B.Sc. GRADUATION PROJECT
ELECTRONICS & COMMUNCATION

Al-Azhar University
Faculty of Engineering
Electronics & Communication Department

ADAS COLLISION AVOIDANCE

Supervised by:

Dr. Mohamed Yasin

Represented by:

Toqa Mohamed	Amina Mohamed
Eman Khaled	Doha Magdy
Esraa Taha	MennatAllah Mohamed
Esraa Ali	Sohaila Ibrahim

2024

B.Sc. GRADUATION PROJECT
ELECTRONICS & COMMUNCATION

Al-Azhar University
Faculty of Engineering
Electronics & Communication Department

ADAS COLLISION AVOIDANCE

Supervised by:

Dr. Mohamed Yasin

Represented by:

Toqa Mohamed	Amina Mohamed
Eman Khaled	Doha Magdy
Esraa Taha	MennatAllah Mohamed
Esraa Ali	Sohaila Ibrahim

2024

Acknowledgement

Our primary thanks go to Allah for the Success in the Completion this work.

Foremost, we would like to express our sincere gratitude to our supervisor, Dr. Mohamed Yasin, for the continuous support of our graduation project and research, and for his patience, motivation, and immense knowledge. We could not have imagined having a better advisor and mentor for our graduation project.

Our appreciation to all members of the electrical engineering department at Al-Azhar University.

Last, we would like to thank our family: our parents supported us spiritually throughout our life.

Abstract

Most road users are aware of the general rules and safety measures when using the road, but only the negligence of some road users causes accidents. The main cause of accidents and incidents is human error. We explain some common human behaviors that lead to accidents.

99% of road accidents are due to human error. It is an inconvenient fact that most drivers continue to make the same mistakes on the road without noticing their poor driving habits.

Many accidents occur due to distracted driving and human error. Partially automated driver assistance systems (ADAS) can help drivers avoid collisions and maintain control of the vehicle by issuing warning signals.

We will use vehicle-to-vehicle (V2V) communication. It is a promising technology for improving road safety by enabling vehicles to exchange information about their speed, position, and direction in real-time. In this project, we propose a V2V collision avoidance advanced driver assistance system (ADAS) that can reduce the risk of accidents by alerting drivers of potential collisions and assisting them in taking corrective actions.

We also help drivers to park easily by using a self-parking system and this help drivers who do not have high driving skills.

We will explain the wireless communication between the drive subsystem and the vehicle, the hardware components in each subsystem, the function of each subsystem, and the data exchanged between them.

Overall, this graduation project demonstrates the effectiveness of V2V collision avoidance ADAS systems and highlights their potential for improving road safety and reducing accidents.

Contents

List of figures

List of tables

INTRODUCTION

0.1 Road accidents:

Road safety becomes a major public health concern when the statistics show that more than 3,000 people around the world succumb to death daily due to road traffic injury, Also, road crashes lead to the global economic losses as estimated in road traffic injury costs of US$518 billion per year. The huge economic losses are an economic burden for developing countries. It is reflected that the road crash costs are estimated to be US$ 100 billion in developing countries which is twice the annual amount of development aid to such countries. Over 130,000 fatalities and nearly 500,000 people were permanently disabled due to road crashes over past decades. The economic losses due to the road crashes are considerably high, costing approximately US$2,500 million per year (about US$0.3 million per hour), or 3.4% of the Gross National Product (GNP) [1].

The collection and use of accurate and comprehensive data related to road accidents is very important to road safety management. The road accident data are necessary not only for statistical analysis in setting priority targets but also for in-depth study in identifying the contributory factors to have a better understanding of the chain-of-events. Having the inconsistencies in the aims of the police and the road safety engineers, the data analysis and its interpretation usually does not result in proper countermeasures. Sometimes a lack of proper knowledge of crash and proper training of the police officers in charge of systematic data collection procedures from a crash scene adds to the diverging nature of the role of the police and the road safety professionals. These problems have become a burning issue for developing countries addressing road safety without completed crash data due to the negligence of the concerned authorities [1].

Road accidents are the most unwanted thing to happen to a road user, though they happen quite often. The most unfortunate thing is that we don't learn from

our mistakes on the road. Most of the road users are quite well aware of the general rules and safety measures while using roads but it is only the laxity on part of road users, which cause accidents and crashes. Main cause of accidents and crashes are due to human errors [1]. We are elaborating some of the common behavior of humans which results in accidents:

- Over Speeding

Most of the fatal accidents occur due to over speeding. If given a chance man is sure to achieve infinity in speed. But when we are sharing the road with other users, we will always remain behind some or another vehicle. Increasing speed multiplies the risk of accident and severity of injury during accidents. Faster vehicles are more prone to accident than the slower one and the severity of accident will also be more in case of faster severity of accident will also be more in case of faster vehicles. Higher the speed, greater the risk. At high speed the vehicle needs greater distance to stop i.e. braking distance. A slower vehicle comes to halt immediately while a faster one takes a long way to stop and also skids a long distance due to the law of notion. A vehicle moving on high speed will have greater impact during the crash and hence will cause more injuries. The ability to judge the forthcoming events also gets reduced while driving at faster speed which causes error in judgment and finally a crash [2].

- Distraction to Driver

Though distraction while driving could be minor but it can cause major accidents. Distractions could be outside or inside the vehicle. The major distraction nowadays is talking on a mobile phone while driving. Act of talking on the phone occupies a major portion of the brain and the smaller part handles the driving skills. This division of the brain hampers reaction time and ability of judgement. This is one of the reasons for crashes. One should not attend to telephone calls while driving. If the call is urgent, one should pull out beside the road and attend the call. Some of the distractions on road are:

I. Adjusting mirrors while driving.
II. Stereo/Radio in vehicle.

III. Animals on the road.
IV. Banners and billboards.

The driver should not be distracted due to these things and reduce speed to remain safe during diversions and other kinds of outside distractions [2].

How our project solves this problem?

we are looking to develop more efficient solutions using modern technology:

- V2V technology and several tools were used to assist the driver, reduce his stress, and protect people's lives, time, and environmental resources.
- This project is partially smart car to assist drivers, it includes some of advanced driver assistance system (ADAS) applications.

0.2 V2V Communications

0.2.1 What is V2V?

➢ Vehicle-to-vehicle (V2V) communication's ability to wirelessly exchange information about the speed and position of surrounding vehicles shows great promise in helping to avoid crashes, ease traffic congestion, and improve the environment [3].

➢ Using vehicle-to-vehicle (V2V) communication, a vehicle can detect the position and movement of other vehicles up to a quarter of a kilometer away [3].

➢ Vehicle can change information using Global Positioning System (GPS) and technology similar to Wi-Fi. If a V2V-equipped vehicle brakes suddenly, this event can be relayed back to a following V2V vehicle, which can then trigger an alert such as a flashing display or beeping warning to the following driver [3].

➢ V2V communication technology can increase the performance of vehicle safety systems and help save lives. There were an estimated 6.8 million police-reported crashes in 2019, resulting in 36,096 fatalities and an estimated 2.7 million people injured. Connected vehicle technologies will provide drivers with

the tools they need to anticipate potential crashes and significantly reduce the number of lives lost each year [3].

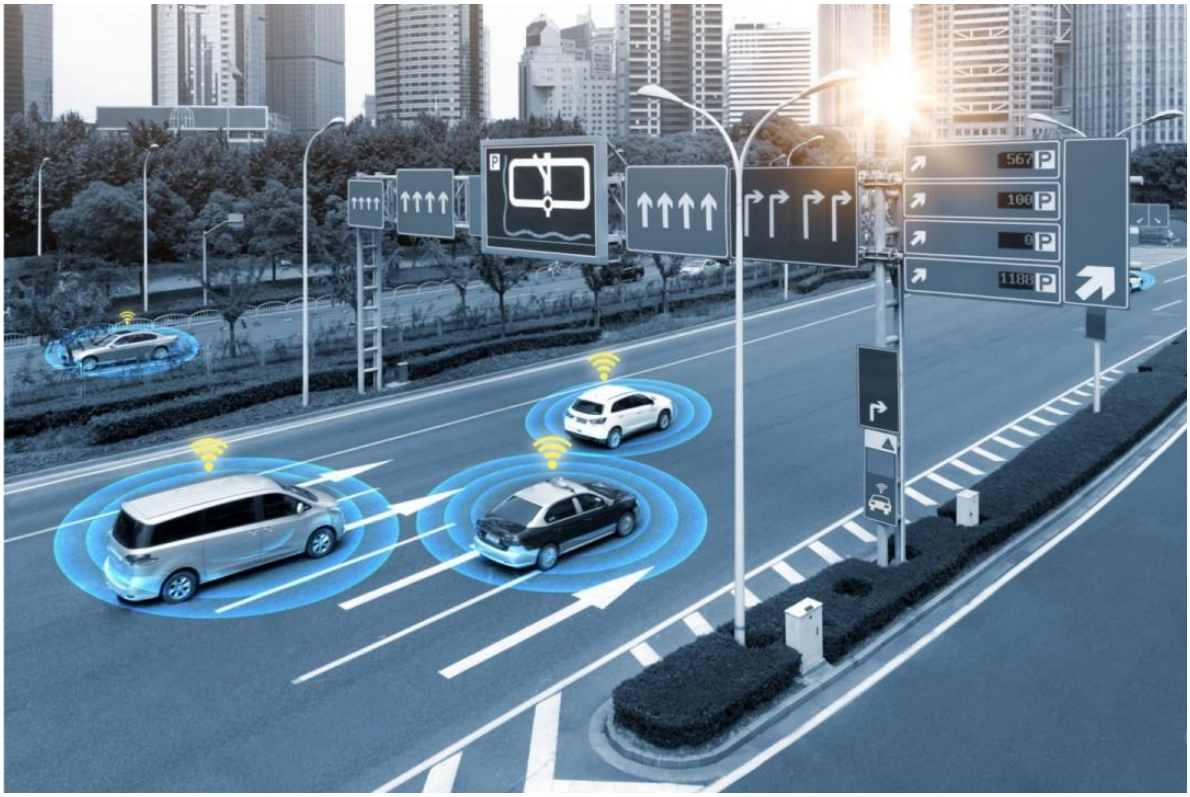


Figure 0.1 V2V System

0.2.2 V2V Market

➢ The commercial vehicle category is expected to witness the faster growth in V2V communication market, during the forecast period (2020-2030). Factors like stringent government regulations and rise in adoption of cloud-based fleet management telematics solutions in developed countries is expected to drive the market, globally [4].

➢ The original equipment manufacturer devices category dominated the vehicle-to-vehicle communication market, during the historical period (2015-2019). This is mainly because almost all the vehicle manufacturers provide in-built systems in their vehicles [4].

0.2.3 V2V Standard Communication Model

➢ Dedicated Short-Range Communication is the most reliable communication model in Vehicle-to-Vehicle to supply wireless communications features for ITS (Intelligent Transportation Systems) programs within a 1 KM range at standard speeds of the highway [5].

➢ Table **0.1** provides a comparison of Existing Communication Technologies for V2V Communication from many perspectives (Characteristics, Efficiency, *etc*.) [5]

Communication Technologies	Communication Protocols	Range	Characteristics
DSRC	IEEE 802.11p	1000 m	High Data Transfer Rate Reliable for large networks
Infrared	IEEE 802.11	10 m	Reliable, mature and easy to master
Bluetooth	IEEE 802.15.1	10 m	Low Power Good communication Security
Wi-Fi	IEEE 802.11a/b/g	76~305 m	High Data Transfer Rate
UWB	IEEE 802.15.3a	10 m	Strong anti-interference ability
Zigbee	IEEE 802.15.4	100 m	Low Cost Low Power
Mm Wave	IEEE 802.11ad/IEEE 802.15.3c	10 m	High Transmission Quality

Table **0.1**: V2V Communication technologies

0.3 Project Objectives

- The objective of this project is to design a system based on V2V communication technology, and also implement different use cases using this system to solve the problem of road accident, saving time by helping people to park easily and make the driving journey easily and more comfortable.

- This project aims to create a low-cost solution that can be implemented in large scale to help reduce a significant number of accidents. This is a fully autonomous system as it is, but an effective driver assistance system which helps the driver use an automobile in a safe way without getting into a crash situation from which the driver may find it hard to get out of.

- The project aims to make cars safer and more user-friendly by combining Adaptive Cruise Control (ACC), Automatic Emergency Braking (AEB), and a Self-parking system. The goal is to improve how these systems work together, ensuring smooth communication and coordination. Safety features, like collision detection and emergency braking, will be continually improved, and advanced models will help predict and avoid potential accidents. The self-parking system will be fine-tuned for different parking situations, and user interfaces will be made simpler for easy customization. Rigorous testing in different environments will ensure the system is reliable and complies with safety standards. The project also plans for future improvements in autonomous driving and explores partnerships with car manufacturers to put these integrated systems into commercial vehicles, making cars safer, smarter, and more widely used.

0.4 Project Descriptions

- This system will provide a great help to reduce road accidents by making communication between vehicles to collect surrounding vehicles' motion information this information is one of the keys for accident prevention by helping the driver to make the right decision in the right time.

- These services will enable the vehicle to transmit necessary data such as the current location, motion's direction and the speed directly to other vehicles, these vehicles would form a network, and pass information about road conditions, accidents, and congestion.

- The project aims to enhance automotive safety and convenience by integrating advanced driver assistance systems into a cohesive and intelligent framework. The project will primarily focus on three key systems: Adaptive Cruise Control (ACC), Automatic Emergency Braking (AEB), and a Self-Parking System.

1. Adaptive Cruise Control (ACC):

- ACC is a technology that automatically adjusts the vehicle's speed to maintain a safe following distance from the vehicle ahead.
- The system utilizes sensors, to monitor the traffic environment and dynamically adjust the vehicle's speed.

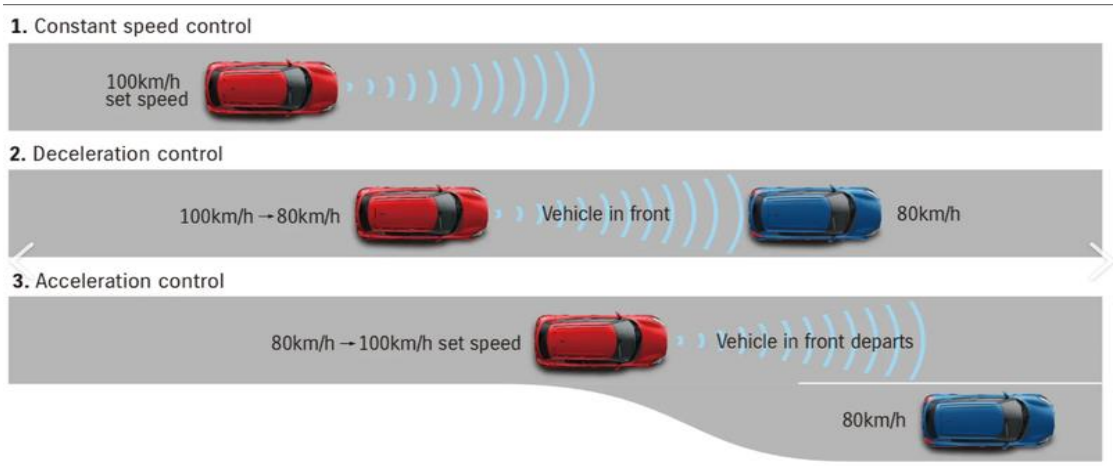


Figure 0.2 ACC System

2. Automatic Emergency Braking (AEB):

- AEB is a safety feature designed to detect an impending collision with another vehicle or obstacle and automatically apply the brakes to prevent or mitigate the impact.
- It relies on advanced sensor systems and real-time data processing to provide rapid and precise braking in emergency situations.

Figure 0.3 AEB System

3. Self-Parking System:

- The self-parking system is an automated technology that enables a vehicle to park itself without direct driver input.
- Utilizing a combination of sensors and advanced algorithms, the system identifies suitable parking spaces and executes parking maneuvers with high precision.

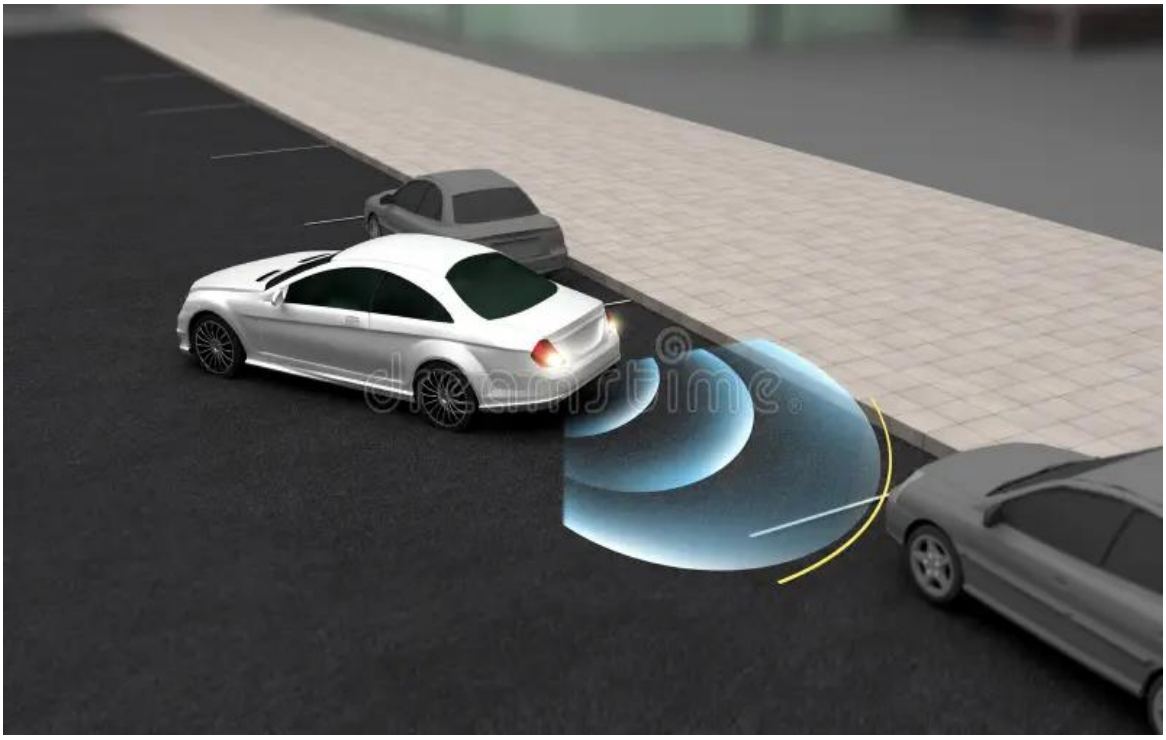


Figure 0.4 Self-Parking system

0.5 Expected Outcome

The major outcome of this project is aimed to the robot car that can map all road conditions, control of the car, and warning the driver in case there is danger to the driver or the car.

