Some important unsupervised learning problems include clustering, anomaly detection, visualization, and dimensionality reduction. Some important algorithms include k-means clustering and DBSCAN.

Semi-supervised learning is a branch that combines a small amount of labeled data with a large amount of unlabeled data during training. It falls between unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data). Semi-supervised learning aims to alleviate the issue of having limited amounts of labeled data available for training. Unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy (9). Some important semi-supervised learning algorithms include Label Propagation, Label Spreading, and Self-Training (10).

Artificial neural networks (ANNs) are computing systems inspired by the biological neural networks that constitute animal brains (11)

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection,

like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives signals, processes them, and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs called an activation function. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold.

Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly after traversing the layers multiple times.

An ANN is trained by processing examples, each of which contains a known input and result. The network forms probability-weighted associations between the two, which are stored within the data structure of the net itself. The training is conducted by determining the difference between the processed output of the network and a target output. This difference is the error. The network then adjusts its weighted associations according to a learning rule and using this error value. Successive adjustments will cause the neural network to produce output that is increasingly similar to the target output. After a sufficient number of these adjustments, the training can be terminated based on certain criteria. This is a form of supervised learning (12).

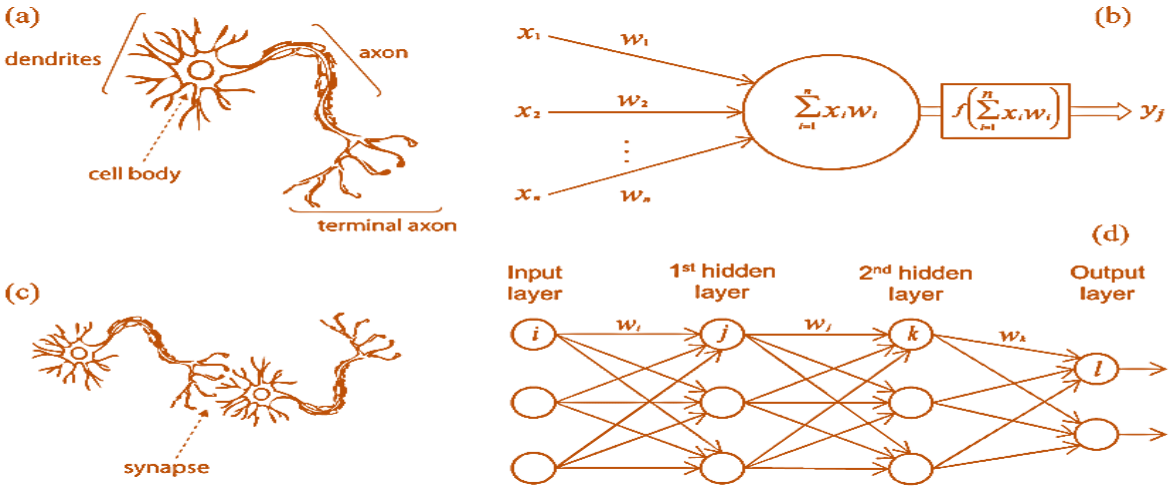

Figure 0.2 Artificial neural network inspired by the biological neural networks

Image recognition technology based on deep learning:

Compared with other image recognition methods, image recognition technology based on deep learning does not require the extraction of specific features. Instead, it can find appropriate features through iterative learning. This allows it to acquire global and contextual features of images and results in strong robustness and higher recognition accuracy (4).

Deep learning theory:

The concept of Deep Learning (DL) originated from a paper published in Science by Hinton et al. in 2006 (13). The basic idea of deep learning is to use neural networks for data analysis and feature learning. Data features are extracted by multiple hidden layers, with each hidden layer acting as a set of perceptrons. The perceptron is used to extract low-level features, which are then combined to obtain abstract high-level features. This can significantly alleviate the problem of local minima.

Deep learning overcomes the disadvantage of traditional algorithms relying on "artificially designed features" and has attracted more and more researchers' attention. It has now been successfully applied in computer vision, pattern recognition, speech recognition, natural language processing, and recommendation systems (14).

Traditional image classification and recognition methods that rely on manually designed features can only extract underlying features and have difficulty extracting deep and complex image feature information (15). Deep learning can solve this bottleneck by directly conducting unsupervised learning from the original image to obtain multi-level image feature information such as low-level features, intermediate features, and high-level semantic features.

Traditional plant disease and pest detection algorithms mainly adopt the image recognition method of manually designed features, which is difficult and depends on experience and luck. It cannot automatically learn and extract features from the original image. On the contrary, deep learning can automatically learn features from large data without manual manipulation. The model is composed of multiple layers, which have good autonomous learning ability and feature expression ability. It can automatically extract image features for image classification and recognition. Therefore, deep learning can play a great role in the field of plant disease and pest image recognition.

At present, deep learning methods have developed many well-known deep neural network models, including deep belief network (DBN), deep Boltzmann machine (DBM), stack de-noising auto-encoder (SDAE), and deep convolutional neural network (CNN) (16). In the area of image recognition, the use of these deep neural network models to realize automated feature extraction from high-dimensional feature space offers significant advantages over traditional manual design feature extraction methods. In addition, as the number of training samples grows and computational power increases, the characterization power of deep neural networks is being further improved.

Nowadays, the boom of deep learning is sweeping both industry and academia, with the performance of deep neural network models significantly ahead of traditional models. In recent years, the most popular deep learning framework is the deep convolutional neural network (4).

Convolutional neural network:

Convolutional Neural Networks (CNN) has a complex network structure and can perform convolution operations. As shown in Fig. 3, the CNN model is composed of an input layer, convolution layer, pooling layer, fully connected layer, and output layer. In one model, the convolution layer and the pooling layer alternate several times. When the neurons of the convolution layer are connected to the neurons of the pooling layer, no full connection is required.

CNN is a popular model in the field of deep learning due to its huge model capacity and complex information processing capabilities. This enables CNN to excel in image recognition tasks. The successes of CNN in computer vision have also boosted the growing popularity of deep learning.

In the convolution layer, a convolution kernel is first defined. This kernel can be considered as a local receptive field, which is one of the greatest advantages of CNN. When processing data information, the kernel slides over the feature map to extract part of the feature information. After feature extraction by the convolution layer, the neurons are input into the pooling layer for further feature extraction. Commonly used pooling methods include calculating the mean, maximum, and random values of all values in the local receptive field.

After passing through several convolution and pooling layers, the data enters the fully connected layer where all neurons are fully connected with those in the upper layer. Finally, data in the fully connected layer can be classified using methods such as softmax before being transmitted to the output layer for result output (17) (18) (4).

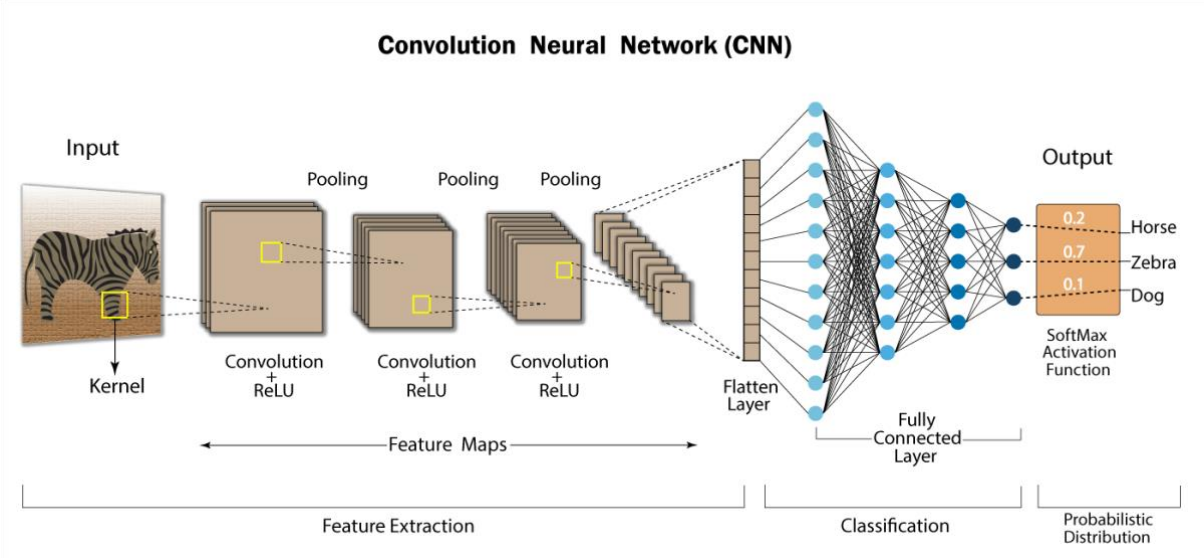

Figure 6.0.3 Convolutional neural network

Plant diseases and pests detection and diagnosis methods based On deep learning:

This sub-section provides a summary overview of plant disease and pest detection and diagnosis methods based on deep learning. Since the goal achieved is consistent with computer vision tasks, these methods can be seen as an application of relevant classical networks in the field of agriculture. As shown in Fig. 4, the network can be further subdivided into classification, detection, and segmentation networks according to their different structures. This sub-section is also subdivided into several different sub-methods according to the processing characteristics of each type of method.