0.6 Research Methodology, Implementation and Validation

The methods that will use to finish this project are enlisted here.

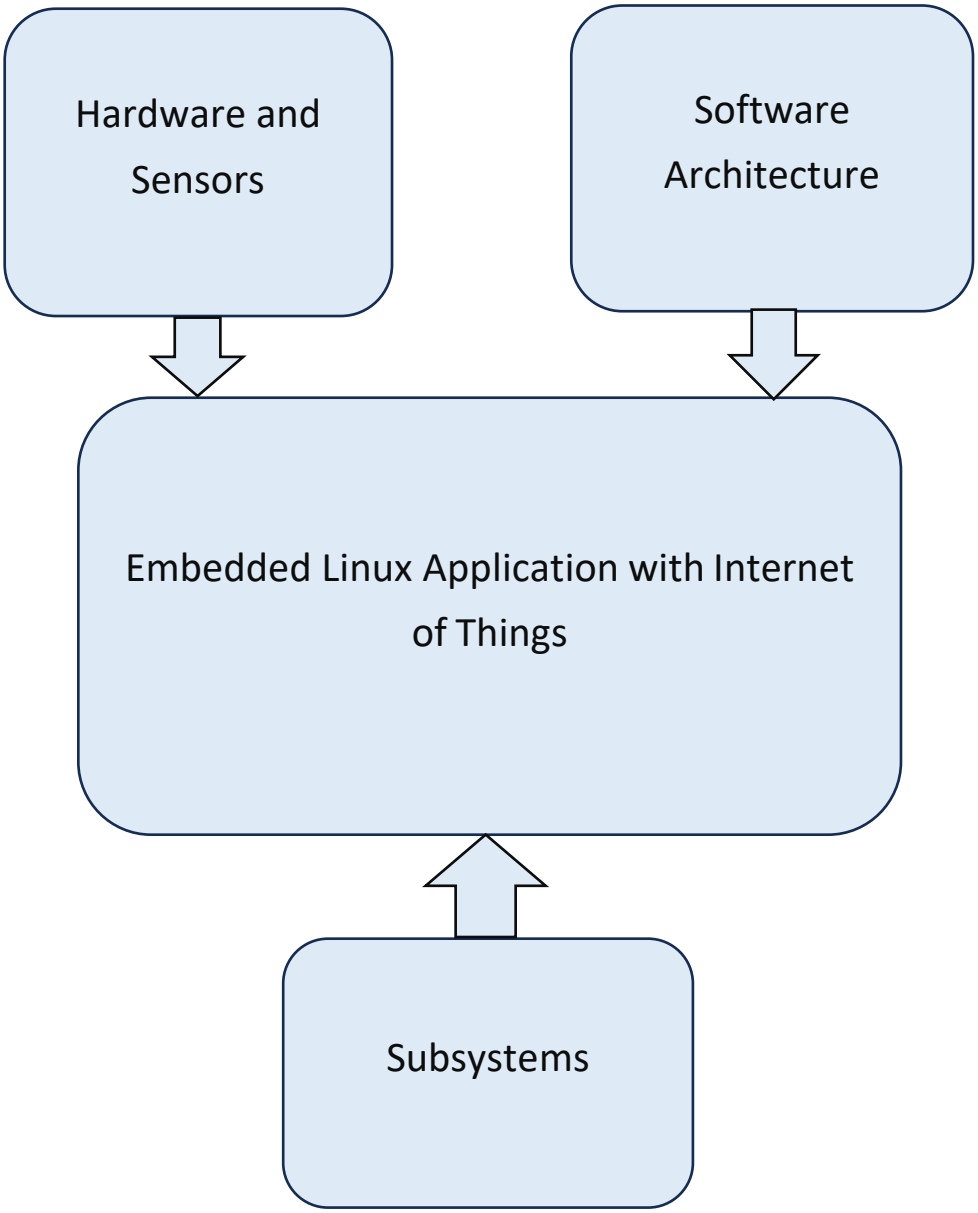
- Learn the basic principle of Real-Time Operating systems (RTOS), then choose the appropriate one and implement it in the project to control all system requirements.
- Learn the basic principle of interfacing and coding the drivers, then implement it in the project. Learn the basic principle of Automotive and How car works, then implement it in the project to control of all cases of road.
- Learn the basic principle of ARM and interfacing with blue pill, then implement it in the project.
- Learn the basic principle of Linux administration and interfacing with RPI, then implement it in the project.
- Learn the basic principle of Battery Management System (BMS), then implement it in the project to control the power of the robot and charge time.
- Programming of microcontroller to control the I/O devices. Circuits developments and design for the robot control unit.
- Connect all parts of the systems and test the communication between them.
- Making hardware prototype and testing the robot function.

0.7 Project Architecture

Our Project contains 4 modules:

- Module A: Hardware and Sensors.
- Module B: Software Architecture.
- Module C: Subsystems.
- Module D: Embedded Linux Application with Internet of Things.

We'll talk about them individually.



Module A: Hardware and Sensors

We'll explain the hardware used, the ECU used and components of it, the communication of ECU and the components, the Sensors which connected to the overall system.

-This module consists of three chapters:

Chapter 1: STM32F103C8T6.

Chapter 2: The Raspberry Pi.

Chapter 3: The Sensors and Actuators.

Module B: Software Architecture

We'll explain the software architecture used, which is the communication between the two blue pill and the two-raspberry pi.

-This module consists of two chapters:

Chapter 4: Universal Asynchronous Receiver Transmitter (UART).

Chapter 5: Bluetooth.

Module C: Subsystems

We'll explain the Subsystems used to solve the problem of collision and crash, these are three Applications (ACC, AEB and Self-Parking) and explain the function of each subsystem and the role of it in the system overall.

-This module consists of three chapters:

Chapter 6: ACC System.

Chapter 7: AEB System.

Chapter 8: Self-Parking System.

Module D: Embedded Linux Application with Internet of Things

We'll explain the Linux operating system especially to perform Embedded Linux and its architecture. Then we'll explain the IoT value chain, MQTT protocol, Raspberry PI, and system application of our robot.

-This module consists of three chapters:

Chapter 9: Embedded Linux.

Chapter 10: Internet of Things (IoT).

Chapter 11: Embedded Linux &Internet of Things (IoT) tasks.

Module A
Hardware & Sensors

Chapter 1: STM32F103C8T6.

Chapter 2: The Raspberry Pi.

Chapter 3: The Sensors and Actuators.

CHAPTER 1: BLUE PILL (STM32F103C8T6)

In this chapter, we'll talk about the STM32F103C8T6.Also, we'll discuss why STM32F103C8T6, its Versions, Hardware Component, Interfacing with GPIO pins and Operating System.

1.1 Introduction

The STM32F103C8T6, often referred to simply as the STM32F103 or more commonly as the "Blue Pill" in hobbyist circles, is a popular microcontroller from STMicroelectronics. It belongs to the STM32F1 series and features an ARM Cortex-M3 core, which is a 32-bit RISC processor capable of executing instructions with high efficiency. This microcontroller is widely used in various applications ranging from industrial automation to consumer electronics and hobbyist projects due to its versatility, performance, and cost-effectiveness.

This board offers a range of features including various sensors, communication interfaces (such as UART, SPI, I2C, CAN), GPIO pins. Its ARM Cortex-M3 processor running at up to 72 MHz, coupled with generous Flash and RAM resources, makes it suitable for a wide range of applications including industrial automation, IoT devices, robotics, and more.

The STM32F103C8T6 is known for its ease of use in both professional and hobbyist environments, thanks to its rich feature set and availability of development resources. Its affordability and broad compatibility with various software tools and libraries make it a preferred choice for embedded systems development across different industries.

1.2 Why STM32F103C8T6?

The STM32F103C8T6 microcontroller is favored by developers and engineers for a variety of reasons, making it a popular choice in the embedded systems industry and among hobbyists alike:

1. **Powerful ARM Cortex-M3 Core:**
    - The STM32F103C8T6 is based on the ARM Cortex-M3 core, which provides significant processing power (up to 72 MHz) for real-time processing tasks. This makes it suitable for applications requiring efficient computation and fast response times.
2. **Rich Peripherals and Connectivity:**
    - It integrates a wide range of peripherals including UARTs, SPI, I2C, ADCs, DACs, timers, and more. This versatility allows developers to interface with various sensors, actuators, displays, and communication modules easily.
3. **Memory and Storage Options:**
    - With 64 KB of Flash memory and 20 KB of SRAM, the STM32F103C8T6 offers sufficient memory for storing program code and data, enabling more complex applications and algorithms.
4. **Low Power Consumption:**
    - It is designed to operate efficiently in low-power modes, making it suitable for battery-powered applications where power consumption is critical.
5. **Cost-Effective Solution:**
    - The STM32F103C8T6 is cost-effective compared to higher-end STM32 microcontrollers while still offering robust features and performance. This makes it accessible for both commercial and hobbyist projects.
6. **Extensive Development Ecosystem:**
    - Supported by a comprehensive development ecosystem including STM32CubeMX for graphical MCU configuration and initialization, STM32CubeIDE for integrated development environment, and STM32CubeHAL (Hardware Abstraction Layer) for low-level software development. This ecosystem streamlines the development process and reduces time-to-market.

7. **Community Support and Documentation:**
    - Due to its popularity, the STM32F103C8T6 has extensive community support, forums, tutorials, and example projects available online. This makes it easier for developers to troubleshoot issues, learn, and share knowledge.
8. **Versatility Across Applications:**
    - Its versatility makes it suitable for a wide range of applications including industrial automation, consumer electronics, IoT devices, motor control, robotics, and more. Developers can adapt it to various project requirements with relative ease.

In conclusion, the STM32F103C8T6 microcontroller is chosen for its powerful ARM Cortex-M3 core, rich peripherals, low power consumption, cost-effectiveness, extensive development tools, and broad application suitability. These factors collectively make it a preferred choice for embedded systems development, from simple control tasks to more complex and demanding applications.

1.3 STM32F103C8T6 Board Features [6]

Includes ST state-of-the-art patented technology

- Arm® 32-bit Cortex®-M3 CPU core
    - 72 MHz maximum frequency, 1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
    - Single-cycle multiplication and hardware division
- Memories
    - 64 or 128 Kbytes of Flash memory
    - 20 Kbytes of SRAM
- Clock, reset and supply management
    - 2.0 to 3.6 V application supply and I/Os
    - POR, PDR, and programmable voltage detector (PVD)
    - 4 to 16 MHz crystal oscillator
    - Internal 8 MHz factory-trimmed RC
    - Internal 40 kHz RC – PLL for CPU clock
    - 32 kHz oscillator for RTC with calibration
- Low-power
    - Sleep, Stop and Standby modes
    - VBAT supply for RTC and backup registers
- 2x 12-bit, 1 µs A/D converters (up to 16 channels)
    - Conversion range: 0 to 3.6 V

- Dual-sample and hold capability
- Temperature sensor

- DMA
  - 7-channel DMA controller
  - Peripherals supported: timers, ADC, SPIs, I 2Cs and USARTs
- Up to 80 fast I/O ports
  - 26/37/51/80 I/Os, all mappable on 16 external interrupt vectors and almost all 5 V-tolerant
- Debug mode:
  - Serial wire debug (SWD) and JTAG interfaces
- Seven timers
  - Three 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
  - 16-bit, motor control PWM timer with dead-time generation and emergency stop
  - Two watchdog timers (independent and window)
  - SysTick timer 24-bit down counter
- Up to nine communication interfaces
  - Up to two I2C interfaces (SMBus/PMBus®)
  - Up to three USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)
  - Up to two SPIs (18 Mbit/s)
  - CAN interface (2.0B Active)
  - USB 2.0 full-speed interface
- CRC calculation unit, 96-bit unique ID
- Packages are ECOPACK®

1.4 STM32F103C8T6 Board Hardware

In this Part, we will talk about the STM32F103C8T6 Specifications Pin Diagram and Description which we are used it in our project. Also, we will discuss the Layout and Interfacing of GPIO pins and Hardware Components of it.

1.4.1 Hardware component of STM32F103C8T6 [8]

- STM32F103C8T6 provides the following hardware components:

1. **ARM Cortex-M3 Core:**

- The heart of the STM32F103C8T6 is its ARM Cortex-M3 processor core, which operates at speeds up to 72 MHz. This 32-bit core provides efficient processing power suitable for real-time applications.

2. **Memory:**

- **Flash Memory:** The STM32F103C8T6 typically includes 64 KB of Flash memory for storing program code. This non-volatile memory retains its contents even when the power is turned off.
- **SRAM (Static Random Access Memory):** It features 20 KB of SRAM for temporary data storage during program execution. SRAM provides faster access times compared to Flash memory.

3. **Clock and Reset Management:**

- **Internal Oscillators:** The microcontroller includes internal RC oscillators and may support external crystal oscillators for accurate timing and clock generation.
- **Reset Circuitry:** Includes hardware reset sources such as power-on reset (POR), brown-out reset (BOR), and external reset pins for system initialization and recovery.

4. **Peripherals:**

- The STM32F103C8T6 integrates a variety of peripherals, including but not limited to:

    - **Timers:** General-purpose timers, advanced-control timers (PWM), and watchdog timers.
    - **Communication Interfaces:** UARTs (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), CAN (Controller Area Network), and USB (Universal Serial Bus).
    - **Analog-to-Digital Converters (ADCs):** For converting analog signals from sensors or other sources into digital data.
    - **Digital-to-Analog Converters (DACs):** Some variants may include DACs for generating analog signals.
    - **GPIO (General-Purpose Input/Output):** For interfacing with external devices and peripherals.

5. **Power Management:**

- Includes power supply circuitry for various voltage levels (typically 3.3V and sometimes 5V tolerant I/O), and low-power modes to optimize energy consumption.

6. **Debug and Programming Interfaces:**
- **JTAG/SWD (Serial Wire Debug):** Used for programming and debugging the microcontroller.
- **Bootloader Support:** Allows firmware updates and programming via different interfaces (USB, UART, etc.) depending on the specific board configuration.

7. **Package and Pin Configuration:**
- The STM32F103C8T6 is available in various package options (such as LQFP48) with a defined number of I/O pins for connecting to external components and circuits.

1.4.2 Layout of the GPIO headers [7]

The GPIO (General-Purpose Input/Output) headers on the STM32F103C8T6, commonly found on development boards like the "Blue Pill," typically have a specific layout. Here's a general description of how these GPIO headers is organized:

1. **Pin Headers:**
    - The STM32F103C8T6 development board usually has two rows of pin headers, one on each side of the board.
    - Each header row typically consists of 20 pins (10 pins per row).
2. **Pin Configuration:**
    - **Top Row (Typically Even Pins):** Pins are numbered consecutively from 2 to 40 (for a total of 20 pins).
    - **Bottom Row (Typically Odd Pins):** Pins are also numbered consecutively, usually from 1 to 39 (for a total of 20 pins).
3. **Pin Functions:**
    - The pins on these headers serve various functions, including GPIO, power supply, ground, and communication interfaces (such as UART, SPI, I2C).
    - **GPIO Pins:** Several of these pins are dedicated to GPIO functions, allowing digital input and output operations.
4. **Power Supply and Ground Pins:**
    - Typically, pins 1 and 2 (bottom row) are connected to VCC (3.3V) and GND respectively.
    - Other pins might also be designated for specific power supply voltages (e.g., 5V, VIN) and additional ground connections.

5. **Communication Interfaces:**
    - Some pins might be multiplexed to support serial communication interfaces like UART, SPI, and I2C.
    - These pins often have dual functionalities, where they can be configured either as GPIO or for specific communication protocols.
6. **Labeling and Documentation:**
    - The pin headers are usually labeled on the PCB itself or in the accompanying documentation.
    - The STM32 microcontroller datasheet and reference manual provide detailed pinout diagrams and descriptions, which are crucial for understanding the exact functions and capabilities of each pin.
7. **Connection to External Components:**
    - Users can connect external sensors, actuators, displays, and other peripherals directly to these GPIO headers, enabling easy prototyping and development of embedded systems.

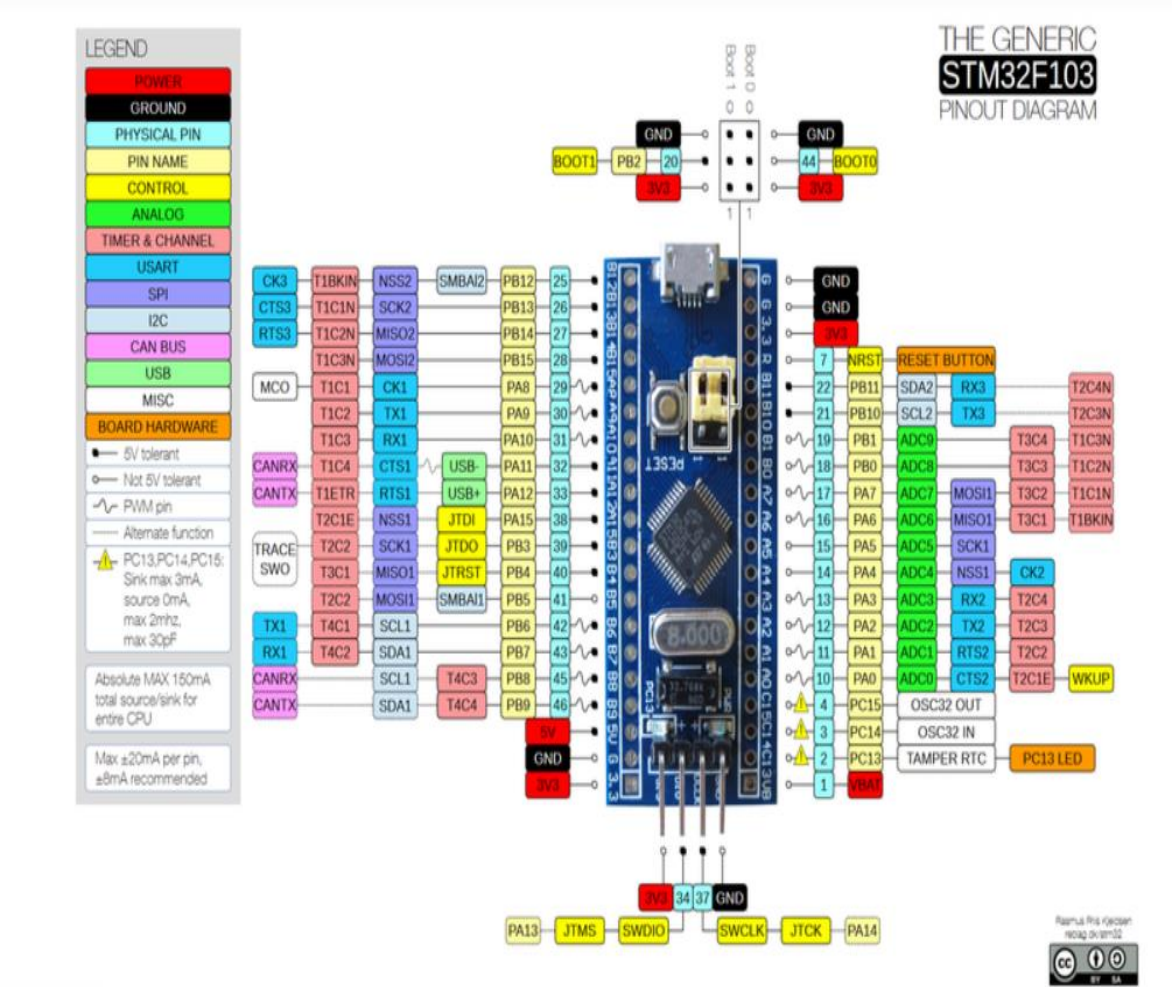


Figure 1.1 Blue Pill Pin Configuration

1.5 STM32F103C8T6 OS[9]

The STM32F103C8T6 microcontroller itself does not have a built-in operating system (OS) in the traditional sense like desktop or server operating systems such as Windows or Linux. Instead, it is typically used in embedded systems where it runs a real-time operating system (RTOS) or operates in a bare-metal (no OS) environment, depending on the application requirements.

-Here are the common scenarios for using an STM32F103C8T6 microcontroller with different types of software environments:

1. **Bare-Metal Programming:**
    - Many applications for the STM32F103C8T6 involve writing firmware directly for the microcontroller without an operating system. In bare-metal programming, developers directly control

hardware peripherals and manage tasks and timing within their application code.

2. **Real-Time Operating Systems (RTOS):**
    - For more complex applications requiring multitasking, task scheduling, and better resource management, developers may opt to use an RTOS. Popular RTOS choices for STM32 microcontrollers include FreeRTOS, RTX (from Keil), and STM32CubeRTOS. These RTOS provide scheduling mechanisms, synchronization primitives, and other features to simplify development of complex embedded systems.
3. **Embedded Frameworks and Libraries:**
    - STM32 microcontrollers are often programmed using vendor-provided frameworks and libraries such as STM32CubeMX and STM32 HAL (Hardware Abstraction Layer). These frameworks simplify configuration and use of microcontroller peripherals like timers, UARTs, ADCs, etc., abstracting low-level hardware details.
4. **Custom Operating Systems:**
    - In some cases, developers may choose to port or develop custom lightweight operating systems tailored to specific application needs. This approach is less common for STM32F103C8T6 due to its resource constraints compared to higher-end STM32 series microcontrollers.

In summary, while the STM32F103C8T6 microcontroller itself does not come with a full-fledged operating system, it is highly versatile and supports various software environments ranging from bare-metal programming to real-time operating systems. The choice of software environment depends on factors such as application complexity, real-time requirements, resource constraints, and developer preference.

CHAPTER 2: RASPBERRY PI

In this chapter we will discuss what is the raspberry pi, why Raspberry pi, its Versions, Hardware Component, Interfacing with GPIO pins and Raspberry pi Operating System.

2.1 Introduction

The Raspberry Pi is probably the most popular electronics prototyping platform ever. Its creators, the Raspberry Pi Foundation, wanted to create a single-board computer that could be used to teach computer science and programming to high school students in developing countries.

2.1.1 What is a Raspberry Pi?

- Raspberry Pi is a low-cost, credit card-sized computer that was first developed in 2012 by the Raspberry Pi Foundation in the UK.
- It can connect to a huge variety of sensors and other modules like LCD displays, servos, and motors.
- The Raspberry Pi was designed to be a tool to promote the teaching of basic computer science in schools and developing countries.
- The original Raspberry Pi was equipped with a 700 MHz ARM11 processor, 256 MB of RAM, and had a single USB port, however, today's models are much more powerful, with quad-core processors, up to 8GB of RAM, and multiple USB and HDMI ports.
- The Raspberry Pi can run a variety of operating systems, including Linux distributions, Windows 10, and even Android, it can be used for a wide range of projects, from simple tasks such as browsing the web and playing games, to more complex projects such as controlling robots, home automation, and even building a supercomputer.

Overall, the Raspberry Pi has become an incredibly versatile and useful tool for anyone interested in learning about computing and electronics. There’re

multiple types of mobile robotics such as wheeled, tracked, legged, flying, and underwater robots.

We used Raspberry pi 3 model B+ and Raspberry pi 3 model B in our project:

The speed and performance of the new Raspberry Pi 3 is a step up from earlier models. For the first time, Raspberry Pi 3is a complete desktop experience. Whether you're editing documents, browsing the web with a bunch of tabs open, juggling spreadsheets or drafting a presentation, we'll find the experience smooth and very recognizable, but on a smaller, more energy- efficient and much more cost-effective machine.

-Raspberry Pi 3 features can list as:

1. Silent, energy-efficient: The fan less, energy-efficient Raspberry Pi runs silently and uses far less power than other computers.

2. Fast networking: Raspberry Pi 3 comes with Gigabit Ethernet, along with onboard wireless networking and Bluetooth [3].

3. USB 3: Raspberry Pi 3 has upgraded USB capacity: along with two USB 2 ports we'll find two USB 3 ports, which can transfer data up to ten times faster.

4. Choice of RAM: There are different variants of the Raspberry Pi 3 available, depending on how much RAM you need — 1GB, 2GB, 4GB, or 8GB.

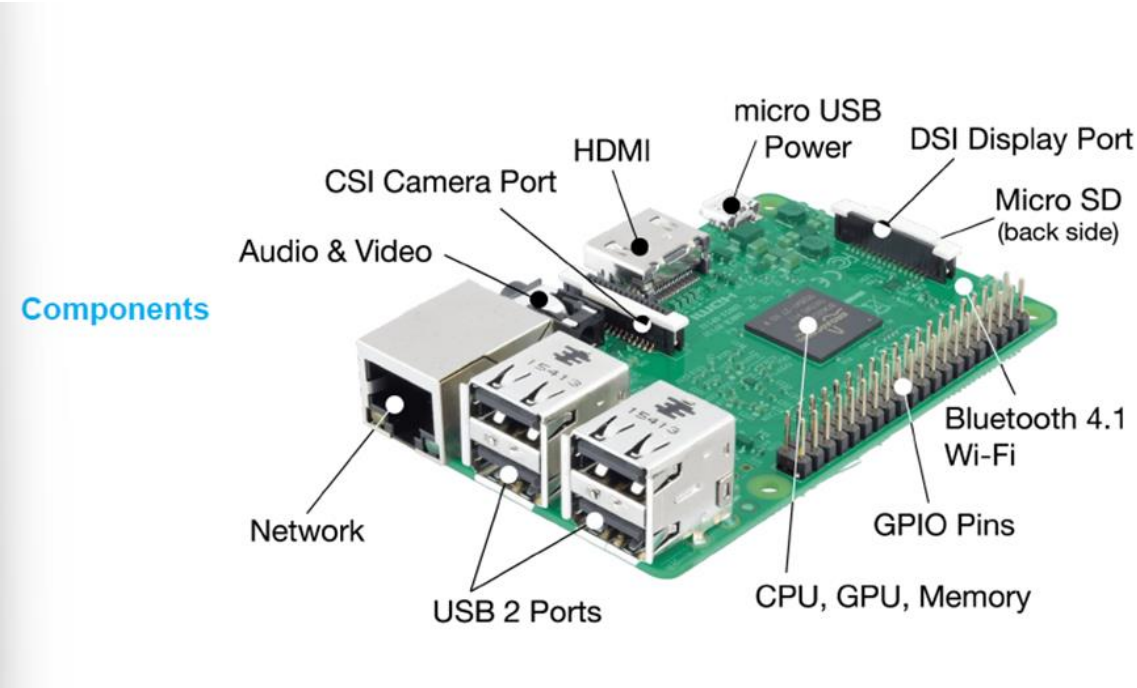


Figure 2.1 Raspberry pi

2.2 Why Raspberry pi

Raspberry Pi has many advantages that make it a popular choice for a wide range of projects and applications. Here are some reasons why Raspberry Pi is a great choice:

- Low cost (Cheap): Raspberry Pi is a low-cost computer, making it an affordable option for beginners or for use in projects where cost is a concern.
- Small Size: The Raspberry Pi is small and lightweight, so it's easy to carry around and fit into tight spaces.
- Low Power Consumption: Raspberry Pi uses very little power, making it an energy-efficient option.
- Flexibility: The Raspberry Pi can run a variety of operating systems and applications, making it a versatile tool for a wide range of projects.
- GPIO Pins (easily access to the GPIO): The Raspberry Pi's GPIO pins make it easy to connect to a wide range of sensors and devices, allowing you to build custom electronics projects.
- Community Support: Raspberry Pi has a large and active community of

users and developers who share knowledge and resources, making it easy to get help and find solutions to problems.

- Educational Value: Raspberry Pi was designed as a tool to promote the teaching of basic computer science in schools and developing countries, so it's a great way to learn about computing and electronics.
- Complete computing platform.
- Runs Linux (MMU).

Overall, Raspberry Pi is a cost-effective, flexible, and versatile tool that can be used for a wide range of projects and applications, making it a popular choice for hobbyists, students, and professionals alike.

2.3 Raspberry pi Versions [10]

Here are the different versions of Raspberry Pi that have been released so far, each version of the Raspberry Pi has its own unique features and capabilities, and they can be used for a wide range of applications, including education, hobby projects, and industrial automation:

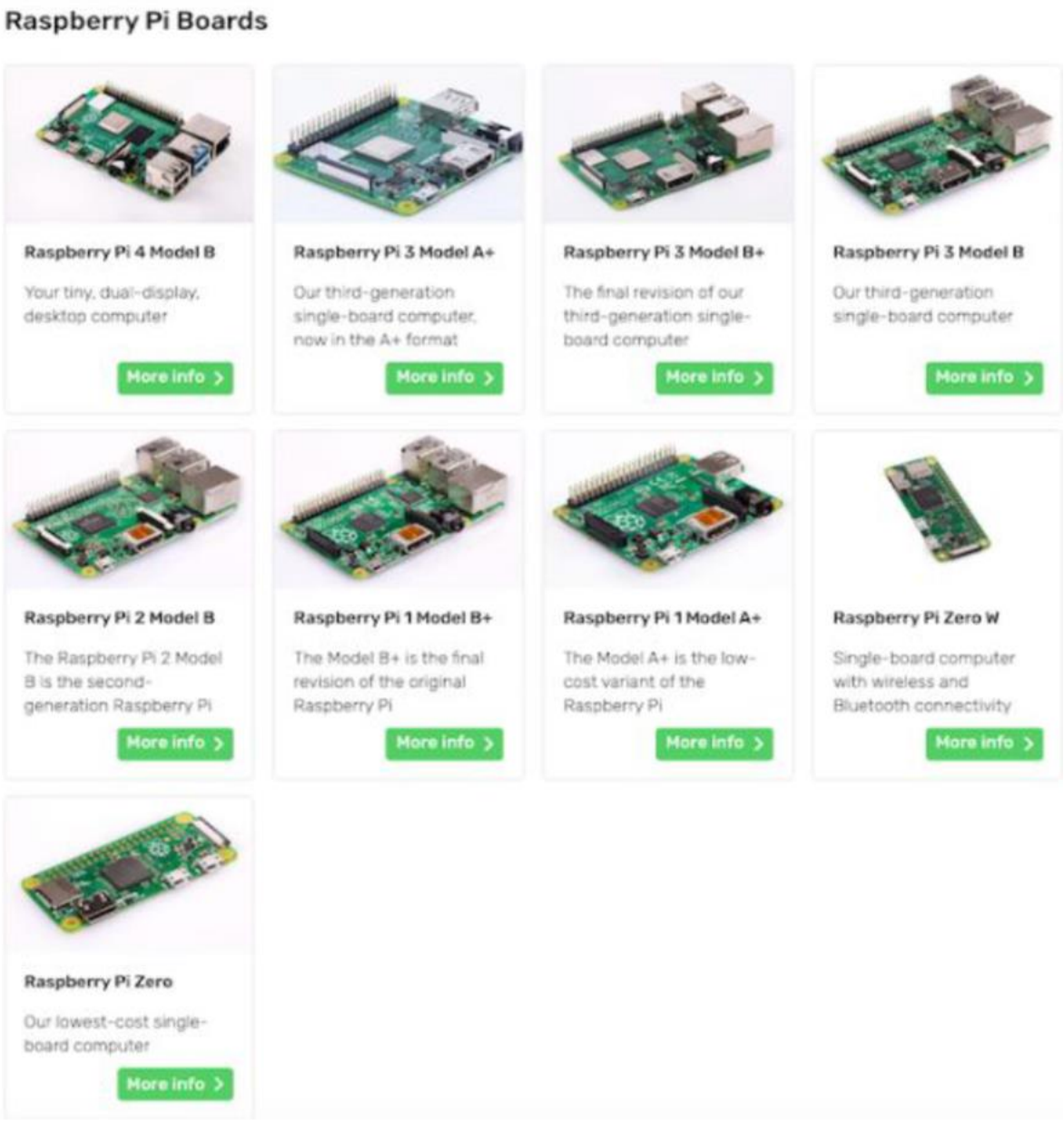


Figure 2.2 Raspberry Pi boards

- Raspberry Pi Model A: Released in 2012, it was the first Raspberry Pi board and featured a 700MHz ARM11 processor, 256MB of RAM, one USB port, and no Ethernet port.
- Raspberry Pi Model B: Released in 2012, it was a higher-end version of the Model A and featured a 700MHz ARM11 processor, 512MB of RAM, two USB ports, and an Ethernet port.
- Raspberry Pi Model A+: Released in 2014, it was a smaller and cheaper version of the Model A, with 256MB of RAM, one USB port, and no

Ethernet port.

- Raspberry Pi Model B+: Released in 2014, it featured a 900MHz quad-core ARM Cortex-A7 processor, 1GB of RAM, four USB ports, and an Ethernet port.
- Raspberry Pi 2 Model B: Released in 2015, it featured a 900MHz quad-core ARM Cortex-A7 processor, 1GB of RAM, four USB ports, and an Ethernet port.
- Raspberry Pi Zero: Released in 2015, it was a smaller and cheaper version of the Model A, with a single-core 1GHz ARM11 processor, 512MB of RAM, and a micro-USB port.
- Raspberry Pi 3 Model B: Released in 2016, it featured a 1.2GHz 64-bit quad-core ARM Cortex-A53 processor, 1GB of RAM, four USB ports, and an Ethernet port.
- Raspberry Pi Zero W: Released in 2017, it was the same as the Zero but with added wireless connectivity, including Wi-Fi and Bluetooth.
- Raspberry Pi 3 Model B+: Released in 2018, it featured a 1.4GHz 64-bit quad-core ARM Cortex-A53 processor, 1GB of RAM, four USB ports, and an Ethernet port.
- Raspberry Pi 4 Model B: Released in 2019, it featured a 1.5GHz 64-bit quad-core ARM Cortex-A72 processor, up to 8GB of RAM, two USB 2.0 ports, two USB 3.0 ports, and two micro-HDMI ports.
- Raspberry Pi Compute Module 3: Released in 2017, it was designed for industrial applications and included a 1.2GHz quad-core ARM Cortex-A53 processor, 1GB of RAM, and 4GB of eMMC flash storage.
- Raspberry Pi 400: Released in 2020, it featured a Raspberry Pi 3 board built into a compact keyboard, with 4GB of RAM, two USB 3.0 ports, and one USB 2.0 port.
- Raspberry Pi Compute Module 4: Released in 2020, it was the latest version of the Compute Module and included a choice of processors, up to 8GB of RAM, and various storage options.

Figure 1.2 and figure 1.3 provides a summary feature comparison of the different RPi models that are presently available. Here is a quick summary of this table:

Model	RPi 2 B	RPi 3 B	RPi 3 B+
SOC Type	Broadcom BCM2836	Broadcom BCM2837	Broadcom BCM2837B0
CPU Clock	900 MHz Quad Core ARM Cortex-A7	1.2 GHz Quad Core ARM Cortex-A53	1.4 GHz Quad Core ARM Cortex-A53
RAM	1 GB	1 GB	1 GB
GPU	Broadcom VideoCore IV 1080p30	Broadcom VideoCore IV 1080p60	Broadcom VideoCore IV 1080p60
USB Ports	4	4	4
Ethernet	100 Mbit/s base Ethernet	100 Mbit/s base Ethernet	Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps
'ower over Ethernet	No	No	Yes (requires separate PoE HAT)
WiFi	No	On Board WiFi 802.11n	On Board WiFi 802.11ac Dual Band 2.4 GHz & 5 GHz
Bluetooth	No	On Board Bluetooth 2.0/4.1	On Board Bluetooth 2.0/4.1/4.2 LS BLE
Video Output	HDMI 3.5 mm Composite DSI (for LCD)	HDMI 3.5 mm Composite DSI (for LCD)	HDMI 3.5 mm Composite DSI (for LCD)
Audio Output	I^2 S HDMI 3.5 mm Composite	I^2 S HDMI 3.5 mm Composite	I^2 S HDMI 3.5mm Composite
Camera Input	15 Pin CSI	15 Pin CSI	15 Pin CSI
GPIO Pins	40	40	40
Memory	MicroSD	MicroSD	MicroSD

Table 2.1 Comparison of different RPi

- If you need an RPi for general-purpose computing, consider the RPi 3.
- The 1 GB of memory and 1.2 GHz quad-core processor provide the best performance out of all the boards.
- For applications that interface electronics circuits to the Internet on a wired network, consider the RPi 3, RPi 2, or RPi B+, with cost being the deciding factor.
- If you need a small-footprint device with wireless connectivity, consider the RPi Zero. The RPi A+ could be used to develop the initial prototype.

- If you want to design your own PCB that uses the RPi (or multiple RPi boards), investigate the Compute module.