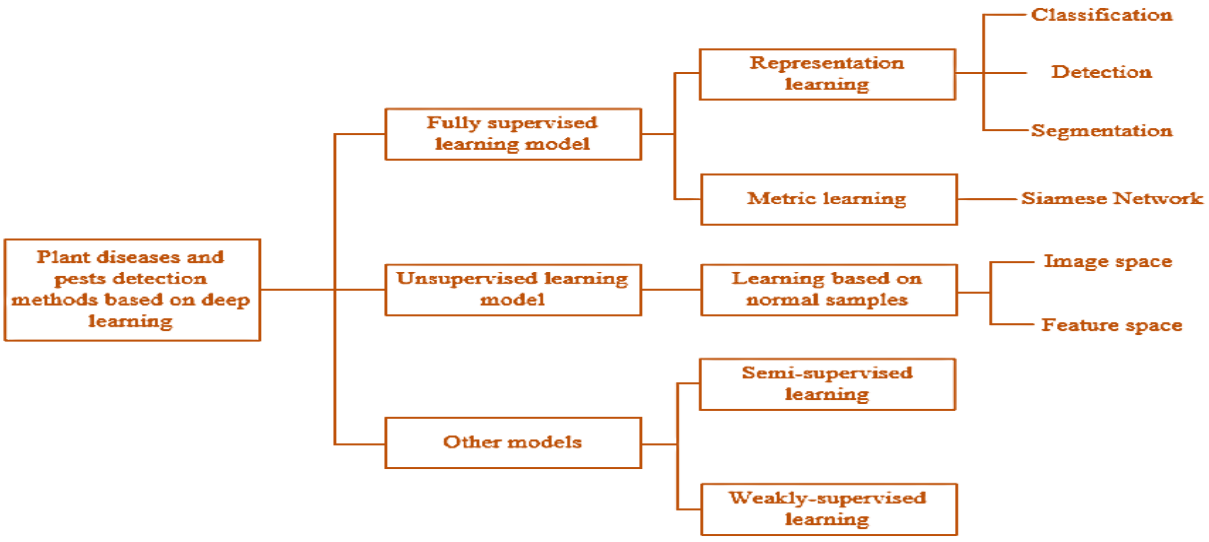

Figure 6.0.4 Framework of plant diseases and pests detection and diagnosis methods based on deep learning

Classification network:

In a real natural environment, the great differences in shape, size, texture, color, background, layout, and imaging illumination of plant diseases and pests make recognition a difficult task. Due to the strong feature extraction capability of CNNs, the adoption of CNN-based classification networks has become the most commonly used pattern in plant disease and pest classification. Generally, the feature extraction part of a CNN classification network consists of cascaded convolution layers and pooling layers, followed by a fully connected layer (or average pooling layer) and a softmax structure for classification. Existing plant disease and pest classification networks mostly use mature network structures in computer vision, including AlexNet (19), GoogleLeNet (20), VGGNet (21), ResNet (22), Inception V4 (23), DenseNets (24), MobileNet (25), and SqueezeNet (26).There are also some studies that have designed network structures based on practical problems . By inputting a test image into the classification network, the network analyzes the input image and returns a label that classifies the image. According to the differences in tasks achieved by the classification network method, it can be subdivided into three subcategories: using the network as a

feature extractor, using the network for direct classification, and using the network for lesion location.

- **Using network as feature extractor:**

  In early studies on plant disease and pest classification methods based on deep learning, many researchers took advantage of the powerful feature extraction capability of CNNs. These methods were combined with traditional classifiers (27). First, images are input into a pre-trained CNN network to obtain image characterization features. The acquired features are then input into a conventional machine learning classifier (e.g., SVM) for classification. This approach was used by Yalcin et al. (28)

- **Using network for classification directly:**

  Directly using a classification network to classify lesions is the earliest common means of applying CNN in plant disease and pest detection. According to the characteristics of existing research work, it can be further subdivided into original image classification, classification after locating the Region of Interest (ROI), and multi-category classification.

  1- Original image classification: This involves directly inputting the collected complete plant disease and pest image into the network for learning and training. Thenmozhi et al. (29) proposed an effective deep CNN model and used transfer learning to fine-tune the pre-training model.

  2- Classification after locating the Region of Interest ROI: For the whole image acquired, we should focus on whether there is a lesion in a fixed area. We often obtain the Region of Interest (ROI) in advance and then input the ROI into the network to judge the category of diseases and pests. This approach was used by Nagasubramanian et al. (30).

  3- Multi-category classification: When the number of plant disease and pest classes to be classified exceeds 2 classes, the conventional plant disease and pest classification network is the same as the original image classification method. That is, the output nodes of the network are the number of plant disease and pest classes + 1 (including normal class). However, multi-category classification methods often use a basic network to classify lesions and normal samples, and then share feature extraction parts on the same network to modify or increase the classification branches of lesion categories. This approach is equivalent to preparing a pre-training weight parameter for subsequent multi-objective plant disease and pest classification networks, which is obtained by binary training between normal samples and plant disease and pest samples. Picon et al. (31) used this approach.

Using a network for lesion location:

Generally, a classification network can only complete the classification of image labels. However, it can also achieve the location of lesions and pixel-by-pixel classification by combining different techniques and methods. According to the different means used, it can be further divided into three forms: sliding window, heatmap, and multi-task learning network.

I. Sliding window: This is the simplest and most intuitive method to achieve coarse lesion location. The image in the sliding window is input into the classification network for plant disease and pest detection by redundantly sliding on the original image through a smaller size window. Finally, all sliding windows are connected to obtain the results of the lesion location.

II. Heatmap: This is an image that reflects the importance of each region in the image; the darker the color represents the more important it is. In the field of plant disease and pest detection and diagnosis, the darker the color in the heatmap represents the greater probability that it is a lesion.

III. Multi-task learning network: If a pure classification network does not add any other skills, it could only realize image-level classification. Therefore, to accurately locate plant diseases and pests, the designed network should often add an extra branch, with both branches sharing the results of feature extraction. In this way, the network generally has both classification and segmentation output for plant diseases and pests, forming a multi-task learning network that takes into account the characteristics of both networks. For segmentation network branches, each pixel in the image can be used as a training sample to train the network. Therefore, a multi-task learning network not only uses segmentation branches to output specific segmentation results of lesions but also greatly reduces sample requirements for classification networks.

In summary, methods based on classification networks are widely used in practice and many scholars have carried out application research on plant disease and pest classification. At the same time, different sub-methods have their own advantages and disadvantages as shown in Table 2 (4).

Method	Advantages	Disadvantages
Using network as feature extractor	Obtaining effective lesion features	Relying on other classifiers for final classification results
Original image classification	Classic in structure, it is also the basis of other classification network sub-methods and can refer to many existing networks	Lesions need to account for a certain proportion in the image, otherwise their characteristics are easily pooled out, and generally only one class of lesion is allowed in an image
Classification after locating ROI	Obtaining ROI information of the lesions	Additional methods are needed to obtain ROI
Multi-category classification	Solving sample imbalance to some extent	Secondary training is needed
Sliding window	Get rough localization of lesions in images	Sliding window size requires accurate selection, and can only get rough position, slow speed of traversal and sliding
Heatmap	Generate more accurate lesion areas	Accurate lesions location depends on network classification performance
Multi-task learning network	Combining other networks to obtain exact location and category of lesions simultaneously, and reducing the number of training samples required	The network structure is relatively complex, and a pixel-by-pixel label is required when adding segmentation branches

Table 6.0.2: Comparison of advantages and disadvantages of each sub-method of classification network

Detection network:

Object positioning is one of the most basic tasks in the field of computer vision and is also closely related to plant disease and pest detection. Its purpose is to obtain accurate location and category information for objects. Currently, there are many deep learning-based object detection methods. Generally speaking, plant disease and pest detection networks based on deep learning can be divided into two types: two-stage networks represented by Faster R-CNN and one-stage networks represented by SSD and YOLO. The main difference between the two types of networks is that the two-stage network first generates a candidate box (proposal) that may contain lesions and then further executes the object detection process. In contrast, the one-stage network directly uses the features extracted in the network to predict the location and class of lesions.

Plant disease and pest detection and diagnosis based on two-stage networks:

The basic process of a two-stage detection network (Faster R-CNN) is to first obtain the feature map of the input image through the backbone network, then calculate the anchor box confidence using RPN and get the proposal. Then, input the feature map of the proposal area after ROIpooling into the network, fine-tune the initial detection results, and finally get the location and classification results of lesions. Therefore, according to the characteristics of plant disease and pest detection, common methods often improve on the backbone structure or its feature map, anchor ratio, ROIpooling, and loss function.

- Plant disease and pest detection and diagnosis based on one-stage networks:

The one-stage object detection algorithm has eliminated the region proposal stage but directly adds a detection head to the backbone network for classification and regression, thus greatly improving the inference speed of the detection network. The single-stage detection network is divided into two types: SSD and YOLO. Both use the whole image as input to the network and directly return the position of bounding boxes and their categories at the output layer.

Compared with traditional convolutional neural networks, SSD selects VGG16 as its trunk network and adds a feature pyramid network to obtain features from different layers and make predictions.

"You Only Look Once" (YOLO) considers the detection task as a regression problem and uses global information to directly predict the bounding box and category of an object to achieve end-to-end detection with a single CNN network. YOLO can achieve global optimization while satisfying higher accuracy with greatly improved detection speed (4). YOLO has many versions starting from 1 (32) up to 8 which we used. For more information go to (33) (34) . To give an overview: YOLO takes an entire image in a single shot and predicts bounding box coordinates and class probabilities for these boxes. The biggest advantage of YOLO is its lightning-fast execution speed. YOLO also understands generalized object representation. The network does not look at the entire image but only at parts of images that have a higher chance (probabilities) of containing an object.