	Raspberry Pi 4 B	Raspberry Pi 3 Model A+
Image		
Release date	2019 Jun 24	2018 Nov 15
Description		
Product details		
Price	US$35.00	US$25.00
SOC		
SOC Type	Broadcom BCM2711	Broadcom BCM2837B0
Core Type	Cortex-A72 (ARM v8) 64-bit	Cortex-A53 64-bit
No. Of Cores	4	4
GPU		VideoCore IV
CPU Clock	1.5 GHz	1.4 GHz
RAM	1 GB , 2 GB, 4 GB	512 MB DDR2
Wired Connectivity		
USB	2x USB3.0 + 2x USB2.0	1xUSB 2.0
Ethernet	Gigabit	
SATA Ports		
HDMI port	2x micro HDMI	
Analog Video Out	shared with audio jack	shared with audio jack
Analog Audio Out	3.5mm jack	3.5mm jack
Analog Audio Out	3.5mm jack	3.5mm jack
Analog Audio In		
SPI		
I2C		
GPIO		
LCD Panel		
Camera		
SD/MMC	microSD	microSD
Serial		
Wireless Connectivity (On-Board)		
Wi-Fi	2.4GHz and 5GHz 802.11 b/g/n/ac	2.4GHz and 5GHz 802.11 b/g/
Bluetooth	5.0	4.2, BLE
Dimensions		
Height	3.37 in	2.55 in
Width	2.22 in	2.20 in
Depth	0.43307 in	0.43307 in
Weight		1.02 oz
Power		
Power ratings		
Power sources	USB-C	microUSB, GPIO
Power Over Ethernet	with PoE Hat	

Figure 2.3 A summary comparison of commonly available RPi 4 B & RPI 3 model A+

2.4 Raspberry pi Hardware

In this Part, we'll talk about the Raspberry pi 3 model B and model B+ Specifications Pin Diagram and Description which we are used it in our project. Also, we'll discuss the Layout and Interfacing of GPIO pins and Hardware Components of Raspberry pi.

2.4.1 Specifications Pin Diagram and Description

Here are the specifications of the Raspberry Pi 3 Model B &B+, the latest version of the Raspberry Pi as of my knowledge cut-off in September 2021:

- It has a 64bit quad-core processor having cortex-A72 (ARM v8) clocked

@1.5GHz.

- The new Pi board includes Broadcom BCM2711 VC6 GPU able to handle two 4kp30 displays also it can handle H.265 decoding at 4kp60. It has two micro-HDMI ports.
- Now the new Pi board comes in 2GB, 4GB, and 8GB LPDDR-4 RAM options.
- It has a dual-band 2.4/5.0 GHz Wi-Fi, Bluetooth 5.0, Gigabit Ethernet Port, 2 USB 3.0, and 2 USB 2.0 ports.
- USB type C power input port. Also, it has POE capability via separate POE HAT (add-on).
- It has a standard 40 pin GPIO header (having backward compatibility)
- The new Raspberry Pi is even more fast, powerful, and versatile for a huge variety of projects involving robotics, automation, image processing, AI, and many more.

2.4.2 Hardware Components of Raspberry pi [10]

Raspberry Pi is a credit card-sized computer that can be used for a variety of projects, including robotics, home automation, and media centers. It consists of several hardware components, including:

1. Processor Of Raspberry: Pi has a more powerful quad-core Cortex-A72 64-bit CPU running at 1.5GHz and Broadcom Video Core VI GPU running at 0.5GHz. It can handle 4k videos, H.265 encoding 2 HDMI outputs. It has a fully functional Gigabit Ethernet interface and USB 3.0 port

Figure 2.4 Processor of Raspberry pi

2. Ram Memory: The new Raspberry Pi 4 comes with an option of 2GB, 4GB, and 8GB of RAM options. It has LPDDR4 RAM. Also, you get a heat sink for the RAM chip as it gets hot while operating.

Figure 2.5 Ram Memory of Raspberry pi

3. Ethernet Controller: This Pi board has a Broadcom BCM54213 Gigabit ethernet controller. It falls to a slower speed if the network speed is slow.

- Software-driven router.
- Media server.
- A bridge between Wi-Fi and Ethernet networks.

Figure 2.6 Ethernet Controller of Raspberry pi

4. USB Controller: VL805 chip is the USB port controller for Raspberry Pi 3. The Pi has 2 x USB 3.0 and 2 x USB 2.0 ports. VL805 chip is a USB 3.0 Host controller. it allows the PCI Express platform to interface with USB Super-Speed (5 Gbps), High-Speed (480 Mbps), Full-Speed (12 Mbps), and Low-Speed (1.5 Mbps) devices. The root hub is consisting of four downstream facing ports, it enables simultaneous operation of multiple peripheral devices.

Figure 2.7 USB Controller of Raspberry pi

5. Power Circuitry: The Pi3 uses the MXL7704 it was developed specifically for Raspberry. This chip reduced the complexity of the Pi board and delivers more power to all the peripheral to run at more power. Due to this chip cost of the board has also been reduced as the number of components has been reduced.

Figure 2.8 Power Circuitry of Raspberry pi

2.4.3 Layout of The GPIO header

The GPIO header allows users to connect a wide range of devices and peripherals to the Raspberry Pi, making it highly versatile and flexible for a wide range of projects.

And here is a pin diagram of the Raspberry Pi 3 Model B &B+, showing the layout of the GPIO header there are 40 GPIO Pins of Raspberry pi 3:

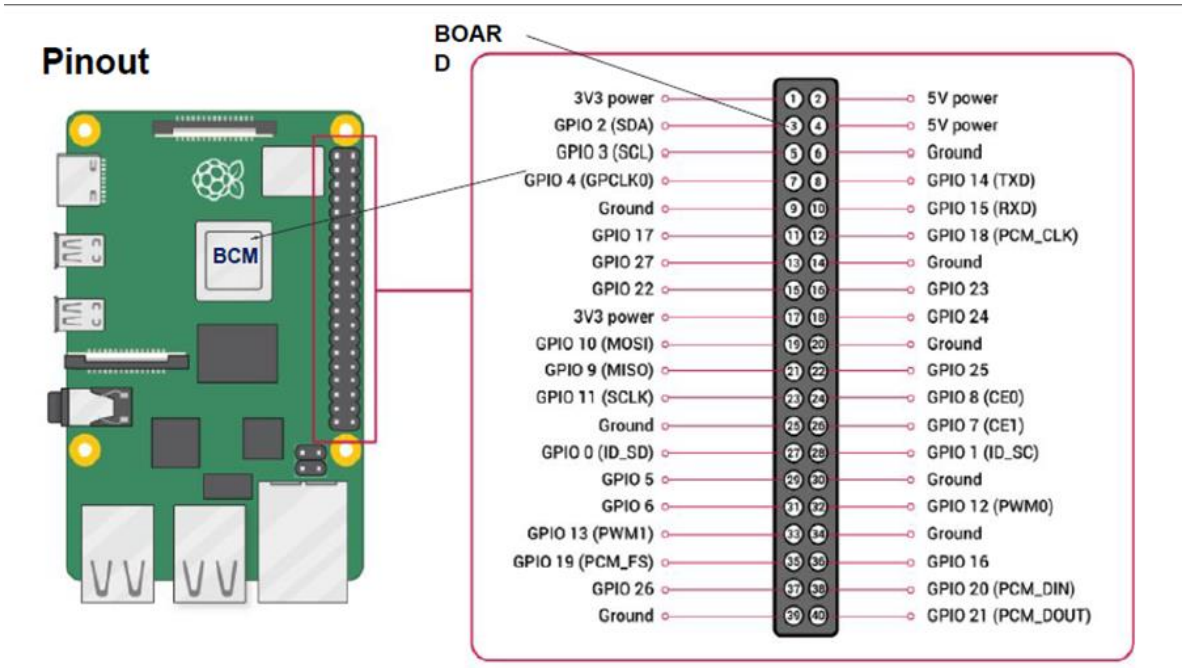


Figure 2.9 GPIO Pins of Raspberry pi

2.5 Raspberry pi OS [10]

Raspberry Pi OS (formerly known as Raspbian) is the official operating system for the Raspberry Pi.

It is a free and open-source operating system based on the Debian Linux distribution.

Raspberry Pi OS is specifically designed for the Raspberry Pi hardware and comes pre-installed with a variety of software.

-Here are some of the features and components of Raspberry Pi OS:

1. Desktop Environment: Raspberry Pi OS comes with a desktop environment based on the LXDE desktop environment, which is lightweight and optimized for the Raspberry Pi's hardware.

2. Software: Raspberry Pi OS comes with a variety of pre-installed software, including the Chromium web browser, the Python programming language, and the Thonny Python IDE.

3. Configuration Tools: Raspberry Pi OS includes a variety of tools for configuring and customizing the operating system, including the Raspberry Pi Configuration tool and the raspi-config tool.

4. Command-Line Interface: Raspberry Pi OS includes a powerful command-line interface, which can be used to perform system administration tasks and run scripts.

5. Package Manager: Raspberry Pi OS includes the apt package manager, which can be used to install new software packages and keep the operating system up to date.

6. GPIO Support: Raspberry Pi OS includes support for the Raspberry Pi's GPIO pins, which can be used to interface with external hardware and sensors.

Overall, Raspberry Pi OS is a powerful and flexible operating system that is optimized for the Raspberry Pi hardware. It is a great choice for a variety of projects, from building a media center to creating a home automation system.

2.6 Uses of Raspberry Pi [11]

There are so many things you can make with a Raspberry Pi. And due to its extreme popularity, the number of third-party sensors, modules, and code libraries has grown to epic proportions. This has greatly expanded the possible applications for the Raspberry Pi. The applications really are endless, but here are some common ones:

- Web servers
- Local network hubs for IoT
- Sensor control
- Robotics control
- Industrial equipment control
- Automated control systems
- Data processing

2.7 Programming the Raspberry Pi [11]

One of the goals of the creators of the Raspberry Pi was to create a computing platform that would allow people with no background in computer science or programming to learn how to use and program a computer. This is one of the reasons why the Raspberry Pi supports a wide range of programming languages:

- Python
- C/C++
- Java
- HTML5
- Scratch
- JavaScript

- jQuery
- Perl
- Erlang

The Raspberry Pi is an amazing feat of engineering and anyone interested in programming or building electronics projects should consider learning how to use it. There are hundreds of microcontroller and development boards out there, but the Raspberry Pi combines all of those functions on a device the size of a credit card.

CHAPTER 3: SENSORS & ACTUATORS

The V2V Collision Avoidance system relies on a set of sensors and actuators to detect and analyze information about the surrounding vehicles and prevent collisions between them.

These sensors and actuators are installed in the car and work together to provide a detailed picture of the environment around the vehicle.

Among the sensors that can be used in the V2V Collision Avoidance systems are:

3.1 ULTRASONIC SENSOR

3.1.1 Introduction

The Ultrasonic Sensor is an electronic device that calculates distance by emitting sound waves and collecting their echoes. It can measure distances ranging from 2 centimeters to 400 centimeters, about 1 inch to 13 feet within a 30-degree cone. It provides precise measurements to the nearest 0.3 centimeters, which makes it a versatile instrument for correctly measuring both short and long distances without making contact with the target object. The sensor features adjustable pulse widths, allowing for higher resolution measurements when set at lower levels.

The ultrasonic sensor works on the principle of SONAR system which is used to determine the distance to an object. SONAR basically stands for Sound Navigation and Ranging. An ultrasonic sensor generates the high frequency sound (ultrasound) waves. When this ultrasound hits the object, it reflects as

echo which is sensed by the receiver. By measuring the time required for the echo to reach the receiver, we can calculate the distance [12].

3.1.2 About the Module

Ultrasonic module HC-SR04 has an ultrasonic transmitter, receiver, and control circuit [12].

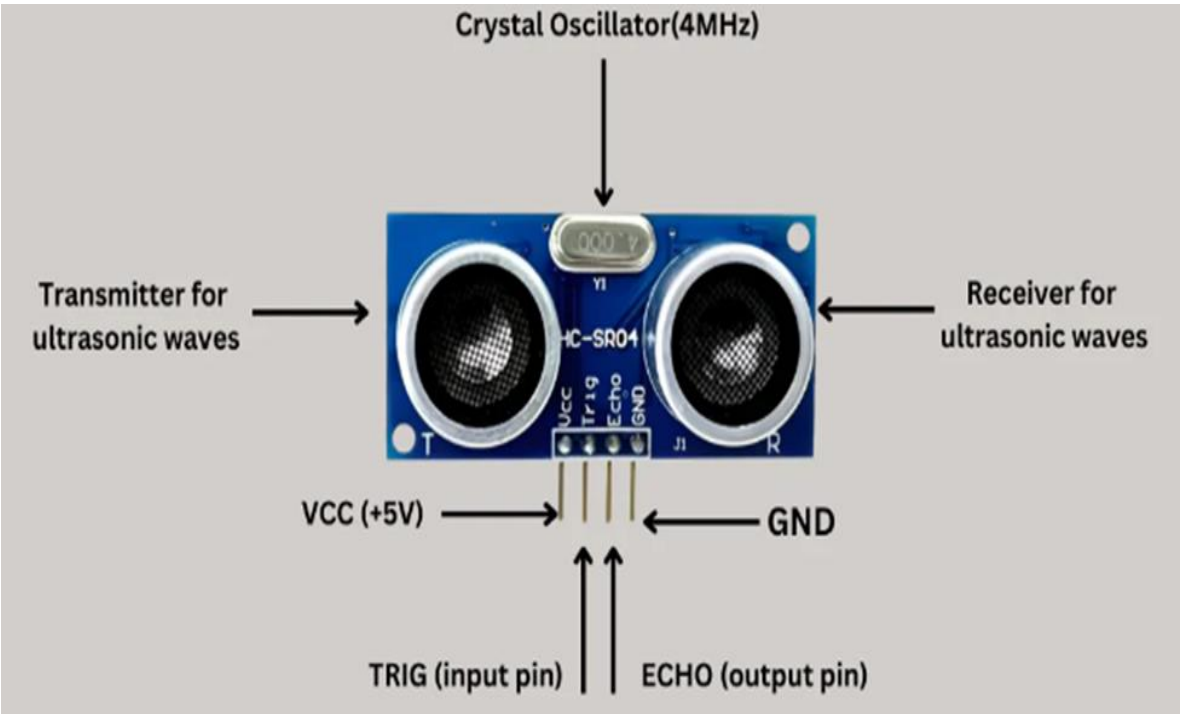


Figure 3.1 HC-SR04 Ultrasonic module

1. Microcontroller U1: The heart of the unit is the EM78P153 8-bit microprocessor [12].

- Interface to Trigger and Echo pins.
- Timing and sending antiphase burst for ping to send.
- Squelch control, whereby during Ultrasonic transmit, a threshold for the incoming receiver is effectively disabled threshold for the incoming receiver is effectively disabled to avoid bogus echoes. This is important as whilst sending vibrations from the TX transducer will be received through PCB and by air between the transducers on the RX transducer.

- Receiving the processed signal from the Receiver section as an interrupt (Rising edge), this is actually a filtered and much amplified version of all the echoes received.

2. Transmitter U3 [12]**:**

- Voltage drive to TX transducer from the antiphase TX signals from the micro (U1). By using antiphase signals, a differential voltage can be driven across the transducer effectively +/-5V across the transducer.
- The other part is two transistors with a common base pin, collectors available on other pins. This transistor forms part of the feedback loop on the final part of the receiver chain, to change an analog signal into a TTL type digital signal.

3. Receiver U2 [12]**:**

- The quad op-amp IC (U2) is a LM324 is 1 MHz unity gain bandwidth device, with limited range I/O. It is a ubiquitous and cheap device.
- Considering some of the gain levels used in the stages means that 40 kHz signals are passing through stages around 10 kHz bandwidth.

Four pins in the ultrasonic sensor:

- Vcc: power supply +5 V
- Gnd: Common ground
- Trigger pin: To start the sensor
- Eco pin: Receive the signal

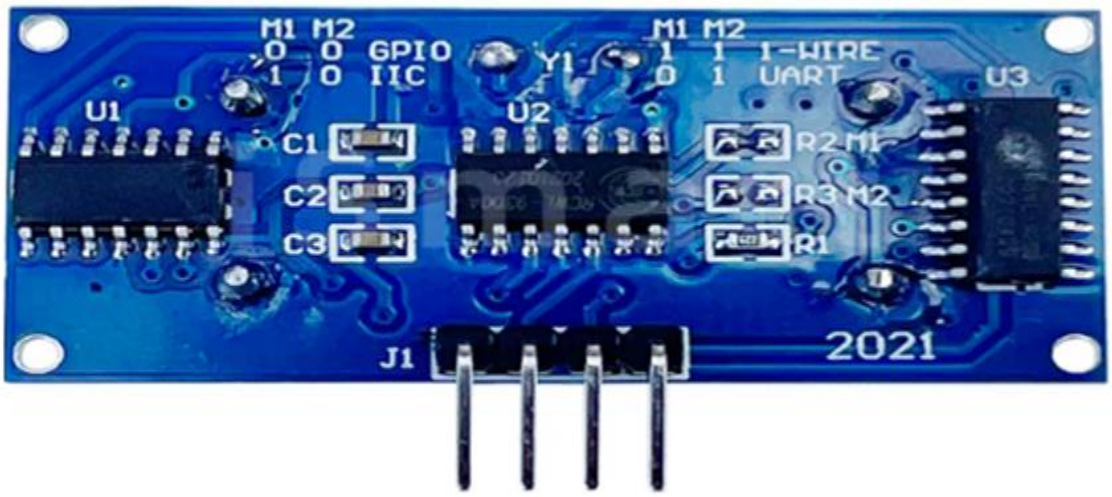


Figure 3.2 HC-SR04 Ultrasonic module

3.1.3 HCSR04 Specifications

These specifications are from the Citron Technologies HCSR04 User's Manual [13]

- Power Supply: +5V DC
- Quiescent Current: <2mA
- Working current: 15mA
- Effectual Angle: <15º
- Ranging Distance: 2-400 cm
- Resolution: 0.3 cm
- Measuring Angle: 30º
- Trigger Input Pulse width: 10uS
- Dimension: 45mm x 20mm x 15mm
- Weight: approx. 10 g

3.1.4 How Ultrasonic Sensors Work?

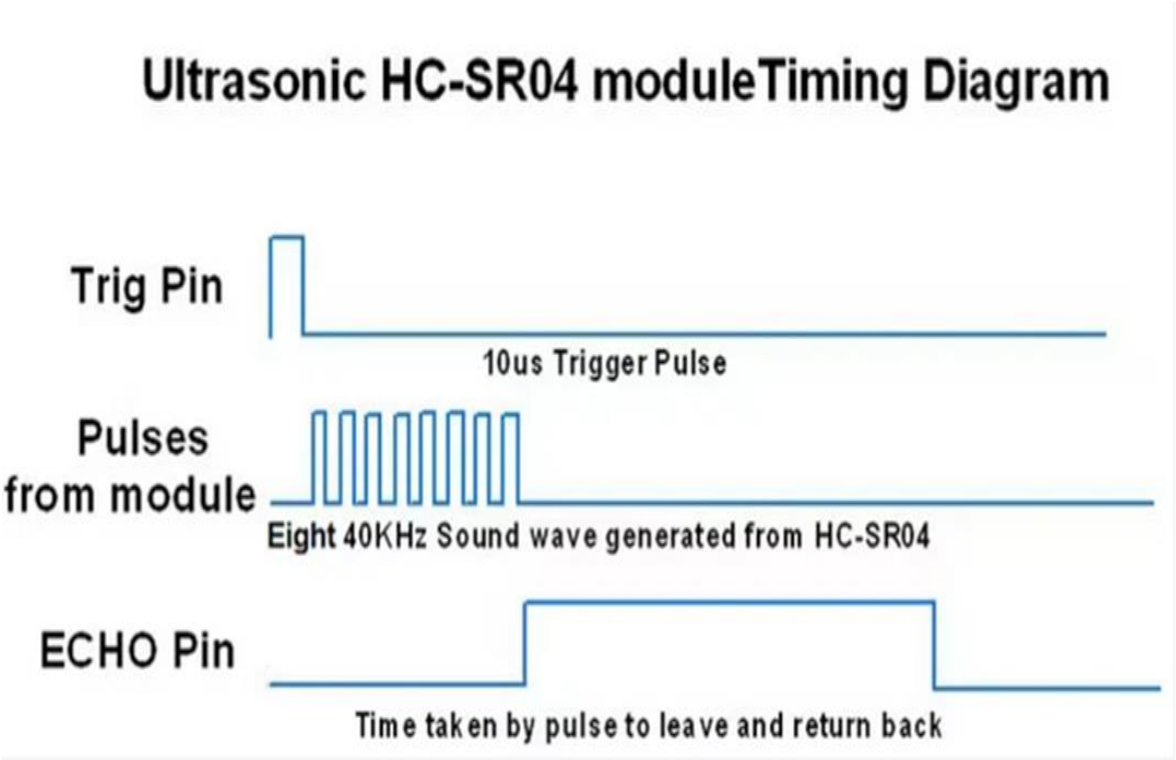


Figure 3.3 HC-SR04 modules Timing Diagram

- We need to transmit trigger pulse of at least 10 us to the HC-SR04 Trig Pin. 10 us is enough periods for the controller, after which it starts to transmit ultrasonic signal [12].
- Then the HC-SR04 automatically sends Eight 40 kHz sound wave pulses and waits for rising edge output at Echo pin [12].
- As the number of pulses, the amplitude of the received signal increases and saturates at a point [12].
- After further increase in number of pulses results in increase in reflection peak, which are not required [12].
- When the rising edge capture occurs at Echo pin, start the Timer and wait for falling edge on Echo pin. As soon as the falling edge is captured at the Echo pin, read the count of the Timer [12].
- This time count is the time required by the sensor to detect an object and return back from an object [12].

Conversion from duration to distance can be done using the following formula:

Distance = (Travel time / 2) x speed of sound

We need to divide the travel time by 2 because we have to take into account that the wave was sent, hit the object, and then returned back to the sensor [12].

If the HC-SR04 does not receive an echo, then the output never goes low. Accordingly, some sensors timeout from 28ms to 36ms [12].

3.1.5 Ultrasonic usage in our project

We have two ultrasonic sensors in the car, located as follows: front and back.

The one located in the front is used to measure the distance between my car and the vehicle in front of me. Based on this distance, the driver is given a warning if the distance is not safe, or advised to reduce speed to avoid a collision.

The one located on the back is used to measure the distance when parking or reversing. If an object is detected to be too close to the car, it will trigger a stop to prevent a collision. Otherwise, a notification about the distance will be displayed to the driver.

3.2 BUZZER [14]

3.2.1 Introduction

- An audio signaling device like a beeper or buzzer may be electromechanical or piezoelectric or mechanical type.
- The main function of this is to convert the signal from audio to sound.
- Generally, it is powered through DC voltage and used in timers, alarm devices, printers, alarms, computers, etc.
- Based on the various designs, it can generate different sounds like alarm, music, bell & siren.

3.2.2 Buzzer Pin Configuration

The pin configuration of the buzzer is shown below. It includes two pins namely positive and negative. The positive terminal of this is represented with the '+' symbol or a longer terminal. This terminal is powered through 6Volts whereas the negative terminal is represented with the '-' symbol or short terminal and it is connected to the GND terminal.

Figure 3.4 BUZZER

3.2.3 Specifications

- The frequency range is 3,300Hz.
- Operating Temperature ranges from – 20° C to +60°C.
- Operating voltage ranges from 3V to 24V DC.
- The sound pressure level is 85dBA or 10cm.
- The supply current is below 15mA.

3.2.4 Working principle

The working principle of a buzzer is based on the piezoelectric effect.

- A piezoelectric material is one that generates an electric charge in response to mechanical stress and produces mechanical stress in response to an electric field. Buzzer sensors typically use a piezoelectric ceramic element as the sound-producing element.

- When a voltage is applied across the piezoelectric element, it causes the element to vibrate at a high frequency, typically in the range of several kilohertz to several megahertz.

- These vibrations cause the air molecules around the element to vibrate, producing sound waves that propagate through the air.

- The frequency of the sound produced by the buzzer is determined by the frequency of the voltage applied to the piezoelectric element.

- The intensity of the sound, or the volume, is determined by the amplitude of the voltage.

- Buzzer sensors can be designed to produce different types of sounds, including continuous tones, intermittent tones, and complex tone patterns.

- The specific sound produced by the buzzer is determined by the design of the piezoelectric element and the electrical circuit driving it.

Overall, the working principle of a buzzer is based on the piezoelectric effect and the vibration of a piezoelectric element to produce sound waves
in the surrounding air.

3.3 DC motor

3.3.1 Introduction

A DC motor is an electrical motor that uses direct current (DC) to produce mechanical force. The most common types rely on magnetic forces produced by currents in the coils. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current in part of the motor.

DC motors were the first form of motors widely used, as they could be powered from existing direct-current lighting power distribution systems. A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings. Small DC motors are used in tools, toys, and appliances. The universal motor, a lightweight brushed motor used for portable power tools and appliances can operate on direct current and alternating current. Larger DC motors are currently used in propulsion of electric vehicles, elevator and hoists, and in drives for steel rolling mills. The advent of power electronics has made replacement of DC motors with AC motors possible in many applications [15].

Figure 3.5 DC Motor

3.3.2 Basic Principles and Operation

At the heart of a DC motor's operation is the interaction between magnetic fields and current-carrying conductors. The basic principle involves generating rotational motion through the Lorentz force, which acts on a conductor carrying current within a magnetic field.

- **Components of a DC Motor:**

- Stator: The stationary part of the motor, which produces a magnetic field.
- Rotor (Armature): The rotating part that carries the current and interacts with the stator's magnetic field.
- Commutator: A rotary switch in brushed DC motors that reverses the direction of current through the rotor windings.
- Brushes: Conductive materials that maintain an electrical connection between the stationary and rotating parts of the motor.

When current flows through the rotor windings, a magnetic field is created around the rotor. This field interacts with the stator's magnetic field, producing a torque that causes the rotor to turn. The commutator periodically reverses the current direction, ensuring continuous rotation [16].

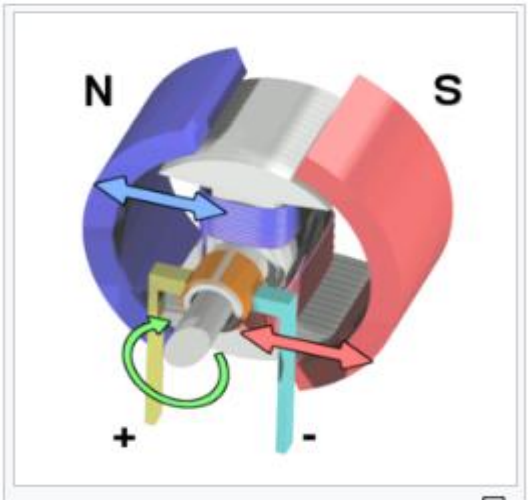


Figure 3.6 Structure of DC Motor

3.3.3 Types of DC Motors [17]

- **Brushed DC Motors**

Brushed DC motors are the most traditional type of DC motor, featuring a simple design where the commutator and brushes manage the current direction.

- **Structure and Function:** Consists of a rotor with windings, a commutator, and brushes that make contact with the commutator segments.
- **Advantages**: Simple control, low cost, and easy to maintain.
- **Disadvantages**: Brush wear and tear, frequent maintenance, and limited speed control precision.

- **Brushless DC Motors**

Brushless DC motors (BLDC) eliminate the commutator and brushes, using electronic controllers to manage current flow.

- **Structure and Function:** Comprises a rotor with permanent magnets and a stator with windings, controlled by an electronic controller.
- **Advantages:** Higher efficiency, less maintenance, and better speed and position control.
- **Disadvantages**: Higher initial cost and more complex control system.

- **Applications**
  - **Brushed DC Motors:** Suitable for applications where simplicity and cost-effectiveness are crucial, such as in toys, small appliances, and some power tools.

- **Brushless DC Motors**: Ideal for applications requiring high efficiency and precise control, including drones, electric vehicles, and industrial robots.

3.3.4 Detailed Components and Design [18]

- **Stator Design and Materials**

The stator is typically made of laminated steel to minimize eddy current losses, with windings made from copper or aluminum.

- **Rotor Design and Materials**

The rotor is constructed from laminated steel and wound with copper wire, designed to handle high rotational speeds and torque.

- **Commutator and Brushes**

In brushed motors, the commutator is made of segmented copper rings, and brushes are often made of carbon or graphite, chosen for their conductivity and wear resistance.

- **Electronic Controllers**

In brushless motors, electronic controllers manage the current flow through the stator windings, using sensors to determine the rotor's position and adjust the current accordingly.

3.3.5 Performance Characteristics [19]

- **Torque-Speed Characteristics**

DC motors exhibit a linear relationship between torque and speed, making them suitable for applications requiring a wide range of speeds and consistent torque.

- **Efficiency Factors**

Efficiency is influenced by factors such as winding resistance, friction, and magnetic losses. Brushless motors generally offer higher efficiency due to reduced friction and more precise control.

- **Common Performance Metrics**

Key performance metrics for DC motors include torque, speed, power output, and efficiency. These metrics help determine the suitability of a motor for specific applications.

3.4 Servo motor [20]

3.4.1 Introduction

A servo motor is defined as an electric motor that allows for precise control of angular or linear position, speed, and torque. It consists of a suitable motor coupled to a sensor for position feedback and a controller that regulates the motor's movement according to a desired set point.

Servo motors are essential in industries like robotics, CNC machinery, and automated manufacturing due to their precision, quick responsiveness, and fluid motion.

However, modern servo motors are capable of providing high performance and precision as main drives in various applications.

-A servo motor consists of three main components:

- **A motor**: This can be either a DC motor or an AC motor depending on the power source and the application requirements. The motor provides the mechanical power to rotate or move the output shaft.
- **A sensor**: This can be either a potentiometer, an encoder, are solver, or another device that measures the position, speed, or torque of the output shaft and sends feedback signals to the controller.

- **A controller**: This can be either an analog or a digital circuit that compares the feedback signals from the sensor with the desired set point signals from an external source (such as a computer or joystick) and generates control signals to adjust the motor's voltage or current accordingly.

The controller employs a closed-loop feedback system, adjusting the motor's movement to closely align with the desired set point, maintaining strict accuracy.

The controller can also implement various control algorithms, such as proportional-integral-derivative (PID) control, fuzzy logic control, adaptive control, etc., to optimize the performance of the servo motor.

3.4.2 Servo Motor Work:

The basic working principle of a servo motor involves the controller receiving two types of input signals:

Figure 3.7 Servo Motor

A set point signal: This is an analog or digital signal that represents the desired position, speed, or torque of the output shaft. A feedback signal: This is an analog or digital signal that represents the actual position, speed, or torque of the output shaft measured by the sensor.

The controller compares these two signals and calculates an error signal that represents the difference between them.

The error signal is then processed by a control algorithm (such as PID) that generates a control signal that determines how much voltage or current should be applied to the motor.

The control signal is sent to a power amplifier (such as an H-bridge) that converts it into an appropriate voltage or current level for driving the motor.

The motor then rotates or moves according to the control signal and changes its position, speed, or torque, and sends a new feedback signal to the controller.

The process repeats until the error signal becomes zero or negligible, indicating that the output shaft has reached the desired set point.

3.4.3 Types of Servo Motors [21]:

Servo motors can be classified into different types based on their power source, construction, feedback mechanism, and application.

- **AC Servo Motors**: AC servo motors are electric motors that operate on alternating current (AC). They have a stator that generates a rotating magnetic field and rotor that follows the field.
  AC servo motors, powered by alternating current, feature a stator that creates a rotating magnetic field, with a rotor that synchronizes with this field for efficient operation.

-AC servo motors can be further divided into two types: synchronous and asynchronous:

- **Synchronous AC servo motors:** have a permanent magnet rotor that rotates at the same speed as the stator field. They are more efficient,

precise, and responsive than asynchronous motors, but they require a more complex controller and a position sensor.

- **Asynchronous AC servo motors:** have a wound rotor that induces current and a magnetic field that lags behind the stator field. They are simpler, cheaper, and more rugged than synchronous motors, but they have lower efficiency, accuracy, and speed.

AC servo motors are suitable for high-power applications that require high speed, torque, and reliability. They are commonly used in industrial machines, robotics, CNC machines, etc.

- **DC Servo Motors:** DC servo motors are electric motors that operate on direct current (DC). They have a permanent magnet stator that generates a fixed magnetic field and a wound rotor that rotates when a current is applied.

-DC servo motors can be further divided into two types: brushed and brush less:

- **Brushed DC servo motors:** have a commutator and brushes that switch the current direction in the rotor windings. They are simple, inexpensive, and easy to control, but they have lower efficiency, lifespan, and speed due to friction and wear of the brushes.
- **Brush less DC servo motors:** have an electronic controller that switches the current direction in the stator windings. They are more efficient, durable, and fast than brushed motors, but they require a more sophisticated controller and a position sensor.

DC servo motors are suitable for low-power applications that require high precision, responsiveness, and smooth motion. They are commonly used in hobby projects, toy cars, CD/DVD players, etc.

- **Linear Servo Motors**: Linear servo motors are electric motors that produce linear motion instead of rotary motion. They have a stationary part called a forces or primary that contains coils or magnets, and a moving part called a platen or secondary that contains magnets or iron cores.

-Linear servo motors can be further divided into two types: iron-core and iron less:

- Iron-core linear servo motors have iron cores in the platen that interact with the magnetic field of the forces. They have high force density, stiffness, and accuracy, but they also have high cogging force, weight, and heat generation.
- Iron less linear servo motors have no iron cores in the platen, only magnets. They have low clogging force, weight, and heat generation, but they also have low force density, stiffness, and accuracy.

Linear servo motors are suitable for applications that require high speed, acceleration, and precision over long distances. They are commonly used in semiconductor manufacturing, meteorology, laser cutting, etc.

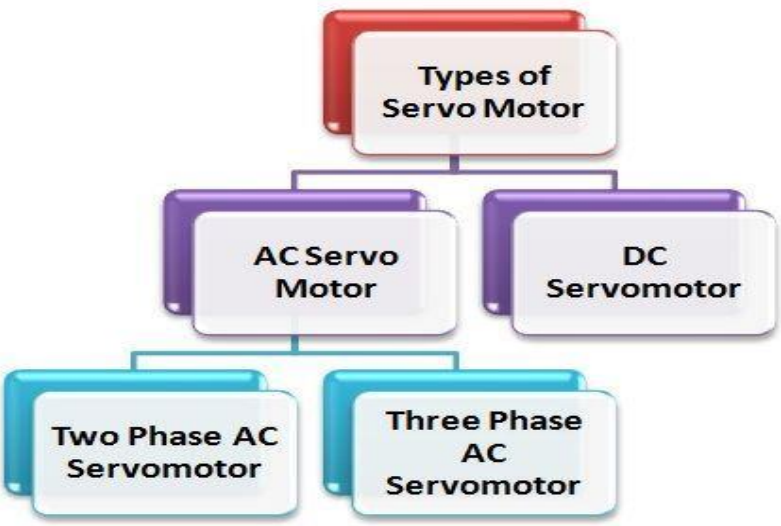


Figure 3.8 Types of Servo Motors

3.4.4 How we Control a Servo Motor:

The control of a servo motor depends on the type of motor, the feedback mechanism, and the application requirements.

Generally, there are two types of control signals that can be used to control a servo motor: analog and digital.

- **Analog control signals**: are continuous voltage or current signals that vary proportionally to the desired set point. They are typically used for simple or low-cost servo systems that do not require high accuracy or resolution. For example, a potentiometer can be used to generate an analog control signal for a hobby servo motor.

- **Digital control signals:** are discrete pulses or bits that represent the desired set point in a coded form. They are typically used for complex or high-performance servo systems that require high accuracy, resolution, or communication. For example, a pulse- width modulation (PWM) signal can be used to generate a digital control signal for a brush-less DC servo motor.

The controller of a servo motor can be either an external device or an integrated circuit within the motor. The controller receives the control signals from an external source (such as a computer or a joystick), and the feedback signals from the sensor and generates the appropriate control signals for driving the motor.

The controller can also implement various control algorithms to optimize the performance of the servo motor. Some of the common control algorithms are:

- **Proportional-integral-derivative (PID) control:** This is a feedback- based control algorithm that adjusts the control signal based on the proportional, integral, and derivative terms of the error signal. It is widely used for servo systems that require a fast and accurate response.
- **Fuzzy logic control:** This is a rule-based control algorithm that adjusts the control signal based on fuzzy sets and linguistic variables. It is useful for servo systems that deal with uncertainty or non-literariness.

- **Adaptive control:** This is a self-tuning control algorithm that adjusts the control parameters based on the changing conditions of the servo system. It is beneficial for servo systems that face disturbances or variations

3.4.5 Applications of Servo Motors:

Servo motors have a wide range of applications in various fields and industries. Some of the common applications are:

- Robotics: Servo motors are used to provide precise motion and force for robotic arms, legs, joints, grippers, etc. They enable robots to perform tasks such as picking, placing, welding, assembling, etc.
- CNC machinery: Servo motors are used to drive the axes of CNC machines such as lathes, mills, routers, etc. They enable CNC machines to perform accurate and complex machining operations such as cutting, drilling, engraving, etc.
- Automated manufacturing: Servo motors are used to control the movement and position of various components and devices in automated manufacturing systems, such as conveyors, feeders, loaders, un loaders, etc. They enable automated manufacturing systems to achieve high productivity and quality.
- Medical equipment: Servo motors are used to operate various medical devices and instruments such as surgical robots, scanners, pumps, ventilators, etc. They enable medical equipment to perform precise and safe operations and treatments.

3.4.6 Conclusion

In this article, we have learned about the definition, working principle, types, control, and applications of servo motors.