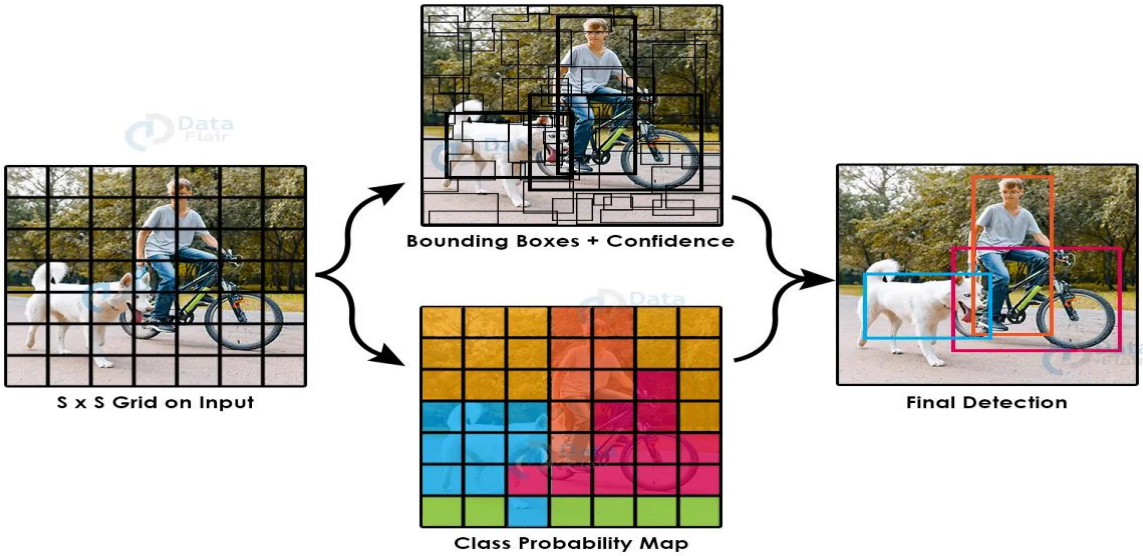

Figure 6.0.5 Yolo Object detection

YOLO first takes input images then divides them into grids (see figure 5). For each box, it checks whether an object is present or not. It calculates X-

Coordinate of bounding box center inside cell (0;1 with respect to grid cell size), Y-Coordinate of bounding box center inside cell (0;1 with respect to grid cell size), W-Bounding box width [0;1] with respect to image, H-Bounding box height [0;1] with respect to image, C-Bounding box confidence (object in box) (see figure 6). PC represents whether an object is present in grid or not (Probability), X,Y,H,W represents bounding box of an object, C1,C2,C3 represent class of objects such as car, bike, bus etc. If PC = 0 then C1, C2, C3 = 0. If PC = 1 it tries to find the center of the available object. If it finds multiple bounding boxes it uses Intersection Over Union (IOU) and Non-max suppression algorithms to find the best bounding box. Once we get the best bounding box for which object exists it uses CNN to classify the images (1).

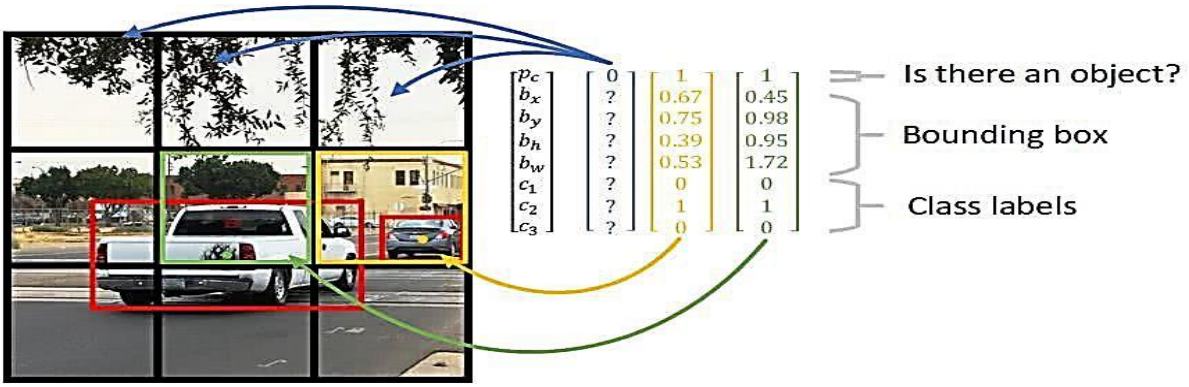

Figure 6.0.6: How does the algorithm predict bounding boxes

In summary, in the field of plant disease and pest detection that emphasizes detection accuracy at this stage, more models based on two-stage networks are used. In contrast, in the field of plant disease and pest detection that pursues detection speed, more models based on one-stage networks are used.

Can a detection network replace a classification network?

The task of a detection network is to solve the location problem of plant diseases and pests. The task of a classification network is to judge the class of plant diseases and pests. Visually, the hidden information of a detection network includes category information; that is, the category information of plant diseases and pests that need to be located needs to be known beforehand, and corresponding annotation information should be given in advance to judge the location of plant diseases and pests. From this point of view, the detection network seems to include the steps of the classification network; that is, the detection network can answer “what kind of plant diseases and pests are in what place.” However, there is a misconception in which “what kind of plant diseases and pests” is given a priori; that is, what is labeled during training is not necessarily the real result. In the case of strong model differentiation; that is, when the detection network can give accurate results, it can answer “what kind of plant diseases and pests are in what place” to a certain extent. However, in

the real world, in many cases it cannot uniquely reflect the uniqueness of plant disease and pest categories but can only answer “what kind of plant diseases and pests may be in what place.” Then the involvement of a classification network is necessary. Thus, a detection network cannot replace a classification network.

- Segmentation network:

A segmentation network converts the plant disease and pest detection and diagnosis task to semantic and even instance segmentation of lesions and normal areas. It not only finely divides the lesion area but also obtains the location, category, and corresponding geometric properties (including length, width, area, outline, center, etc.). It can be roughly divided into Fully Convolutional Networks (FCN) (35) and Mask R-CNN (36).

- FCN:

A Full Convolutional Neural Network (FCN) is the basis of image semantic segmentation. Currently, almost all semantic segmentation models are based on FCN. FCN first extracts and encodes the features of the input image using convolution, then gradually restores the feature image to the size of the input image by deconvolution or up-sampling. Based on differences in FCN network structure, plant disease and pest segmentation methods can be divided into conventional FCN, U-net, and SegNet.

I. **Conventional FCN.**
II. **U-net:** U-net is not only a classical FCN structure but also a typical encoder-decoder structure. It is characterized by introducing a layer-hopping connection that fuses the feature map in the encoding stage with that in the decoding stage, which is beneficial for recovering segmentation details.
III. **SegNet:** It is also a classical encoder-decoder structure. Its feature is that the up-sampling operation in the decoder takes advantage of the index of the largest pooling operation in the encoder.

- Mask R-CNN:

Mask R-CNN is one of the most commonly used image instance segmentation methods at present. It can be considered as a multi-task learning method based on detection and segmentation networks. When multiple lesions of the same type have adhesion or overlap, instance segmentation can separate individual lesions and further count their number. However, semantic segmentation often treats multiple lesions of the same type as a whole (4).

YOLOv8 also introduces segmentation service and classification.

Compared with classification and detection network methods, the segmentation method has advantages in obtaining lesion information. However, like detection networks, it requires a lot of annotation data, and its annotation information is pixel by pixel, which often takes a lot of effort and cost.

- Evaluation indices used in object detection evaluation of our work:

  Common evaluation indices include Precision, Recall, mean Average Precision (mAP) and the harmonic Mean F1 score based on Precision and Recall.

  Precision and Recall is defined as:

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}} . \mathbf{100\%}$$

Equation 0.1

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}} . \mathbf{100\%}$$

Equation 0.2

In Formula 6.1 and Formula 6.2, TP (True Positive) is true-positive, predicted to be 1 and actually 1, indicating the number of lesions correctly identified by the algorithm. FP (False Positive) is false-positive, predicted to be 1 and actually 0, indicating the number of lesions incorrectly identified by the algorithm. FN (False Negative) is false-negative, predicted to be 0 and actually 1, indicating the number of unrecognized lesions.

Detection accuracy is usually assessed using mAP. The average accuracy of each category in the dataset needs to be calculated first:

$$\boldsymbol{P_{Average}} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{N(class)}} \boldsymbol{Precision(j).Recall(j).100\%}$$

Equation 0.3

In the above-mentioned formula, N(class) represents the number of all categories, Precision(j) and Recall(j) represents the precision and recall of class j respectively. Average accuracy for each category is defined as mAP:

$$\boldsymbol{mAP} = \frac{\boldsymbol{P_{average}}}{\boldsymbol{N(class)}}$$

Equation 0.4

The greater the value of mAP, the higher the recognition accuracy of the algorithm; conversely, the lower the accuracy of the algorithm. F1 score is also introduced to measure the accuracy of the model. F1 score takes into account both the accuracy and recall of the model. The formula is

$$\boldsymbol{F1 = \frac{2Precision\,.\,Recall}{Precision + Recall}\,.\,100\%}$$

Equation 0.5

Frames per second (FPS) is used to evaluate the recognition speed. The more frames per second, the faster the algorithm recognition speeds; conversely, the slower the algorithm recognition speed (4).

Methodology

To simplify my work and organize my efforts, I divided my work into three levels. At level one, I worked with the PlantVillage open-source dataset which contains 38 classes of [plant, disease] combinations, including a healthy class. My work focused only on image classification methods. I tried four models of increasing complexity, starting with a scratch model, then Vgg16, Vgg19, and Resnet50V2 which gave the most accuracy. At level two, I went deeper with classes, which included 88 classes of [plant, disease] combinations and also healthy classes collected from a large open-source dataset and more complex model which is ResNet152V2 which is 152 layers and has nearly 53 million parameters. I also started version one of object detection, building two models, xlarge and nano, using the Yolov8 algorithm on the PlanDocPlus dataset built by me, also object detection on an apple dataset collected from PlantApple_2020, Plant-pathology-2021-fgvc8_4 competition which contains 8,837 images of four classes, At level three, At the time of writing this book, I am still working on it.

Level 1:

At this level, I am working with a classification network. I am using the network for classification directly and the sub-category I use is 'original image classification'. I am following the approach outlined in the paper titled 'Deep learning models for plant diseases detection and diagnosis' (3). My dataset consists of 38 classes and is an open-source dataset on Kaggle that was collected in the lab. I trained a total of 5 models and achieved the highest accuracy of 99.812% using the ResNet50v2 model.

The papers:

- **Deep learning models for disease detection and diagnosis. (3)**
- **Using Deep learning for Image-Based plant Disease Detection. (2)**
- **Plant Disease Detection Using Deep Learning. (37)**
- **Plant disease identification from individual lesions and spots using deep learning (38)**

Objectives:

After reading the research paper, my goal is to train a classification network model that can accurately distinguish between healthy and diseased plant leaf images. I aim to achieve a validation accuracy of at least 99% for a dataset consisting of 38 classes.