We have seen that servo motors are electric motors that allow for precise control of angular or linear position, speed, and torque. They consist of a motor, a sensor, and a controller that forma closed-loop feedback system.

We have also seen that servo motors can be classified into different types based on their power source, construction, feedback mechanism, and application. Some of the common types are AC servo motors, DC servo motors, and linear servo motors.

We have also seen that servo motors can be controlled by either analog or digital signals that represent the desired set point. The controller can also implement various control algorithms to optimize the performance of the servo motor.

We have also seen that servo motors have a wide range of applications in various fields and industries, such as robotics, CNC machinery, automated manufacturing, medical equipment, etc.

3.5 H-bridge

3.5.1 Introduction

The H-bridge is a fundamental circuit in electrical engineering, widely used for controlling the direction of a DC motor. It allows for the application of voltage across a load in either direction, making it essential for various applications such as robotics, automotive systems, and industrial machinery. The H-bridge's ability to control motor direction and speed efficiently makes it a critical component in modern electronic design. This section provides a brief overview of the H-bridge, its importance, and its key applications [22].

3.5.2 Basic Principles

An H-bridge is a type of electrical circuit that enables a voltage to be applied across a load in either direction. This circuit is named for its distinctive 'H' shape, formed by four switching elements, usually transistors or MOSFETs, arranged in a configuration that allows for the control of current flow through a load, such as a DC motor [23].

3.5.3 Components of an H-Bridge

- **Switching Elements (Transistors/MOSFETs):**

  The primary components of an H-bridge are four switches, which can be implemented using bipolar junction transistors (BJTs), metal-oxide-semiconductor field-effect transistors (MOSFETs), or even relays.

  These switches are labeled as Q1, Q2, Q3, and Q4, with Q1 and Q3 on one side of the load and Q2 and Q4 on the other.

- **Diodes:**

  Diodes are often used in H-bridge circuits to protect against back electromotive force (EMF) generated by inductive loads like motors. These are known as flyback diodes.

  They ensure that the current flows in the correct direction and help in managing transient voltage spikes.

- **Control Logic:**

  The operation of the H-bridge is controlled by a logic circuit that determines which switches are turned on or off at any given time.

  This logic can be implemented using microcontrollers, dedicated motor driver Ics, or discrete logic gates.

The arrangement of these components in an h-bridge allows for the efficient control of the load's direction and in more advanced designs, the speed of the load [24].

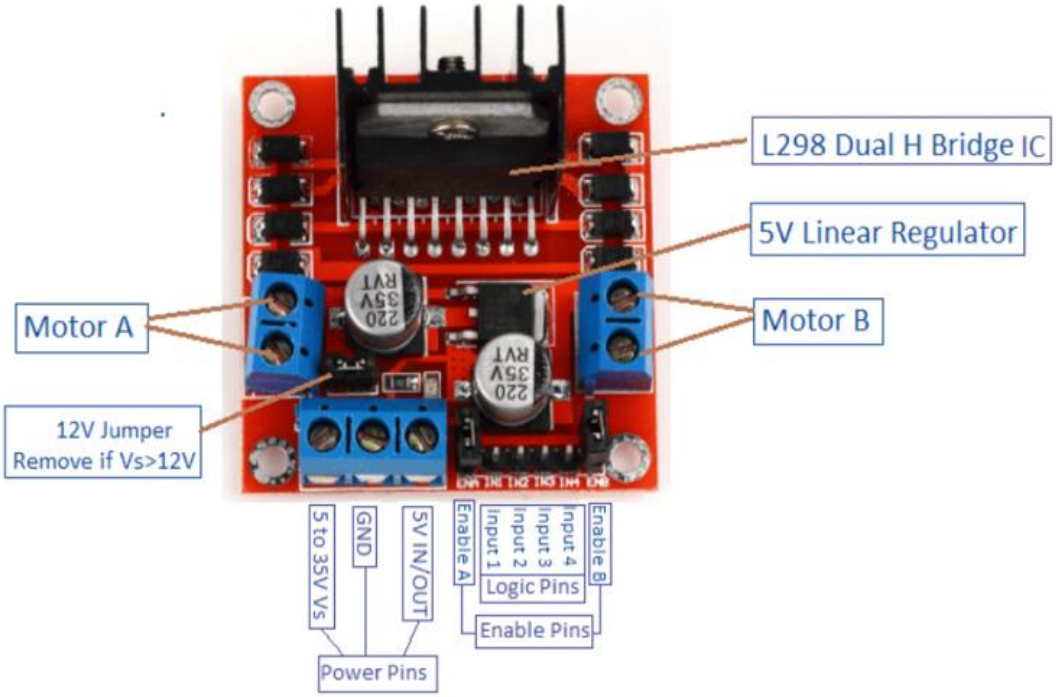


Figure 3.9 L298N Dual H-Bridge

3.5.4 Operation [25]

The operation of an H-bridge involves switching the transistors in various combinations to control the direction and flow of current through the load. The four basic states of an H-bridge are:

Forward Operation:

In forward operation, switches Q1 and Q4 are closed (turned on), while Q2 and Q3 are open (turned off). This allows current to flow from the power supply, through Q1, the load, Q4, and then to ground.

This configuration drives the motor in one direction.

Reverse Operation:

In reverse operation, switches Q2 and Q3 are closed, while Q1 and Q4 are open. This causes current to flow in the opposite direction through the load.

This reverses the motor's direction.

Braking Mode:

To brake the motor, both Q1 and Q2, or Q3 and Q4, can be closed simultaneously. This short-circuits the motor's terminals, causing it to stop quickly due to the back EMF.

High-Impedance State:

In the high-impedance state, all switches are open, and no current flows through the motor. This effectively disconnects the motor from the power supply, allowing it to freewheel.

The control logic must ensure that no two switches on the same side of the load (e.g., Q1 and Q2) are closed simultaneously, as this would create a short circuit directly across the power supply.

Module B

Software Architecture

Chapter 4: Universal Asynchronous Receiver Transmitter (UART).

Chapter 5: Bluetooth.

CHAPTER 4: UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER

Universal synchronous receiver transmitter (USART) /universal asynchronous receiver transmitter (UART).

4.1 USART Introduction

USART Is a hardware circuit which provides serial communication between devices, it has two modes: Asynchronous and Synchronous serial communication, Synchronous is the faster mode with a clock signal.

USART offers a very wide range of baud rates using a fractional baud rate generator. It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication. High speed data communication is possible by using the DMA for multibuffered configuration.

4.1.1 Asynchronous and synchronous serial communication [26]

A UART device can use asynchronous communication protocols. A USART device can use both asynchronous and synchronous communication protocols. Therefore, a USART can do anything a UART can do and more. A USART has more complex circuitry and requires more lines for communication, many devices may only implement a UART to save on cost, complexity or power usage.

Asynchronous and synchronous are two different ways to standardize how serial data is sent.

- **Asynchronous serial data**

In asynchronous mode, only one data line is used to send data from the transmitter to the receiver. There is no shared clock signal from the master to the slave. So, the receiver has no way to know how fast or slow the data is coming. To circumvent this, both the sender and receiver must be manually configured beforehand to use the same data rate. A common shared baud rate is 9,600 bits per second.

Even sharing a common bit rate, it is possible for the devices to become slightly out of sync. Therefore, extra bits are added to each packet of data to ensure reliable transmission. A start and stop bit are almost always added to indicate each packet. Often, a parity bit is also added to indicate small errors. Adding these extra bits causes asynchronous communication to be less efficient.

- **Synchronous serial data**

In synchronous mode, to send the data there is two separated lines for data and a clock. The controller sends a clock signal, which synchronizes the controller and the peripheral at the same data rate. Because the clock signal keeps the devices in sync, the two devices don’t need to be configured ahead of time to use the same bit rate.

There is no need for extra start and stop bits when using a clock signal. All the data can be sent continuously without pause. It also enables much faster bit rates as the sender doesn’t need to worry about the receiver being able to keep up or get out of sync.

The following are some example synchronous protocols:

- Serial peripheral interface
- Inter-Integrated Circuit
- Controller Area Network

4.2 USART main features [27]

- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- Fractional baud rate generator systems
  - Common A common programmable transmit and receive baud rates up to 4.5 Mbits/s
- Programmable data word length (8 or 9 bits)
- Configurable stop bits – support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability
  - 13-bit break generation and 10/11-bit break detection when USART is hardware configured for LIN
- Transmitter clock output for synchronous transmission IrDA SIR encoder decoder
  - Support for 3/16-bit duration for normal mode
- Smartcard emulation capability
  - The Smartcard interface supports the asynchronous protocol Smartcards as defined in the ISO 7816-3 standards
  - 0.5, 1.5 stop bits for Smartcard operation
- Single-wire half-duplex communication
- Configurable multibuffered communication using DMA (direct memory access)
  - Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
- Separate enable bits for transmitter and receiver
- Transfer detection flags:
  - Receive buffer full
  - Transmit buffer empty
  - End of transmission flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- Four error detection flags:

  - Overrun error
  - Noise detection
  - Frame error
  - Parity error
- Ten interrupt sources with flags:
  - CTS changes
  - LIN break detection
  - Transmit data register empty
  - Transmission complete
  - Receive data register full
  - Idle line received
  - Overrun error
  - Framing error
  - Noise error
  - Parity error
- Multiprocessor communication – enter into mute mode if address match does not occur
- Wake up from mute mode (by idle line detection or address mark detection)
- Two receiver wakeup modes: Address bit (MSB, 9th bit), Idle line

4.3 USART functional description

The interface is externally connected to another device by three pins (see Figure 4.1) [3]. Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at a high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data (at USART level, data are then received on SW_RX).

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5,1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator – with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)
- Data Register (USART_DR)
- A baud rate register (USART_BRR) – 12-bit mantissa and 4-bit fraction.
- A guard-time Register (USART_GTPR) in case of Smartcard mode.

Refer to Section 25.6: USART registers for the definition of each bit.

The following pin is required to interface in synchronous mode:

- CK: Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel data can be received synchronously on RX. This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable. In smartcard mode, CK can provide the clock to the smartcard.

The following pins are required for Hardware flow control mode:

- CTS: Clear To Send blocks the data transmission at the end of the current transfer when high
- RTS: Request to send indicates that the USART is ready to receive a data (when low).

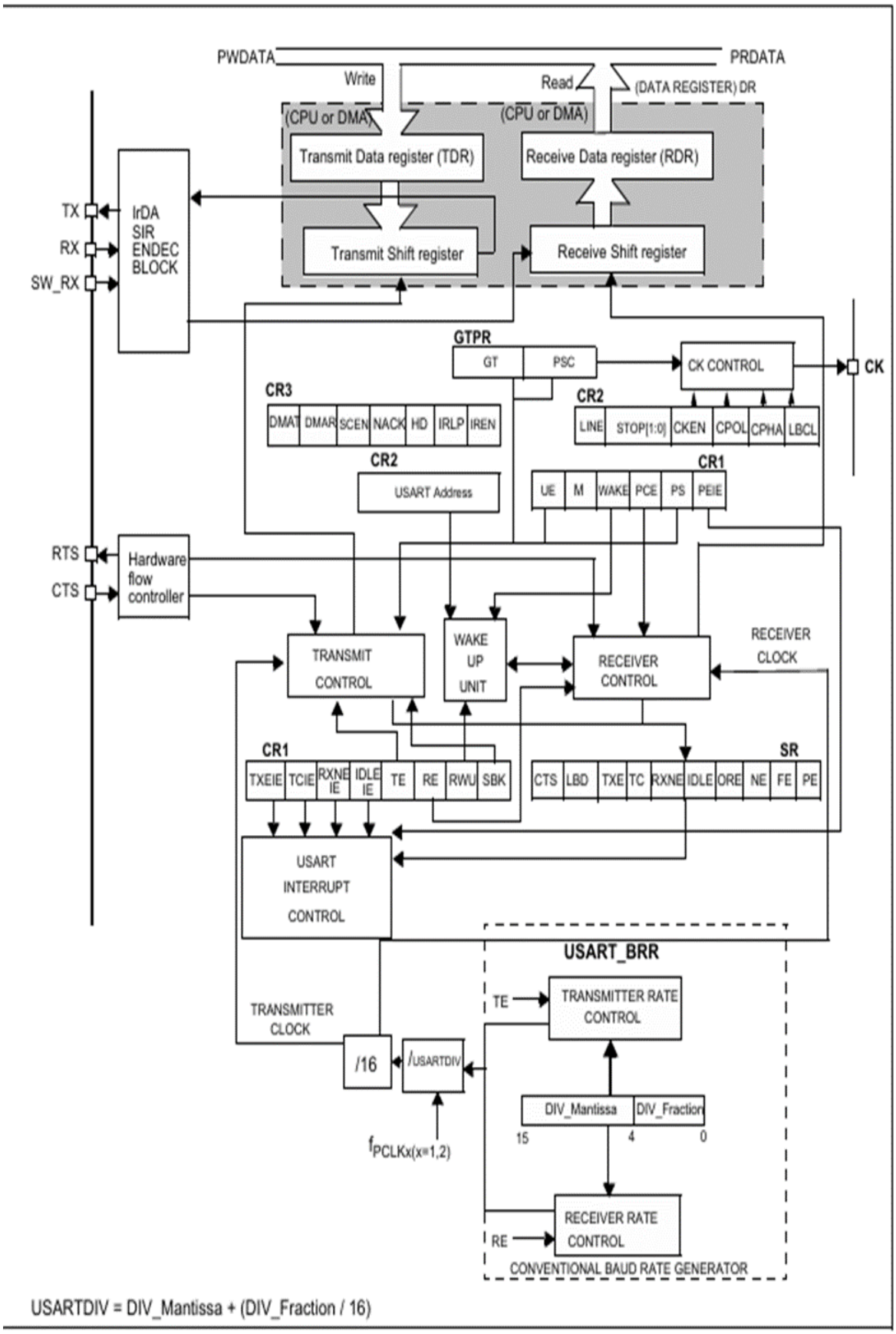


Figure 4.1 USART block diagram

4.3.1 USART character description

Word length can be either 8 or 9 bits it is selected by programming the M bit in the USART_CR1 register (see Figure 4.2).

During the start bit the TX pin is in low state. During the stop bit it is in high state.

An Idle character is interpreted as an entire frame of “1” ‘s followed by the start bit of the next frame that contains data (The number of “1” ‘s will include the number of stop bits).

A Break character is interpreted on receiving “0” s for a frame period. At the end of the break frame the transmitter inserts either 1 or 2 stop bits (logic “1” bit) to acknowledge the start bit.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver. The details of each block are given below.

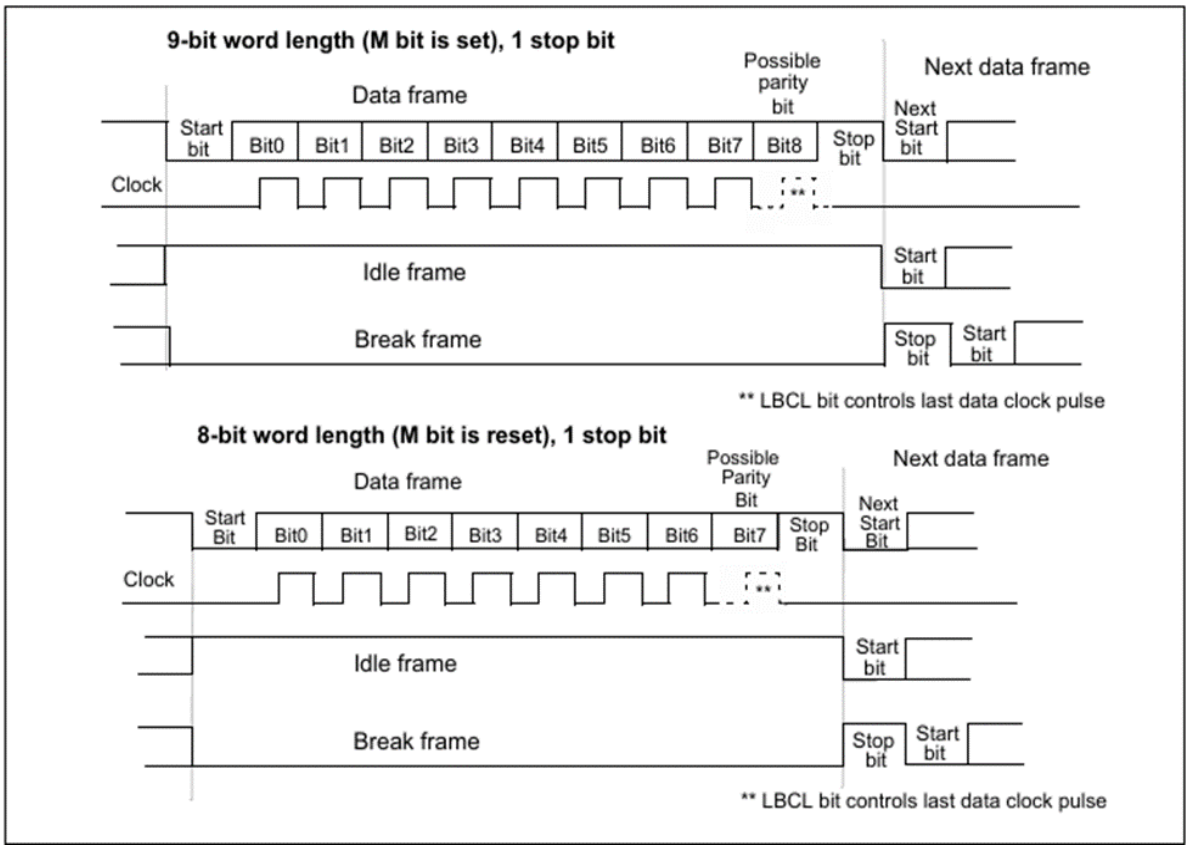


Figure 4.2 Word length programming

4.3.2 Transmitter

When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

The transmitter can send data words of either 8 or 9 bits depending on the M bit status.

Character transmission

Every character is preceded by a start bit that is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.

During an USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see Figure 4.1).

Note: Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost. So, the TE bit should not be reset during transmission of data.

An idle frame will be sent after the TE bit is enabled.

Configurable stop bit:

The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

- **1 stop bit:** This is the default value of the number of stop bits.
- **2 Stop bits:** This is supported by normal USART, single-wire and modem modes.
- **0.5 stop bit:** To be used when receiving data in Smartcard mode.
- **1.5 stop bits:** To be used when transmitting and receiving data in Smartcard mode.

An idle frame transmission will include the stop bits.

A break transmission will be 10 low bits followed by the configured number of stop bits (when m = 0) and 11 low bits followed by the configured number of stop bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits) (see Figure 4.3).

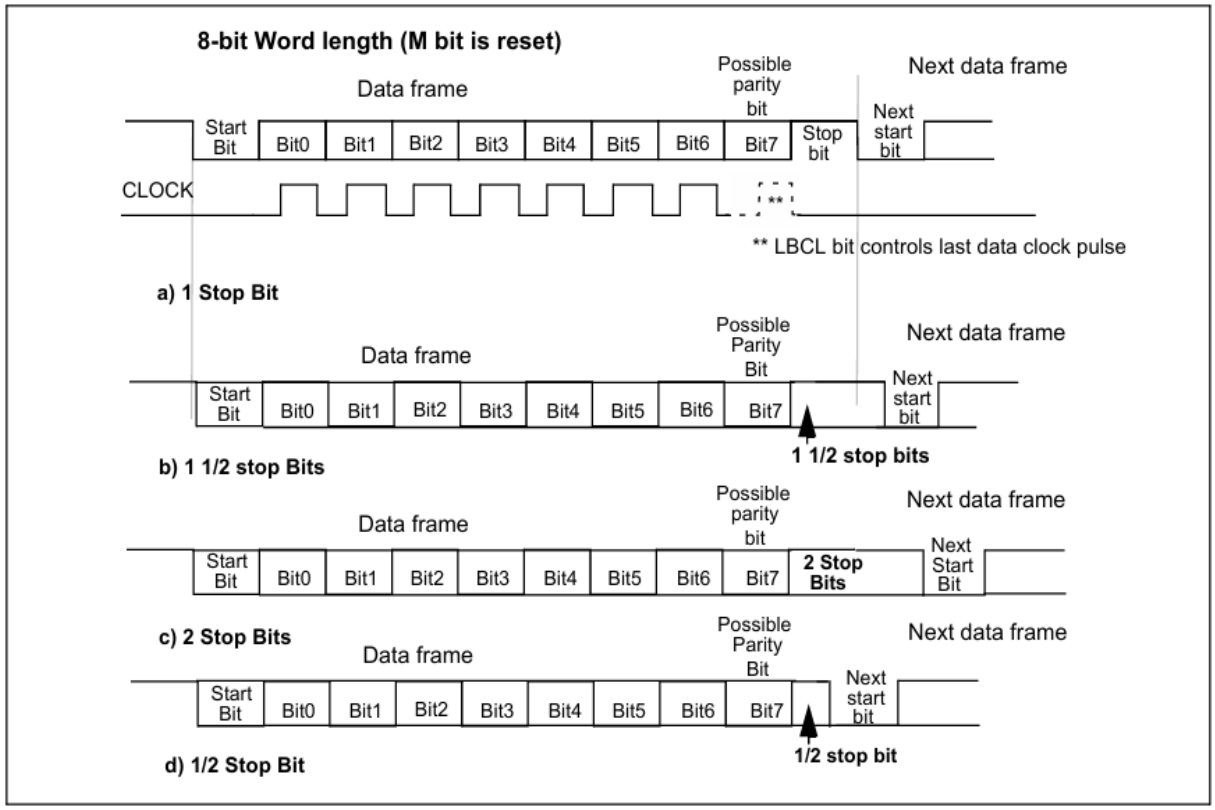


Figure 4.3 Configurable stop bits

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffered communication.
5. Select the desired baud rate using the USART_BRR register.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.

7. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART_DR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

Single byte communication

By writing to the data register it is always clearing the TXE bit.

The TXE bit is set by hardware, it indicates:

- The data has been moved from TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART_DR register without overwriting the previous data.

If the TXEIE bit is set this flag will generate an interrupt.

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register and which is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

After the stop bit the frame is transmitted and the TXE bit is set, the TC bit goes high. An interruption is generated if the TCIE bit is set in the USART_CR1 register.

After writing the last data into the USART_DR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low power mode (see Figure 4.4).

The TC bit can be cleared by the following software sequence:

1. A read from the USART_SR register
2. A write to the USART_DR register

NOTE: The TC bit can also be cleared by writing a ‘0 to it. This clearing sequence is recommended only for Multibuffered communication.

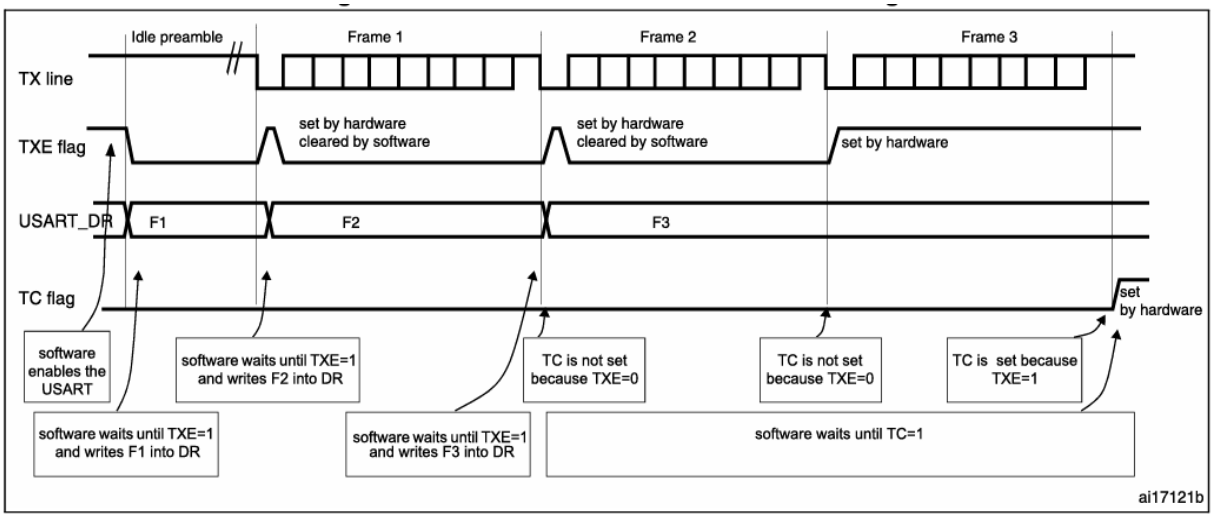


Figure 4.4 TC/TXE behavior when transmitting

Break characters

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see Figure 4.3).

If the SBK bit is set to ‘1 a break character is sent on the TX line after completing the current character transmission. This bit is reset by hardware when the break character is completed (during the stop bit of the break character). The USART inserts a logic 1 bit at the end of the last break frame to guarantee the recognition of the start bit of the next frame.

Note: If the software resets the SBK bit before the commencement of break transmission, the break character will not be transmitted. For two consecutive breaks, the SBK bit should be set after the stop bit of the previous break.

Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

4.3.3 Receiver

USART receives data words of either 8 or 9 bits depending on the M bit in the USART_CR1 register.

Start bit detection

In the USART, the oversampling is 16, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0 X 0 0 0 0 (See Figure 4.5).

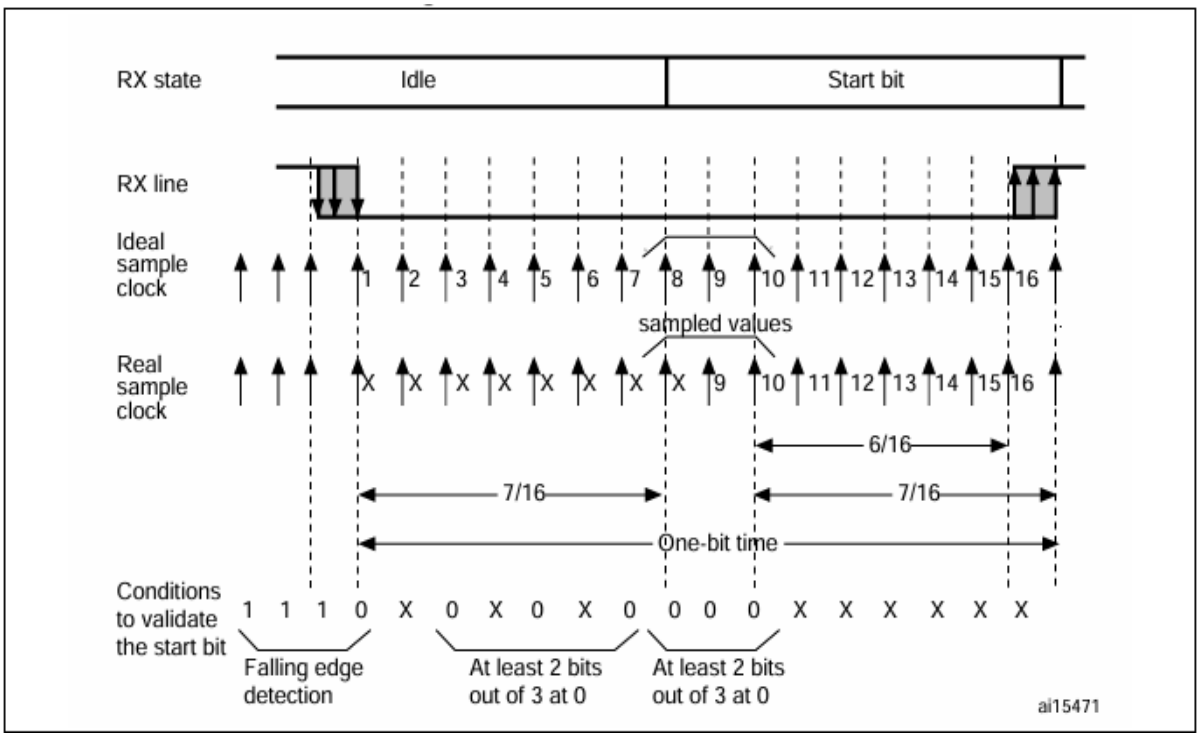


Figure 4.5 Start bit detection

Note: If the sequence is not complete, the start bit detection aborts and the receiver return to the idle state (no flag is set) where it waits for a falling edge.

The start bit is confirmed (RXNE flag set, interrupt generated if RXNEIE=1) if the 3 sampled bits are at 0 (first sampling on the 3rd, 5th and 7th bits finds the 3 bits at 0 and second sampling on the 8th, 9th and 10th bits also finds the 3 bits at 0). The start bit is validated (RXNE flag set, interrupt generated if RXNEIE=1) but the NE noise flag is set if, for both samplings, at least 2 out of the 3 sampled bits are at 0 (sampling on the 3rd, 5th and 7th bits and sampling on the 8th, 9th and 10th

bits). If this condition is not met, the start detection aborts and the receiver returns to the idle state (no flag is set).

If, for one of the samplings (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), 2 out of the 3 bits are found at 0, the start bit is validated but the NE noise flag bit is set.

Character reception

Data shifts in least significant bit first through the RX pin, during an USART reception. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAR) in USART_CR3 if multibuffered communication is to take place.
5. Select the desired baud rate using the baud rate register USART_BRR.
6. Set the RE bit USART_CR1. This enables the receiver that begins searching for a start bit.

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffered, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Note: If the RE bit is disabled during reception, the reception of the current byte will be aborted. So, The RE bit should not be reset while receiving data.

Break character

When a break character is received, the USART handles it as a framing error.

Idle character

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the IDLEIE bit is set.

Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data cannot be transferred from the shift register to the RDR register until the RXNE bit is cleared

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced.

When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART_DR is performed.
- The shift register will be overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or both the EIE and DMAR bits are set.
- The ORE bit is reset by a read to the USART_SR register followed by a USART_DR register read operation.

Note: When the ORE bit is set, it indicates that at least 1 data has been lost. There are two possibilities:

- if RXNE=1, then the last valid data is stored in the receive register RDR and can be read.

- if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received. It may also occur when the new data is received during the reading sequence (between the USART_SR register read access and the USART_DR read access).

Noise error

receiver implements oversampling by 16 for data recovery by discriminating between valid incoming data and noise.

Over-sampling techniques are used (except in synchronous mode), Oversampling by 16 is 16 or 8 times the baud rate clock (See Figure 4.6).

This increases the tolerance of the receiver to clock deviations. The maximum speed is limited to maximum FPCLK/16

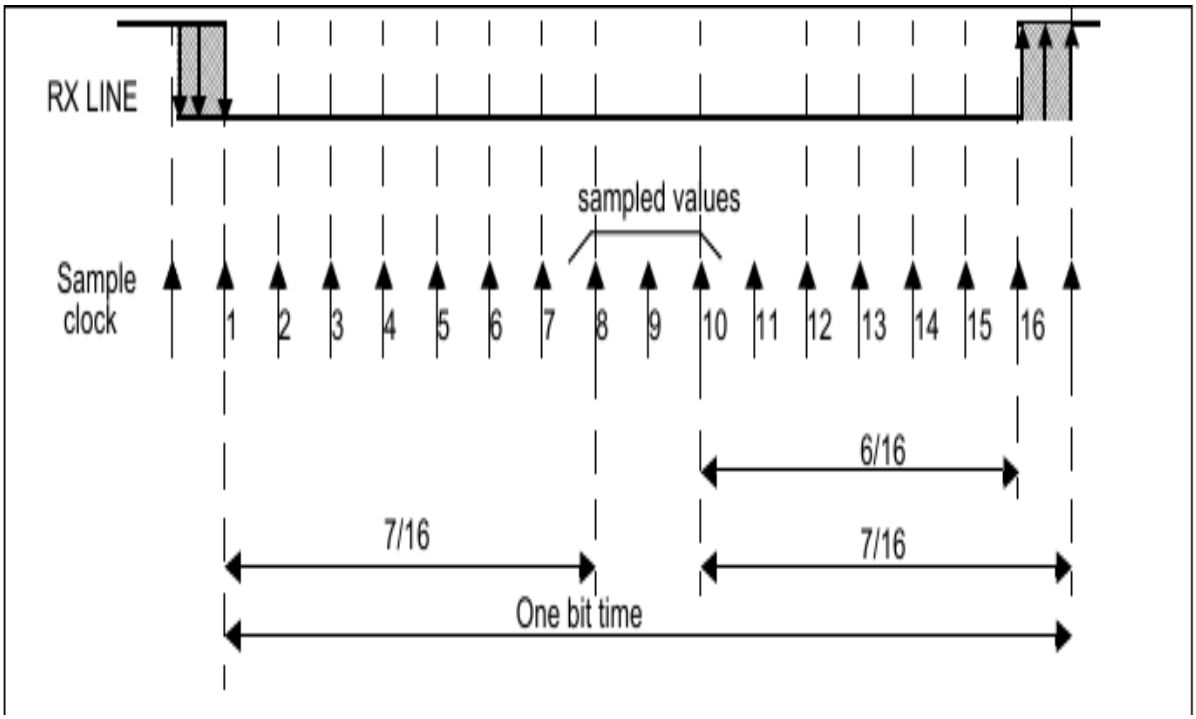


Figure 4.6 Data sampling for noise detection

4.3.4 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

Equation Baud rate for standard USART (SPI mode included)

$$Tx/Rx\ \ baud = \frac{\mathrm{f_{ck}}}{16 \times USARTDIV}$$

USARTDIV is an unsigned fixed-point number that is coded on the USART_BRR register.

Note: The baud counters are updated with the new value of the Baud registers after a write to USART_BRR. Hence the Baud rate register value should not be changed during communication.

4.3.5 Parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART_CR1 register.

Even parity: the parity bit is calculated to obtain an even number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

Odd parity: the parity bit is calculated to obtain an odd number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

4.3.6 USART synchronous mode

The CLKEN bit in the USART_CR2 register should be written to 1 to select the synchronous mode. In this mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register.
- SCEN, HDSEL and IREN bits in the USART_CR3 register.

USART allows the user to control a bidirectional synchronous serial communication in master mode. The CK pin is the output of the USART transmitter clock. No clock pulses are sent to the CK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART_CR2 register, clock pulses will or will not be generated during the last valid data bit (address mark). The CPOL bit in the USART_CR2 register allows the user to select the clock polarity, and the CPHA bit in the USART_CR2 register allows the user to select the phase of the external clock (see Figure 4.7).

During the Idle state, preamble and send break, the external CK clock is not activated.

The USART transmitter works exactly in synchronous mode like in asynchronous mode. But as CK is synchronized with TX (according to CPOL and CPHA), the data on TX is synchronous.

But the USART receiver works in a different manner compared to the asynchronous mode. If RE=1, the data is sampled on CK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time (that depends on the baud rate: 1/16-bit time) must be respected.

Note: The CK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled (TE=1) and a data is being transmitted (the data register USART_DR has been written). This means that it is not possible to receive a synchronous data without transmitting data.

The LBCL, CPOL and CPHA bits have to be selected when both the transmitter and the receiver are disabled (TE=RE=0) to ensure that the clock pulses function correctly. While the transmitter or the receiver is enabled these bits should not be changed.

It is advised that TE and RE are set in the same instruction in order to minimize the setup and the hold time of the receiver.

The USART supports master mode only: it cannot receive or send data related to an input clock (CK is always an output).

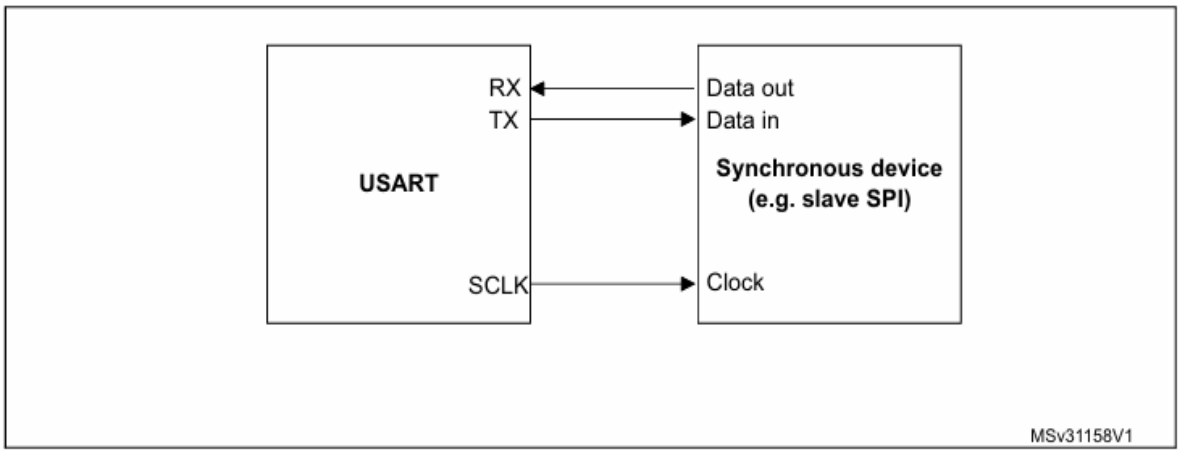


Figure 4.7 USART interface diagram

4.3.7 UART asynchronous mode

UART stands for Universal Asynchronous Receiver/Transmitter. It requires only two lines for communication. Tx and Rx for transmit and receive functions (See Figure 4.8). It is asynchronous communication. Hence data rate should be matched between devices wanting to communicate.