Data Collection:

For my data collection, I utilized a version of the PlantVillage dataset (39) (40), an open access repository of plant health images introduced by Hughes et al (41)to facilitate the development of mobile disease diagnostics. This dataset contains 54,303 images of healthy and unhealthy plant leaves, categorized into 38 classes based on species and disease. I primarily used the New Plant Diseases Dataset (42), which was derived from the original dataset (PlantVillage) using offline augmentation techniques. This dataset comprises approximately 87K RGB images of healthy and diseased crop leaves, organized into 38 distinct classes. The dataset is split into an 80/20 ratio for training and validation purposes. Additionally, a set of 33 test images was later generated for use in prediction tasks.

class	Plant common name	Disease common name	Image
C0	Apple	Apple scab	2520
C1	Apple	Black rot	2484
C2	Apple	Cedar_apple_rust	2200
C3	Apple	healthy	2510
C4	Blueberry	healthy	2270
C5	Cherry(&sour	healthy	2282
C6	Cherry(&sour	Powdery_mildew	2104
C7	Corn(maize)	Cercospora||Gray_leaf_spot	2052
C8	Corn(maize)	Common_rust_	2384
C9	Corn(maize)	healthy	2324
C10	Corn(maize)	Northern_Leaf_Blight	2385
C11	Grape	black rot	2360
C12	Grape	Esca_(Black_Measles)	2400
C13	Grape	healthy	2115
C14	Grape	Leaf_blight_ (Isariopsis_Leaf_Spot)	2152
C15	Orange	Haunglongbing_ (Citrus_greening)	2513
C16	Peach	Bacterial_spot	2297
C_17	Peach	healthy	2160
C_18	Pepper,bell	Bacterial_spot	2391
C_19	Pepper,bell	healthy	2485

Table 6.0.3: New Plant Diseases Dataset

Figure 0.7: New Plant Diseases Dataset

Training:

To develop my models, I utilized the TensorFlow (43) and Keras (44) platforms for coding and implementation. I also leveraged pre-trained models, which I downloaded for transfer learning purposes. All of my work was conducted on the Kaggle platform (45), which provided a convenient and powerful environment for experimentation and development.

Unfortunately, I didn't anticipate that I would be writing this book and accidentally overwrote some of the models. However, I still have two models remaining: VGG19 and ResNet50V2. I will provide their details and mention any information I have from previous versions.

In all models, I used the Adam optimizer (46), which stands for Adaptive Moment Estimation. It is an optimization technique for gradient descent that is highly efficient when working with large problems involving a lot of data or parameters. It requires less memory and is computationally efficient. Intuitively, it can be seen as a combination of the 'gradient descent with momentum' algorithm and the 'RMSProp' algorithm. For the loss function, I used Categorical Crossentropy, which is commonly used in classification problems where the classes are mutually exclusive. It measures the dissimilarity between the true class labels and predicted probabilities.

Scratch model:

I started working by building a CNN model from scratch, as shown in Figure 7. I began training the model using the previously mentioned training conditions, with a batch size of 32 and an image scheme of 'RGB'. I trained the

model for 50 epochs and achieved an accuracy of approximately 92.5%, which did not meet my objective. So, I decided to move forward.

```
#After looking on data starting build small model from scratch

model = keras.Sequential()

model.add(keras.layers.Conv2D(32,(3,3),activation="relu",padding="same",input_shape=(256,256,3)))
model.add(keras.layers.Conv2D(32,(3,3),activation="relu",padding="same"))
model.add(keras.layers.MaxPooling2D(3,3))

model.add(keras.layers.Conv2D(64,(3,3),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(64,(3,3),activation="relu",padding="same"))
model.add(keras.layers.MaxPooling2D(3,3))

model.add(keras.layers.Conv2D(128,(3,3),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(128,(3,3),activation="relu",padding="same"))
model.add(keras.layers.MaxPooling2D(3,3))

model.add(keras.layers.Conv2D(256,(3,3),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(256,(3,3),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(512,(5,5),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(512,(5,5),activation="relu",padding="same"))
model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(1568,activation="relu"))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(38,activation="softmax"))
```

Figure 0.8: Scratch Model

- ## Fine-tuning VGG16:

  At this stage, I fine-tuned the VGG16 model by downloading a pre-trained version that had been trained on the ImageNet dataset. I fixed the weights of all layers except the last one and began training the last layer on my 38-class dataset using the same conditions as the scratch model. This resulted in an accuracy of approximately 97%, which did not meet my objective. As a result, I decided to move forward and explore other options.

- ## Full fine-tuning VGG16:

  At this stage, I performed full fine-tuning on the VGG16 model by downloading a pre-trained version that had been trained on the ImageNet dataset. I began training all layers of the model on my 38-class dataset using the same conditions as the scratch model. This resulted in an accuracy of approximately 98.5%, which did not meet my objective. As a result, I decided to move forward and explore other options

- ## Full fine-tuning VGG19:

  At this stage, I performed full fine-tuning on the VGG19 model by downloading a pre-trained version that had been trained on the ImageNet dataset. I began training all layers of the model on my 38-class dataset using the same conditions as the scratch model with 55 epochs. I also used

another feature of TensorFlow which includes ModelCheckpoint, EarlyStopping, and CSVLogger. The most important one is ModelCheckpoint. It is a callback in TensorFlow that can be used with Keras to save a model or its weights at some frequency. It can be used to serialize your network to disk each time there is an improvement during training. An "improvement" can be defined as either a decrease in loss or an increase in accuracy. My model saved every time validation accuracy improved. This resulted in an accuracy of 99.06% at epoch 45, which is very close to my objective but I decided to move forward and explore other options. The model was saved at "checkpoint7". The last epoch's model is at "potatoes7.h5". You can also find the notebook, logs, and test results.

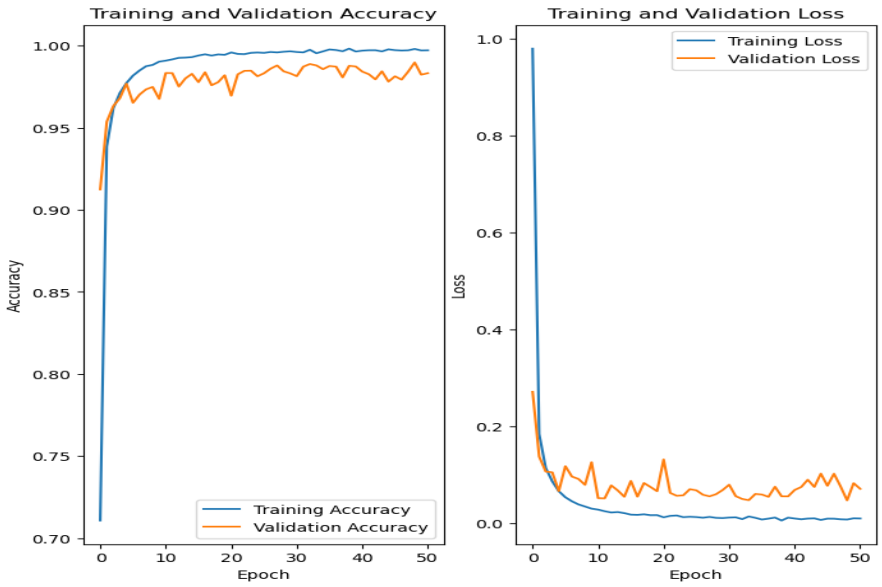

Figure 6.0.9 Vgg19

Full fine-tuningResNet50V2:

At this stage, I performed full fine-tuning on the ResNetV2 model by downloading a pre-trained version that had been trained on the ImageNet dataset. I began training all layers of the model on my 38-class dataset using the same conditions as the Vgg19 model with 55 epochs and another version with 95 epochs. However, there was no improvement in accuracy between the two versions. As a result, I achieved an accuracy of 99.812% at epoch 46, which surpassed my objective. So, I stopped and started thinking about the obstacles of all models at this level. The model was saved at "checkpoint1". The last epoch's model is at "potatoes7.h5". You can also find the notebook, logs, and test results.

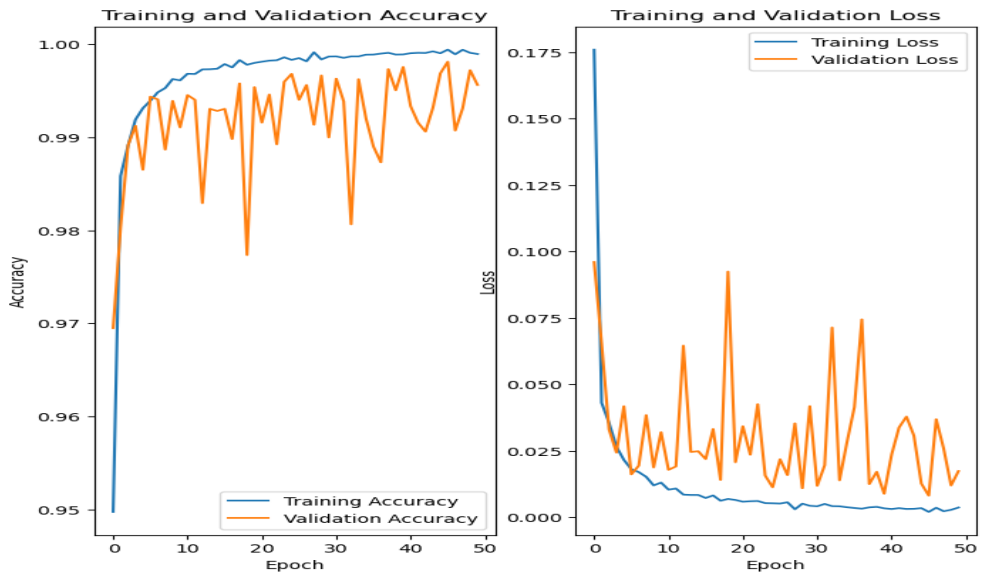

Figure 6.0.10: ResNet50V2

Upgrading to the next level:

One limitation of our current approach is that our photographic material consists solely of images taken in experimental laboratory setups, rather than in real-world cultivation fields. This lack of real-world data can reduce the accuracy of our model when applied to actual field conditions. Additionally, all of our images come from a single source or dataset, which limits the variety of images available for training our model. This can further reduce accuracy when our model is used by a diverse group of users with varying image quality and conditions. Furthermore, the number of classes in our current model is relatively small compared to the wide variety of diseases and pests that can affect crops. To address these issues, we are considering upgrading our model to improve its accuracy and effectiveness.

Level 2:

At this stage, I am working on improving the classification network to address the limitations of the previous version at level one. I am utilizing a dataset called “Plant Disease Classification Merged Dataset” (47), which was compiled from 14 different datasets and includes both laboratory and field images. The dataset comprises 88 classes. To date, I have developed four models: ResNet50V2, ResNet152V2, InceptionResNetV2, and EfficientNetB5. The InceptionResNetV2 model achieved an accuracy of nearly 96.5%. Additionally, I have begun my initial work on an object detection network, but I am encountering difficulty in finding a suitable dataset. Nevertheless, with the assistance of two distinct datasets - PlantDocPlus (an expanded version of PlantDoc) and “Apple Leaf Disease Object Detection dataset” (48) (a private dataset from paper (49)) - I was able to construct two models. Each model has two versions: one for real-time operation and another with more parameters for backend processing. These models were developed using Yolov8, which was the most recent version available at the time.

The papers:

- Crop Infection Detection using YOLO. (1)
- MGA-YOLO: A lightweight one-stage network for apple leaf disease detection (49)
- Real-Time Flying Object Detection with YOLOv8. (33)

Objectives:

Since I was unable to locate the dataset used in the study (Plant Village with annotation) (1) or any other suitable dataset for building a robust model, my goal is to develop an object detection network that can effectively interact with the data. The network should be capable of identifying the presence of a disease at a specific location and suggesting its potential type. To ensure accuracy, a classification network can be employed. Using the apple dataset, I aim to construct a reliable predictor and indicator model.

Data Collection:

Classification Networks:

This dataset was created as part of a study project on plant disease classification. The requirements for the dataset included having a large number of images, with at least one healthy and one diseased image for each plant, covering the most common diseases, annotated images, laboratory and field images, and featuring important staple food and highest global production plant species. To meet these requirements, a new dataset was compiled by combining images from 14 different existing datasets. The properties and imported images of all included datasets are listed in the table below along with their proper references. This new dataset contains both laboratory and field images. Images of non-food plants, singular condition classes, watermarked images, and classes with less than 50 examples were removed. The final dataset used for training within the project contains 88 classes with over 76,000 images and an overall size of 17.6GB. However, it is not possible to completely avoid bias among the classes since different shooting conditions were used for some images (e.g., classes with mainly laboratory images or different soil in the background).

Dataset	properties	Merged images
Plant disease 65 (1)	65 classes, 62,600 files , contains plantVillage images and additional plants, photos of single plant leaves in front of plain backgrounds	all images beside images of nonfood plants and solitary classes
PlantDoc (2)	2598 images in 27 classes , mostly field images , taken from various angles , lightning condition and with various backgrounds , some laboratory images	all beside solitary classes
Coffee plant disease (3)	1000 images, 3 classes, field images of coffee plants	all
Wheat leaf dataset (4)	407 images in 3 classes , real field wheat images from the Holeta wheat farm in Ethiopia , sorted with the assistance of plant pathologist	all
Chili plant disease (5)	500 images in 5 classes of chili plants , field images of whole plant or plant parts	all
Images of soybean leaves (6)	6410 images in 3 classes , field images of soybean plants captured with smartphones and drones in different heights and different times of day	all
Rice leaf diseases (7)	120 pictures in 3 classes , single rice leaf in front of plain white background	all
Rice leafs (8)	3355 files in 4 classes , single rice leaf in front of plain back ground	only Hispa disease
Cucumber plant diseases dataset (9)	691 images in 2 classes ill and healthy , cucumber field images of plant leaves	all
Plant disease expert (10)	single plant leaves in front of plain backgrounds	added 1829 images of 6 classes of tea leaves and 11328 images of grape black rot
Leaf disease (combination) (11)	field images	added 2596 images in 5 classes for cassava
PDDB (12)	image database of plant dis ease symptoms with images from a multitude of plants ,labelled by phytopathologists	added 1 rice leaf blast class , 5 classes of coffee , 5 classes of sugarcane , 4 of cassava , 5 of corn , 7 of soy and 3 of wheat
Sugarcane disease (13)	299 field images in 3 classes of sugarcane leaves	all
Sugarcane leaf disease (14)	224 images 3 classes , field images from sugarcane farms	all

Table 0.4: Plant Disease Classification Merged Dataset

Object Detection:

Apple Leaf Disease Object Detection dataset (yolo_dataset):

The dataset consists of 8,839 images and their corresponding annotation labels. It also includes train.txt, test.txt, and valid.txt files which contain the names of the images that can be used for training, testing, and validation. Additionally, there is a datasets.yaml file that contains the relative directory for building the train, test, and validation directories. I started by developing a software program to read these text files and build the train directory which contains an images directory and its associated images labels. I did the same for the test and validation directories. The images in the dataset contain only four classes: 'Healthy', 'Black rot', 'Scab', and 'Rust'.

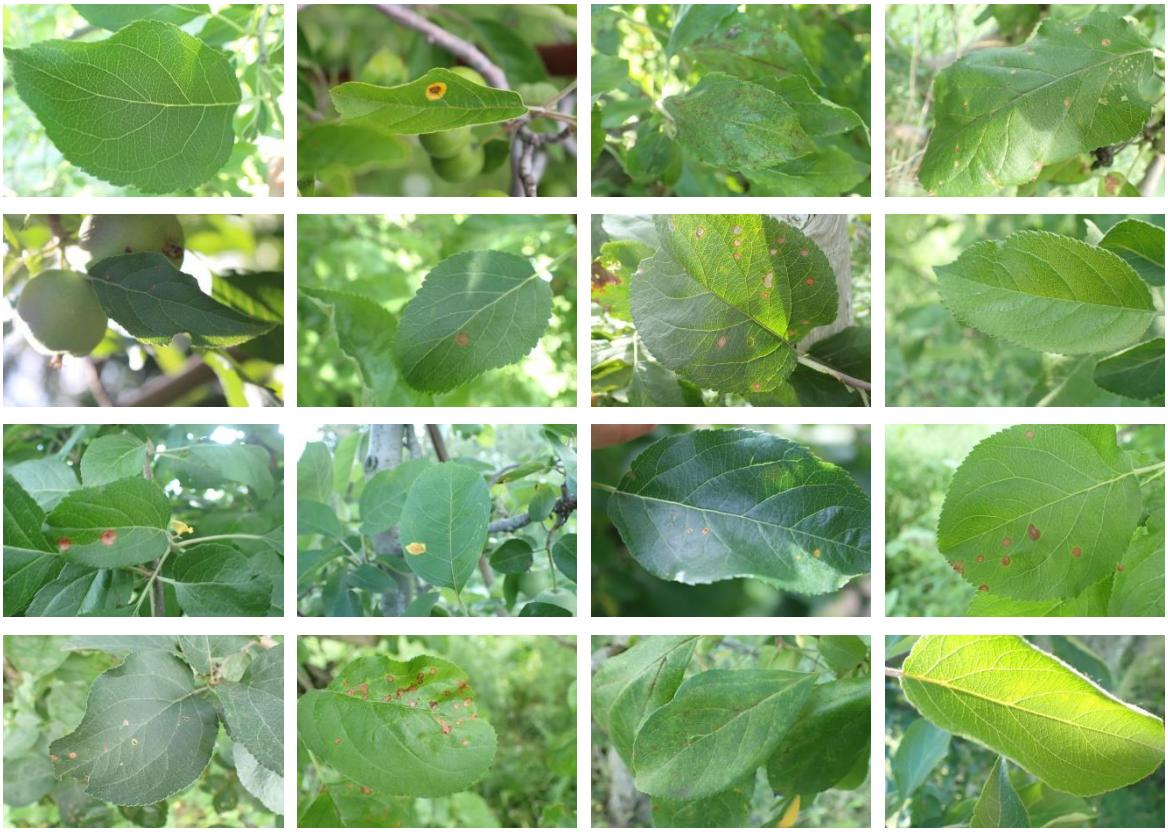

Figure 6.0.11 Apple Leaf Disease Object Detection dataset

PlantDocPlus Dataset:

The PlantDoc dataset is a comprehensive collection of 2598 images, spanning 27 distinct classes. These images, primarily captured in the field, showcase a diverse range of angles, lighting conditions, and backgrounds. Additionally, the dataset includes laboratory images for further analysis. To improve the classification of healthy and diseased tomato and potato plants, I incorporated various small datasets from Roboflow (64). Roboflow is a cutting-edge platform that empowers engineers to build and deploy computer vision models with ease. With over 250,000 engineers utilizing its tools to create datasets, train models, and deploy to production, Roboflow offers a range of utilities to seamlessly integrate computer vision into any application. The final dataset boasts 3812 annotated images, with 3721 designated for training and 541 for validation.

Figure 6.0.12 PantDocPlus

Training:

Classification Model:

In my work, I have continued to use the same approach as in the previous level. I have chosen to use the Adam optimizer and the Categorical Crossentropy loss function. To download, develop, and train my model, I have utilized TensorFlow and Keras, all of which was done on the Kaggle platform.

Full fine-tuningResNet50V2:

Initially, I utilized a model that achieved an accuracy of 77% at the beginning and 86% at the end, which was not satisfactory. As a result, I decided to experiment with another model that had more parameters and layers. Despite the fact that these models were trained on 1000 classes and demonstrated good performance, the task of detecting and diagnosing plant diseases and pests proved to be challenging. This was due to the subtle differences between each class, which were difficult to discern even with the naked eye

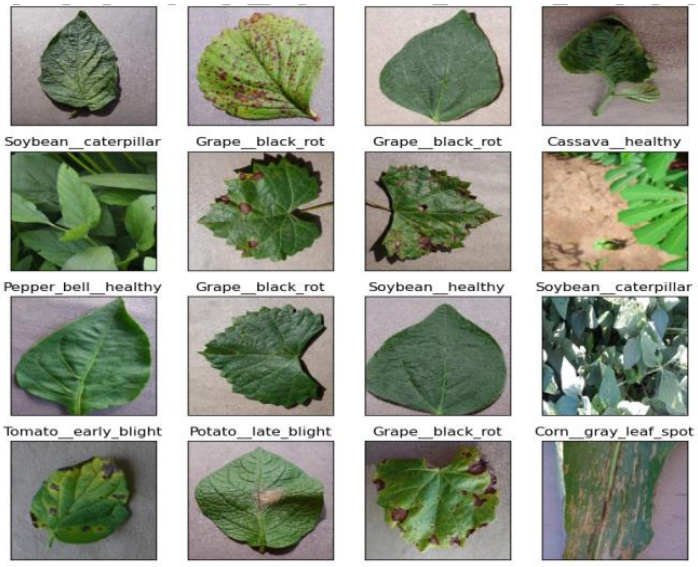

Figure 6.0.13: ResNet152v2

Full fine-tuning ResNet152V2:

I followed the same approach as with the ResNet152V2 model, limiting the number of training epochs per session to 10. After two sessions, I observed that the output for each epoch did not change significantly. The accuracy started at 86.53% in the first epoch and increased to 95.91% by the tenth epoch. It peaked at 96.646% in epoch 17 before plateauing until epoch 20. Given that the training accuracy was 98.45%, I realized that it was necessary to stop here to avoid overfitting. Notably, the VGG19 model, which has 143.7 million trainable parameters, achieved an increase in accuracy from .71% to 99% over just 11 epochs. As a result, I decided to stop here. (It is worth mentioning that I tried many other modes, but due to technical reasons, the output was distorted.)

I began by working with a dataset of 79,087 images, divided into 88 directories, each representing a class defined by the name of the plant and its disease or healthy state. The images were not pre-processed, so I performed some basic augmentation using the ImageDataGenerator library from TensorFlow, with parameters such as rescale=1/255, rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.2, horizontal_flip=True and vertical_flip=True. I also split the images into a test set comprising 5% of the total images and a training set comprising the remaining 95%, which was further split into an 80/20% training/validation split. The input image size was 256x256 pixels with a batch size of 32 and an RGB color scheme. Due to the large number of parameters (nearly 53 million), I could only complete 17 epochs in a 9-hour session. I then reloaded the last saved weights and trained for another 17 epochs but saw no progress, which led me to try a more complex model. The difference between the training accuracy (0.9795) and validation accuracy (0.9609) was small. I

implemented early stopping when the validation loss did not improve for 10 epochs, so training stopped at epoch 26. All these factors led me to try a more complex model. The model was saved at “checkpoint4”. The last epoch’s model is at “potatoes7.h5”. You can also find the notebook, logs, and test results.

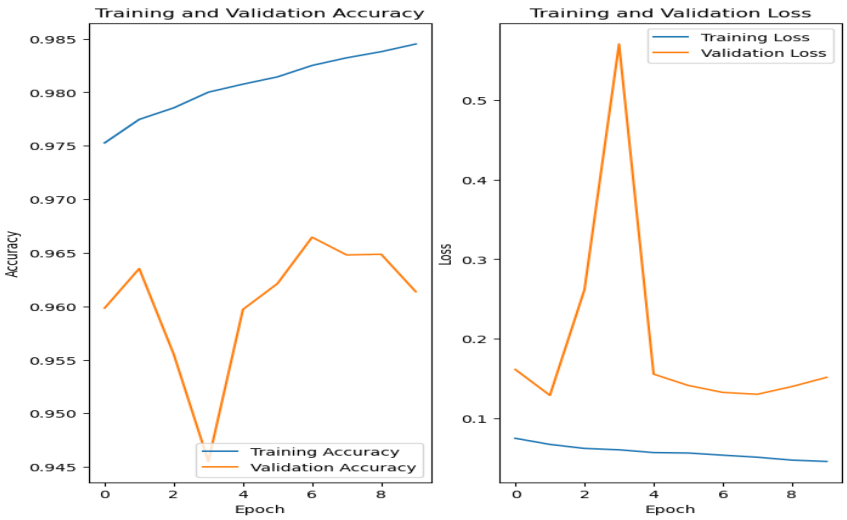

Figure 6.0.14 InceptionResNetV2

Object Detection:

To further develop my work, I relied on the PyTorch (65) platform and the Ultralytics (66) package, which includes the YOLOv8 class. All of this was executed on the Kaggle platform.

Yolov8_Apple:

To train the model, we had several options: we could build a YOLOv8 model with our own specifications and train it from scratch, add pre-trained weights to it, or use a pre-trained model with its weights. For our purposes, we chose to use a pre-trained model with its weights and fine-tune it to our dataset. YOLOv8 is a set of models for classification, detection, and segmentation, available in nano, small, medium, large, and xlarge versions. The larger the model, the more parameters it has, making it more complex to train but also better at prediction. The nano version has fewer parameters and is designed for real-time operation. I built a nano version for use with live cameras or live video and a medium version for prediction and processing large videos on the backend with a GPU.

Nano_Version:

I trained the nano version of the model for 98 epochs, using an input size of 640x640 pixels and a batch size of 16. The Adam optimizer was used for training. To evaluate the performance of the object detection model, I used the mAP metric, which gave me a mAP50 score of 0.964 and a mAP50-95 score of 0.945. The best model saves at best.pt.

Note: mAP (mean Average Precision) is a common metric used to evaluate the performance of object detection models. mAP50 refers to the mAP calculated at an Intersection over Union (IoU) threshold of 0.50. This means that for a predicted bounding box to be considered a true positive, it must have an IoU of at least 0.50 with the ground truth bounding box. An mAP50 of 0.964 means that the model has a high level of accuracy in detecting objects and correctly assigning them to their respective classes when evaluated at an IoU threshold of 0.50

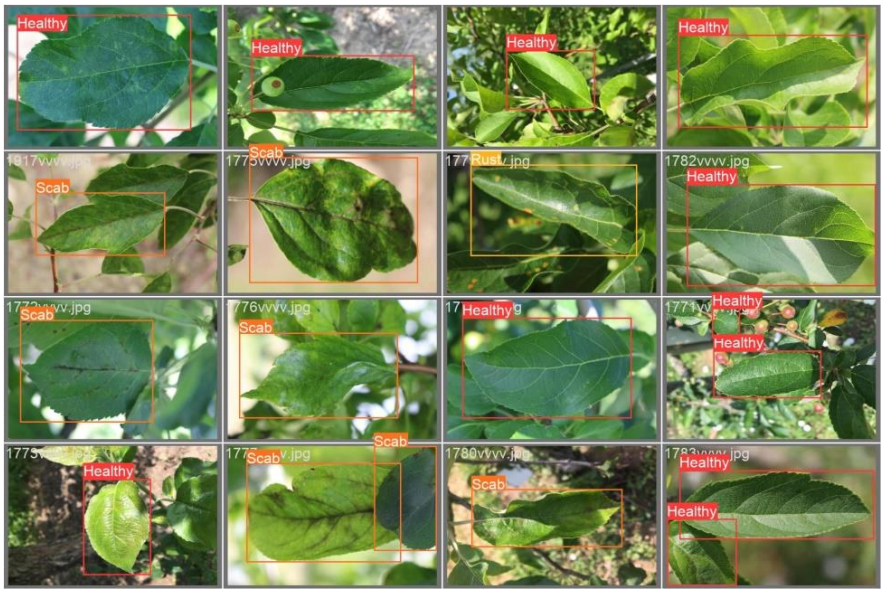

Figure 6.0.15: Nano Version. Top is labeled images and down is predication

Medium _Version:

I trained the medium version of the model for 93 epochs, as the time per epoch was longer for the medium version. I chose the medium version because it had only 4 classes. The input size was 640x640 pixels and the batch size was 16, with the Adam optimizer used for training. To evaluate the performance of the object detection model, I used the mAP metric, which gave me a mAP50 score of 0.963 and a mAP50-95 score of 0.948. Although the difference in accuracy was not large, the medium version still performed better than the small version and was able to detect more objects in the same image. The best model saves at best.pt

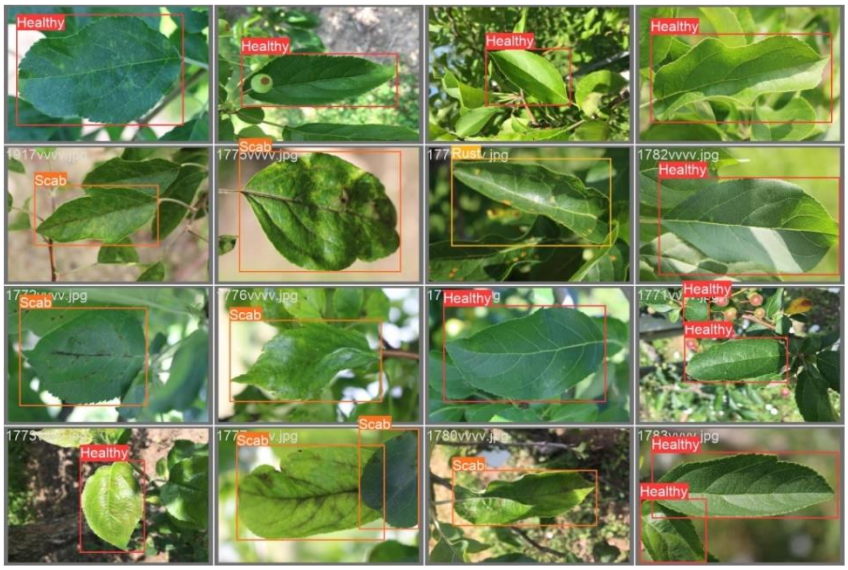

Figure 6.0.16: medium version- Top is labeled images and down is predication

PlantDocPlus:

I followed the same method that I used with the apple dataset, with the only differences being the number of epochs and the use of an xlarge model instead of a medium model, due to the number of classes 27.

Nano Version:

I trained this model for 190 epochs, which resulted in a mAP50 score of 0.552 and a mAP50-95 score of 0.418. The best model saves at best.pt

Figure 6.0.17 On the Top is the annotated images and down is the prediction images

Xlarge_Version:

I trained this model for 295 epochs by reloading the last saved weights, which resulted in a mAP50 score of 0.435 and a mAP50-95 score of 0.331. The best model was saved as 'best.pt'.

Note: that when reloading the last saved weights and resuming training, the mAP accuracy did not continue from where it left off but instead started from an intermediate value. I am not sure why this happened. (I think that the learning rate or other hyperparameters were changed when resuming training, which could affect the mAP accuracy. So, reload is not working well)

Figure 0.18: On the Top is the annotated images and down is the prediction images

Upgrading to the next level:

At this time, my work does not include pests and I believe that the number of images for each class is small, as is the number of classes. We have not yet worked with segmentation, so there is room to go deeper.

Level 3:

At the time of writing this book, I am still working on it.

Due to the significant amount of time required to write the book, I have limited time remaining to complete level three. As such, I would like to note that if I am unable to finish my work on this level, future work on these topics will involve three networks: a classification network(V3)., an object detection network (V2), and a segmentation network (V1). The classification network begins by gathering data from resource (65) - the Plant Disease Classification Merged Dataset - and Dataset (66) from paper (67). Resource (65) refers to a GitHub site that compiles all open-source datasets available online. The Plant Disease Classification Merged Dataset is a dataset I utilized in level two, while Dataset (66) is a private dataset from paper (67), available on Google Drive, containing 97 classes. I anticipate that the final result will include 150,000 images and 150 classes of

plant diseases and pests. The object detection network is constructed using Yolov8 and is based on the Plant Village dataset.

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (MS) per inference step (CPU)	Time (MS) per inference (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-
ConvNeXtTiny	109.42	81.3%	-	28.6M	-	-	-
ConvNeXtSmall	192.29	82.3%	-	50.2M	-	-	-
ConvNeXtBase	338.58	85.3%	-	88.5M	-	-	-
ConvNeXtLarge	755.07	86.3%	-	197.7M	-	-	-

Table 6.0.5 Available models

Conclusion and future work:

In this study, specialized deep learning models were developed based on specific convolutional neural network architectures to identify plant diseases through the analysis of simple leaf images of healthy or diseased plants. The models were trained using openly available photographs from datasets, taken under both laboratory and real-world conditions in cultivation fields. The work is divided into two levels. Level one involves a classification network trained on the PlantVillage dataset, which contains nearly 54,000 images. Five models were trained, with the best performance achieved by ResNet50V2 at 99.812%. Level two comprises three main models and data: classification (V2), data collected from 14 open-source datasets containing 88 classes of plant disease combination. three models were trained, with the best performance achieved by InceptionResNetV2 at 96.646%. Object detection was performed on an open-source apple dataset with 8,839 annotated images. Nano and medium models of Yolov8 were trained, achieving mAP50 scores of 0.964 and 0.963 and mAP50-95 scores of 0.945 and 0.948, respectively. Object detection was also performed on PlantDocPlus (an extended version of PlantDoc), which contains 3,812 annotated images. Nano and Xlarge models of Yolov8 were trained, achieving mAP50 scores of 0.552 and 0.435 and mAP50-95 scores of 0.418 and 0.331, respectively.

The high level of performance achieved by the classification network demonstrates the suitability of convolutional neural networks for the automated detection and diagnosis of plant diseases through the analysis of simple leaf images. Additionally, the results indicate the importance of including real-condition images (captured in cultivation fields) in the training data to enhance model robustness. This suggests that object detection should be used on PlantDocPlus to indicate the presence of disease, while classification should be used to accurately identify the type of disease. Apple object detection should be used for direct prediction.

Furthermore, the low computational power required by the trained model to classify a given image makes it feasible to integrate into mobile applications for use on mobile devices such as smartphones or drones and other autonomous agricultural vehicles for real-time monitoring and dynamic disease detection on large-scale open-field cultivations. In the former case, a farmer in a remote location could receive an early warning about a potential threat to their cultivation, while an agronomist could have a valuable advisory tool at their disposal. A future possibility could be the development of an automated pesticide prescription system that would require confirmation by the automated disease diagnosis system before allowing farmers to purchase appropriate pesticides. This would significantly reduce the uncontrolled acquisition of pesticides that leads to their overuse and misuse, with consequent catastrophic effects on the environment.

Despite the high success rate of the developed system, there are several reasons why a better model could be achieved. For instance, the dataset I used contains only 88

classes and is sourced from various countries, which may not be particularly relevant to Egyptian farmers. If I were to merge two datasets, I would be unable to do so because I am not familiar with the properties of the diseases and cannot differentiate between similar diseases with the naked eye. If I were to make an error while merging the data, it would go unnoticed, and the model would not indicate this mistake. It is akin to a man harming himself. When it comes to object detection annotation of the dataset, my hands are tied. Although some leaves have only one disease, others have complex diseases. How can I draw a square around something that I cannot identify? Even though the data contains field images, they are not from Egypt. I have read numerous papers and have seen many countries construct their own datasets, making them available exclusively to their communities. The model becomes more robust when trained under similar prediction conditions.

As such, I propose the following: (1) anyone wishing to undertake a module on plant disease and pest detection and diagnosis in their final year project at Fayoum University must commence where I left off in order to develop a more robust and advanced prediction model that is updated periodically. (2) Collaboration between the artificial intelligence group and final year students in the College of Agriculture is essential, with at least 12 students from the College of Agriculture working on data. If you would like my assistance in your future endeavors, please lend me a hand now. (3) The Department of Computers at the Faculty of Engineering at Fayoum University must establish a database containing data collected and reviewed during the project, subject to evaluation and supervised by a specialized doctor for ease of access and organization.

Conclusion

Smart farming systems have the potential to revolutionize the way we produce food by leveraging advanced technologies such as the internet of things (IoT), artificial intelligence (AI), and big data analytics. These systems can help farmers optimize their use of resources, improve crop yields, reduce waste, and increase profitability.

Through the use of sensors and other IoT devices, smart farming systems can collect real-time data on various environmental factors such as soil moisture, temperature, and humidity. This data can then be analyzed using AI algorithms to provide insights on crop health, disease outbreaks, and other important metrics. By leveraging this information, farmers can make more informed decisions about when to plant, water, and harvest their crops.

Furthermore, smart farming systems can help reduce the environmental impact of agriculture by reducing the use of pesticides and fertilizers, optimizing water usage, and reducing waste. By adopting these systems, farmers can also improve their overall sustainability and reduce their carbon footprint.

In conclusion, smart farming systems have the potential to transform the agriculture industry by improving efficiency, sustainability, and profitability. As technology continues to advance, we can expect to see even more innovative solutions that will further enhance the capabilities of these systems.

Future Plans

1. **Data Collection and Analysis:**

   The system should be able to collect data from the sensors and analyze it to provide insights into the farming process. This includes analyzing soil moisture levels, temperature, and humidity to determine the optimal conditions for crop growth.

2. **Crop Management:**

   The system should be able to manage crop growth, including monitoring growth rates, identifying pests and diseases, and providing recommendations for fertilizer and pesticide applications.

3. **Livestock Monitoring:**

   The system should be able to monitor the health and well-being of livestock, including tracking their movements, feeding schedules, and health indicators.

4. **Mobile App:**

   The system should include a mobile application that allows farmers to monitor and control their farming operations from anywhere.

5. **Internal Monitor and Control System:**

   The system should include internal control unit that allows farmers monitor and control the farm internally from the farm itself.

References

1. **[Online] https://www.kaggle.com/datasets/fabinahian/plant-disease-65-classes.**

2. [Online] https://doi.org/10.1145/3371158.3371196.

3. [Online] https://www.kaggle.com/datasets/coffeedisease/coffee-plant-disease.

4. [Online] https://www.kaggle.com/datasets/olyadgetch/wheat-leaf-dataset.

5. [Online] https://www.kaggle.com/datasets/dhenyd/chili-plant-disease.

6. [Online] https://data.mendeley.com/datasets/bycbh73438/1.

7. [Online] https://www.kaggle.com/datasets/vbookshelf/rice-leaf-diseases.

8. [Online] https://www.kaggle.com/datasets/shayanriyaz/riceleafs.

9. [Online] https://www.kaggle.com/datasets/kareem3egm/cucumber-plant-diseases-dataset.

10. [Online] https://www.kaggle.com/datasets/sadmansakibmahi/plant-disease-expert.

11. [Online] https://www.kaggle.com/datasets/asheniranga/leaf-disease-dataset-combination.

12. ***Identifying multiple plant diseases using digital image processing.*** **L. V. S. T. T. BARBEDO, J. G. A.; KOENIGKAN. s.l. : Biosystems Engineering, 2016, Vol. 147. Biosystems Engineering.**

13. [Online] https://www.kaggle.com/datasets/prabhakaransoundar/sugarcane-disease-dataset.

14. [Online] https://www.kaggle.com/datasets/pungliyavithika/sugarcaneleaf-disease-classification.

15. ***Crop Infection Detection using YOLO.*** **Satwik Ram Kodandaram, Kushal Honnappa. 2021, ResearchGate.**

16. ***UsingDeepLearningforImage-BasedPlantDiseaseDetection.*** **Sharada P.Mohanty, DavidP.Hughes, MarcelSalathé. 2016, frontiers, p. 10.3389/fpls.2016.01419.**

17. ***Deep learning models for plant disease detection and diagnosis.*** **Ferentinos, Konstantinos P. 2018, Elsevier, pp. Computers and Electronics in Agriculture 145 (2018) 311–318.**

18. ***Plant diseases and pests detection based.*** **Jun Liu and Xuewei Wang. 2021, Plant Methods, pp. 10.1186/s13007-021-00722-9.**

19. Géron, Aurélien. ***Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow".*** **s.l. : O'Reilly, 2019.**

20. ***Some Studies in Machine Learning Using the Game of Checkers.*** **Samuel, Arthur L. 1959, IBM.**

21. Mitchell, Tom M. ***Machine Learning.*** **s.l. : McGraw-Hill , 1997.**

22. Mahesh, Batta. ***Machine Learning Algorithms -A Review.*** **s.l. : ResearchGate, 2019.**

23. wikipedia. Weak supervision. ***wikipedia.*** **[Online] [Cited: 6 24, 2023.] https://en.wikipedia.org/wiki/Weak_supervision.**

24. Brownlee, Jason. Semi-Supervised Learning With Label Spreading. ***machinelearningmastery.*** **[Online] 1 2021, 4. [Cited: 6 24, 2023.] https://machinelearningmastery.com/semi-supervised-learning-with-label-spreading/.**

25. Brahme, Anders. ***Comprehensive Biomedical Physics.*** **s.l. : Elsevier , 2014.**

26. wikipedia. Artificial neural network. ***wikipedia.*** **[Online] [Cited: 6 24, 2023.] https://en.wikipedia.org/wiki/Artificial_neural_network#cite_note-2.**

27. ***Reducing the Dimensionality of Data with Neural Networks.*** **Hinton and Salakhutdinov. 2006, Science, pp. 313(5786):504–7.**

28. ***A survey of deep neural network architectures and their applications.*** **al, Liu et. 2017 , Neurocomputing, pp. 2017;234:11–26.**

29. ***Deep Learning Methods for Vision.*** **presented, Rob Fergus. s.l. : 2012 Conference on Computer Vision and Pattern Recognition (CVPR), 2012. 2012 Conference on Computer Vision and Pattern Recognition (CVPR).**

30. ***Representation learning: a review and new perspectives.*** **al., Bengio et. 2013 , IEEE Trans Pattern Anal Mach Intell, pp. 2013;35(8):1798–828.**

31. ***Ask the Locals: Multi-Way Local Pooling for Image Recognition.*** **Y-Lan Boureau, Nicolas Le Roux, Francis Bach, Jean Ponce, and Yann LeCun. Barcelona, Spain : ICCV), 2011. International Conference on Computer Vision (ICCV). pp. 2651-2658 .**

32. ***Stochastic Pooling for Regularization of Deep Convolutional Neural Networks.*** **Matthew D. Zeiler and Rob Fergus. s.l. : arXiv, 2013, arXiv.**

33. ***ImageNet Classification with Deep Convolutional Neural Networks.*** **Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012, research highlights , p. DOI:10.1145/3065386.**

34. ***Going Deeper with Convolutions.*** **Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich . 2015, Cornell University.**

35. ***Very deep convolutional networks for.*** **Karen Simonyan and Andrew Zisserman . s.l. : arxiv, 2014. 1409.1556.**

36. ***Aggregated Residual Transformations for Deep Neural Networks.*** **Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He. s.l. : arxiv, 2017. 1611.05431.**

37. ***Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.*** **Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. s.l. : arxiv., 2016. 1602.07261.**

38. ***Densely Connected Convolutional Networks.*** **Gao Huang, Zhuang Liu, Laurens van der Maaten, and others . s.l. : IEEE, 2017. 8099726.**

39. ***MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.*** **Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. s.l. : arxiv, 2017. 1704.04861.**

40. ***SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.*** **Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer. s.l. : arxiv, 2016. 1602.07360.**

41. ***Recent Studies of Image and Soft Computing Techniques for Plant Disease Recognition and Classification.*** **H. Sabrol, Satish Kumar. 2015. 10.5120/ijca2015905982.**

42. ***Plant classification using convolutional neural networks.*** **Hulya Yalcin; Salar Razavi. s.l. : IEEE, 2016. 7577698.**

43. ***Crop pest classification based on deep convolutional neural network and transfer learning.*** **K. Thenmozhi, U. Srinivasulu Reddy. s.l. : Computers and Electronics in Agriculture, 2019. S0168169919310695.**

44. ***Plant disease identification using explainable 3D deep learning on hyperspectral images.*** **Koushik Nagasubramanian , Sarah Jones , Asheesh K Singh , Soumik Sarkar , Arti Singh , Baskar Ganapathysubramanian . s.l. : pubmed, 2019. 10.1186/s13007-019-0479-8.**

45. ***Crop conditional Convolutional Neural Networks for massive multi-crop plant disease classification over cell phone acquired images taken on real field conditions.*** **A. Picón, Maximiliam Seitz, J. Echazarra. s.l. : semanticscholar, 2019. 208643496.**

46. ***You Only Look Once: Unified, Real-Time Object Detection.*** **Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. s.l. : arxiv, 2016. 1506.02640.**

47. ***Real-Time Flying Object Detection with YOLOv8.*** **Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. s.l. : arXiv, 2023. 2305.09972.**

48. glenn-jocher. ultralytics. ***github.*** **[Online] [Cited: 6 25, 2023.] https://github.com/ultralytics/ultralytics.**

49. ***"Fully Convolutional Networks for Semantic Segmentation.*** **Jonathan Long, Evan Shelhamer, Trevor Darrell. s.l. : arXiv, 2015. 1411.4038.**

50. ***Mask R-CNN.*** **Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. s.l. : arxiv, 2017. 1703.06870.**

51. ***Plant Disease Detection using Deep Learning.*** **Murk Chohan, Adil Khan, Rozina Chohan, Saif Hassan Katpar, Muhammad Saleem Mahar. 1, s.l. : International Journal of Recent Technology and Engineering, 2020, Vol. 9. 2277-3878.**

52. ***Plant disease identification from individual lesions.*** **Barbedo, Jayme Garcia Arnal. s.l. : ScienceDirect, 2019. 13083-886,.**

53. ARUN PANDIAN J,. Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network. ***mendeley.*** **[Online] [Cited: 6 26, 2023.] https://data.mendeley.com/datasets/tywbtsjrjv/1.**

54. spMohanty. PlantVillage-Dataset. ***github.*** **[Online] [Cited: 6 26, 2023.] https://github.com/spMohanty/PlantVillage-Dataset.**

55. ***An open access repository of images on plant health to enable the development of mobile disease diagnostics.*** **David. P. Hughes, Marcel Salathe. s.l. : arXiv, 2016. 1511.08060.**

56. vipoooool. New Plant Diseases Dataset. ***kaggle.*** **[Online] [Cited: 6 26, 2023.] https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset.**

57. ***tensorflow.*** **[Online] https://www.tensorflow.org/.**

58. ***keras.*** **[Online] https://keras.io/api/applications/.**

59. ***kaggle.*** **[Online] https://www.kaggle.com/.**

60. ***Adam: A Method for Stochastic Optimization.*** **Diederik P. Kingma, Jimmy Ba. s.l. : arxiv., 2017 . 1412.6980.**

61. ***MGA-YOLO: A lightweight one-stage network for apple leaf disease detection.*** **Yiwen Wang, Yaojun Wang and Jingbo Zhao. s.l. : frontiersin, 2022. 2022.927424.**

62. [Online] https://roboflow.com/.

63. [Online] https://pytorch.org/.

64. [Online] https://ultralytics.com/.

65. xm194. Plant_disease_recognition. ***github.*** **[Online] [Cited: 6 30, 2023.] https://github.com/xml94/Plant_disease_recognition.**

66. ***Google Drive.*** **[Online] https://drive.google.com/file/d/1HhtA939IwSjrN2XKRyeTgMQnTaY4zniA/view.**

67. ***Machine learning techniques for plant disease detection: an evaluation with a customized dataset.*** **Amatullah Fatwimah Humairaa Mahomodally, Geerish Suddul, Sandhya Armoogum. 2252-8776, s.l. : International Journal of Informatics and Communication Technology, 2023, Vol. 12. 10.11591/ijict.v12i2.pp127-139.**

68. ALINE DOBROVSKY. Plant Disease Classification Merged Dataset. ***kaggle.*** **[Online] [Cited: 6 27, 2023.] https://www.kaggle.com/datasets/alinedobrovsky/plant-disease-classification-merged-dataset.**

69. LIAOFANYISHI. Apple Leaf Disease Object Detection dataset. ***kaggle.*** **[Online] [Cited: 6 27, 2023.]**

https://www.kaggle.com/datasets/df248b05bcf0246cc0b7add831501d83a30396900302f3e6d01ab293471150f4.

70. Future Electronics. [Online] Future Electronics. https://store.fut-electronics.com/products/.

71. ST. STM32 Cortex®-M4 MCUs and MPUs programming manual. *ST.* [Online] 3 2020. [Cited: 10 1, 2022.] https://www.st.com/.

72. ARM. Cortex-M4 Devices. *ARM.* [Online] 2011. [Cited: 10 1, 2022.] https://www.arm.com/.

73. ST. *STM32F446xC/E Data Sheet.* s.l. : https://www.st.com/, 2021.

74. —. STM32F446RE Reference manual. *ST.* [Online] 2 2021. [Cited: 10 1, 2022.] https://www.st.com/.

75. OmniVision. *OV7670 CMOS VGA (640x480) CameraChip DataSheet.* s.l. : OmniVision, 2006.

76. Bjørner, Dines. *Software Engineering 3: Domains, Requirements, and Software Design.* s.l. : Springer, 2006.

77. herif M. Yacoub, Hany H. Ammar, Sherif Yacoub, Hany Ammar. *Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems.* s.l. : Addison-Wesley Professional, 2003.

78. Nygard, Michael T. *Release It!: Design and Deploy Production-Ready Software.* s.l. : Pragmatic Bookshelf, 2007.

79. software., Lewin Edwards Lewin Edwards is an embedded engineer with over 15 years experience designing embedded systems hardware firmware and control. *Embedded System Design on a Shoestring: Achieving High Performance with a Limited Budget (Embedded Technology).* s.l. : Newnes, 2003.

80. Frank Vahid, Tony D. Givargis. *Embedded System Design: A Unified Hardware Software Introduction.* s.l. : Wiley, 2001.

81. ThingsBoard. ThingsBoard. *ThingsBoard.* [Online] ThingsBoard. ThingsBoard.io.