No common clock is being used. Moreover, devices use their own independent clock signals. UART protocol consists of start bit, 8 or 9 bits of data and stop bit which is called a frame.

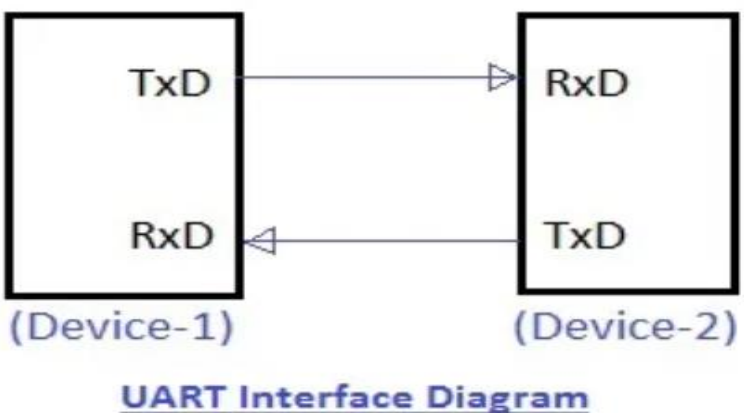


Figure 4.8 UART interface diagram

4.3.8 Single-wire half-duplex communication

By setting the HDSEL bit in the USART_CR3 register the single-wire half-duplex mode is selected. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART_CR2 register.
- SCEN and IREN bits in the USART_CR3 register.

The USART can be configured to follow a single-wire half-duplex protocol. In single-wire half-duplex mode, the TX and RX pins are connected internally. The selection between half- and full-duplex communication is made with a control bit 'HALF DUPLEX SEL' (HDSEL in USART_CR3).

As soon as HDSEL is written to 1:

- RX is no longer used.
- TX is always released when no data is transmitted. Thus, it acts as a standard IO in idle or in reception. It means that the IO must be configured so that TX is configured as floating input (or output high open-drain) when not driven by the USART.

Apart from this, the communications are similar to what is done in normal USART mode. The conflicts on the line must be managed by the software (by the use of a centralized arbiter, for instance). In particular, the transmission is never blocked by hardware and continue to occur as soon as a data is written in the data register while the TE bit is set.

4.3.9 Hardware flow control

It is possible to control the serial data flow between two devices by using the CTS input and the RTS output. (See Figure 4.9) it shows how to connect two devices in this mode:

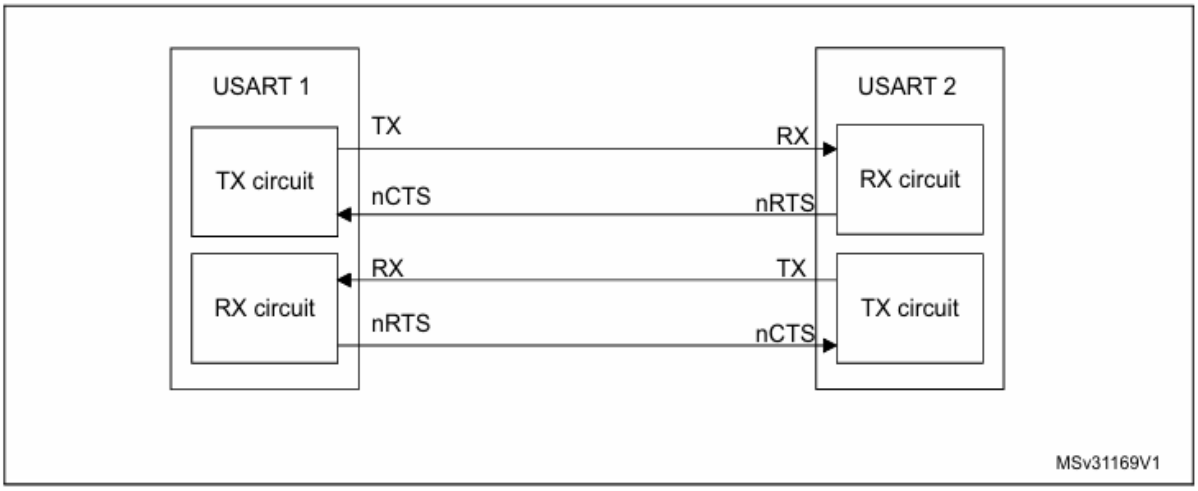


Figure 4.9 Hardware flow control between 2 USARTs

RTS and CTS flow control can be enabled independently by writing respectively RTSE and CTSE bits to 1 (in the USART_CR3 register).

RTS flow control

If the RTS flow control is enabled (RTSE=1), then RTS is asserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register is full, RTS is deserted, indicating that the transmission is expected to stop at the end of the current frame.

CTS flow control

If the CTS flow control is enabled (CTSE=1), then the transmitter checks the CTS input before transmitting the next frame. If CTS is asserted (tied low), then the next data is transmitted (assuming that a data is to be transmitted, in other words, if TXE=0), else the transmission does not occur. When CTS is deserted during a transmission, the current transmission is completed before the transmitter stops.

When CTSE=1, the CTSIF status bit is automatically set by hardware as soon as the CTS input toggles. It indicates when the receiver becomes ready or not ready for communication. An interrupt is generated if the CTSIE bit in the USART_CR3 register is set.

4.4 USART interrupts

Interrupt event	Event flag	Enable control bit
Transmit Data Register Empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	RXNEIE
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffered communication	NF or ORE or FE	EIE

Table 4.1: USART interrupt requests

The USART interrupt events are connected to the same interrupt vector (see Figure 4.10).

- During transmission: Transmission Complete, Clear to Send or Transmit Data Register empty interrupt.
- While receiving: Idle Line detection, overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

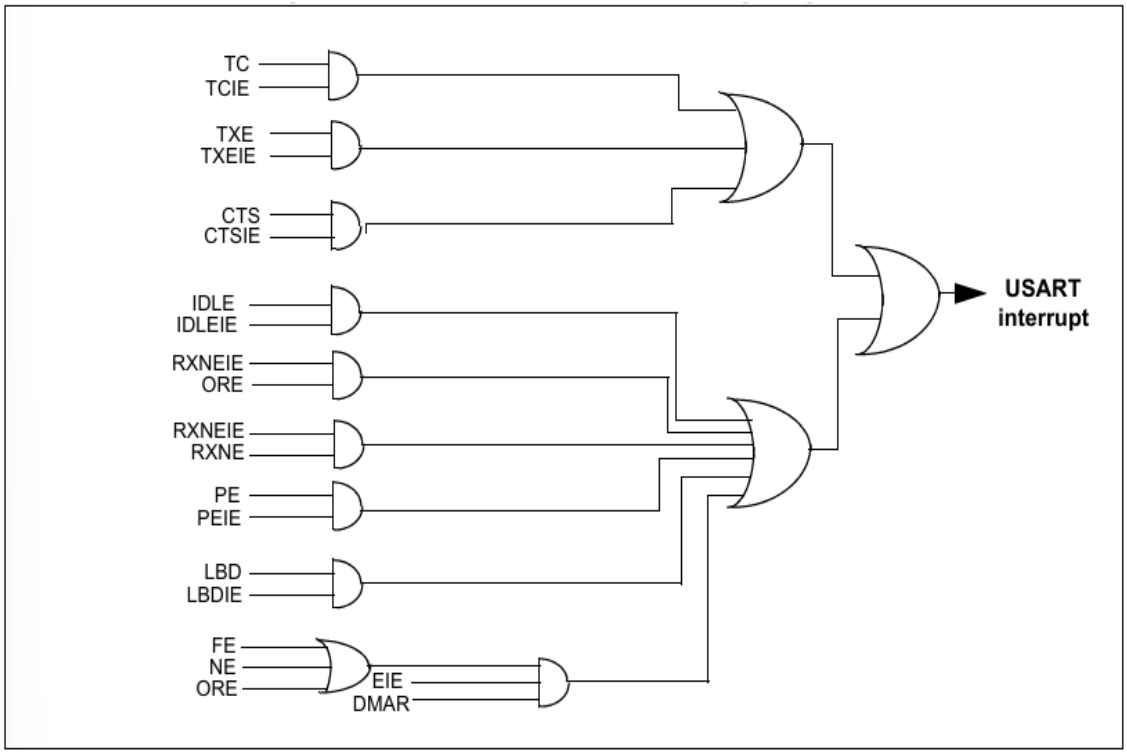


Figure 4.10 USART interrupt mapping diagram

4.5 USART mode configuration

USART modes	USART1	USART2	USART3	UART4	UART5
Asynchronous mode	X	X	X	X	X
Hardware Flow Control	X	X	X	NA	NA
Multibuffered Communication (DMA)	X	X	X	X	NA
Multiprocessor Communication	X	X	X	X	X
Synchronous	X	X	X	NA	NA
Smartcard	X	X	X	NA	NA
Half-Duplex (Single-Wire mode)	X	X	X	X	X
IrDA	X	X	X	X	X
LIN	X	X	X	X	X

Table 4.2: USART mode configuration (1)

X = supported; NA = not applicable.

4.6 USART registers

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

The table below gives the USART register map and reset values.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	Reserved																						DR[8:0]									
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x08	USART_BRR	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	USART_CR3	Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																						0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	Reserved																	GT[7:0]							PSC[7:0]							
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.3: USART register map and reset values

4.7 USART Importance and use cases

Why is it important?[29]

Familiarity with the USART communication protocol is advantageous when developing robust, quality-driven products. Knowing how to send data using

only two wires (and a clock line in case of synchronous Mode), as well as how to transport a whole pack of data or a payload, will help ensure that data is transferred and received without error. Since USART is the most used hardware communication protocol, this knowledge can enable design flexibility in future designs.

Use Cases

USARTs are no longer common for desktop computers, but they are still heavily used in industrial and embedded devices. However. Many control systems still use

USART is used for many applications, such as:

- Debugging: Early detection of system bugs is important during development. Adding USART can help in this scenario by capturing messages from the system.
- Manufacturing function-level tracing: Logs are very important in manufacturing. They determine functionalities by alerting operators to what is happening on the manufacturing line.
- Customer or client updates: Software updates are highly important. Having complete, dynamic hardware with update-capable software is important to having a complete system.
- Testing/verification: Verifying products before they leave the manufacturing process helps deliver the best quality products possible to customers.

Benefits USART [30]

Benefits or advantages of USART:

- Hardware complexity is low.
- As this is one to one connection between two devices, software addressing is not required.

Drawbacks of USART

Disadvantages of USART:

- It is suitable for communication between only two devices.
- It supports fixed data rate between devices wanting to communicate otherwise data will be garbled.
- In synchronous mode extra connections required for the synchronous signal.

CHAPTER 5: BLUETOOTH

In this chapter, we'll talk about Bluetooth in general.

5.1 Introduction [31]

Bluetooth is used for short-range wireless voice and data communication. It is a Wireless Personal Area Network (WPAN) technology and is used for data communications over smaller distances. This generation changed into being invented via Ericson in 1994. It operates within the unlicensed, business, scientific, and clinical (ISM) bands from 2.4 GHz to 2.485 GHz.
Bluetooth stages typically up to 10 meters (33 feet) but can extend up to 100 meters (328 feet) with Bluetooth 5.0. Depending upon the version, it presents information up to at least 1 Mbps or 3 Mbps.

The Bluetooth protocol is divided into several layers, including a physical layer that handles the radio transmission of data, a link layer that establishes connections between devices, and an application layer that defines the data that is transmitted between devices.
The spreading method that it uses is FHSS (Frequency Hopping Spread Spectrum) to reduce interference from other wireless devices operating in the same frequency band.
This protocol is used in a wide range of applications, including mobile phones, wireless headphones, smart speakers, gaming controllers' devices.

A Bluetooth network is called a piconet and a group of interconnected piconets is called a scatternet.

5.2 Advantages of Bluetooth:[32]

Bluetooth offers several advantages over other wireless communication technologies, including:

- Low Power Consumption: Bluetooth uses very little power, which makes it an ideal choice for battery-powered devices.

- Compatibility: Bluetooth is a widely adopted technology, which means that most devices are compatible with it.
- Ease of Use: Bluetooth devices can be paired and connected quickly and easily, without the need for complex setup procedures.

5.3 Disadvantages of Bluetooth:

- It can be hacked and hence, less secure.
- It has a slow data transfer rate of 3 Mbps.
- Bluetooth communication does not support routing.

5.4 Architecture of Bluetooth [31]

-The architecture of Bluetooth defines two types of networks:

- **Piconet:** Piconet is a type of Bluetooth network that contains one primary node called the master node and seven active secondary nodes called slave nodes. Thus, we can say that there is a total of 8 active nodes which are present at a distance of 10 meters. The communication between the primary and secondary nodes can be one-to-one or one-to-many. Possible communication is only between the master and slave; Slave-slave communication is not possible. It also has 255 parked nodes; these are secondary nodes and cannot take participation in communication unless it gets converted to the active state.

- **Scatternet:** It is formed by using various piconets. A slave that is present in one piconet can act as master or we can say primary in another piconet. This kind of node can receive a message from a master in one piconet and deliver the message to its slave in the other piconet where it is acting as a master. This type of node is referred to as a bridge node. A station cannot be mastered in two piconets.

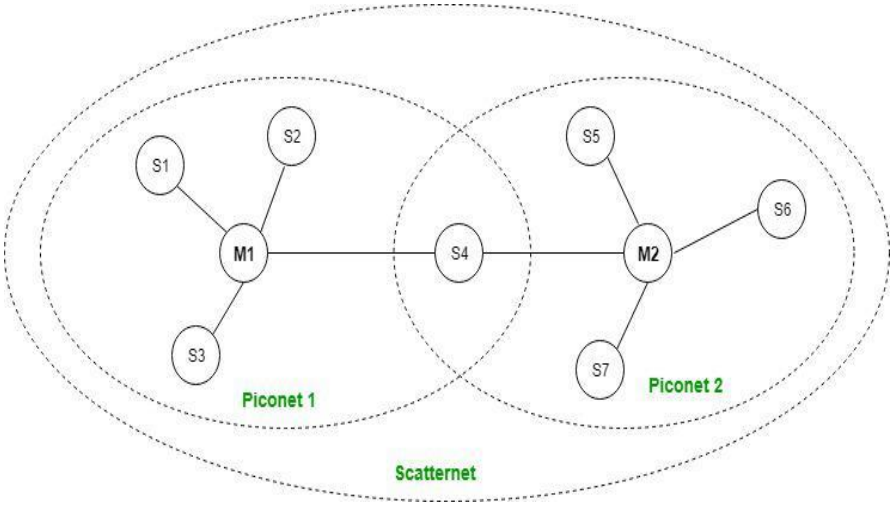


Figure 5.1 Bluetooth Architecture

5.5 Bluetooth Implementation:

A typical implementation of Bluetooth involves several components, including the RF, baseband, HCI interface, and host stacks software.
The RF component operates in the license-free 2.4-2.4835GHz ISM band and uses frequency hopping at a rate of 1600 hops/s across 79 1MHz channels. The range for Bluetooth is typically 10 meters with a 1Mbps rate, but this can be increased to 100 meters with improved transmission power and receiving sensitivity.

The RF component has three classes of transmit power: Class 3 at 0 dBm (1 mW), Class2 at 4 dBm (2.5 mW), and Class 1 at 20 dBm (100 mW).
The minimum receiver sensitivity level is -70 dBm, with most products operating between -75dBm and -90dBm for a 10-3 BER. The baseband component of Bluetooth provides cable replacement for connectivity among various devices in a wireless PAN, also known as a Piconet. Multiple devices, up to 256 parked and 8 actives, can participate in a Piconet.

Bluetooth communications in a Piconet are based on a master/slave relationship, where one unit serves as a master and the rest serve as slaves. The access is synchronized via the master identity, whose Bluetooth address determines the frequency hopping sequence, and the system clock determines the phase.
Each slave follows the hop sequence and adds an offset to its clock to follow the master.

Each Bluetooth packet has a fixed format that starts with a 72-bit access code unique to the Piconet and based on the master identity. A 54-bit header follows, containing error correction, retransmission, and control information.
Finally, a payload of 0 to 2745 bits ends a packet.

5.6 Frequency hopping:

Bluetooth uses frequency hopping at a rate of 1600 hops/s, across 79 one MHz channels using a 1 bit/symbol GFSK modulation scheme. To provide full duplex operation, Bluetooth uses Time-Division Duplex (TDD) scheme to divide the channel into a number of 625 us time slots with a 220 us TDD guard time. The master and slave devices alternate transmission, with the master transmitting in even-numbered time slots, and the slave transmitting in odd-numbered time slots. Bluetooth can support three 64Kbps voice channels through synchronous connection-oriented (SCO) links for circuit-switching audio applications. It also supports asynchronous connection-less (ACL) links for packet data switching, based on a polling access scheme. The typical packet size is one slot, but it can span multiple slots as defined in the specification. Bluetooth provides a maximum data rate of asymmetrical 721 Kb/s upstream (57.6Kb/s downstream) or symmetrical 432.6 kb/s data transfer rate.

5.7 How Does Bluetooth Module Work?[33]

The Bluetooth Module acts as a bridge between two devices, enabling them to establish a secure and efficient communication channel. It uses the UART/TTL communication interface, which is a standard serial communication protocol. This makes it compatible with a wide range of microcontrollers, such as Arduino, Raspberry Pi, and other embedded systems.

Pairing the Bluetooth Module with another device is straightforward. When powered on, the module enters the “AT command mode,” allowing you to configure its settings via simple AT commands. Once configured, it can operate in either master or slave mode.

-Master Mode: In this mode, the takes the initiative to connect to other Bluetooth devices (e.g., smartphones, tablets, or PCs) acting as slaves. It enables seamless data transmission between the master and slave devices.

-Slave Mode: When the is set to slave mode, it waits for incoming connection requests from Bluetooth master devices. Once paired, it facilitates bidirectional communication with the master device.

5.8 Bluetooth Usage:[34]

Bluetooth can be used to control a car through a mobile application. This allows for remote control of various functions of the car, such as locking and unlocking the doors, starting the engine, adjusting the temperature, and more. The mobile application can send commands to the car via Bluetooth, which is integrated into the car's system. This technology provides a convenient and secure way for car owners to control their vehicles remotely using their mobile devices.

Circuit diagram for Bluetooth controlled car is shown in figure.
Bluetooth controlled car moves according to button touched in the android Bluetooth mobile app. We can use any Bluetooth app that supporting or can send data.

When we touch the forward button in the Bluetooth controller app, the car starts moving in the forward direction and continues moving forward until the next command comes. Also, when the back, right or left button is touched, it responds to the command. By touching the stop button, we can stop the car.

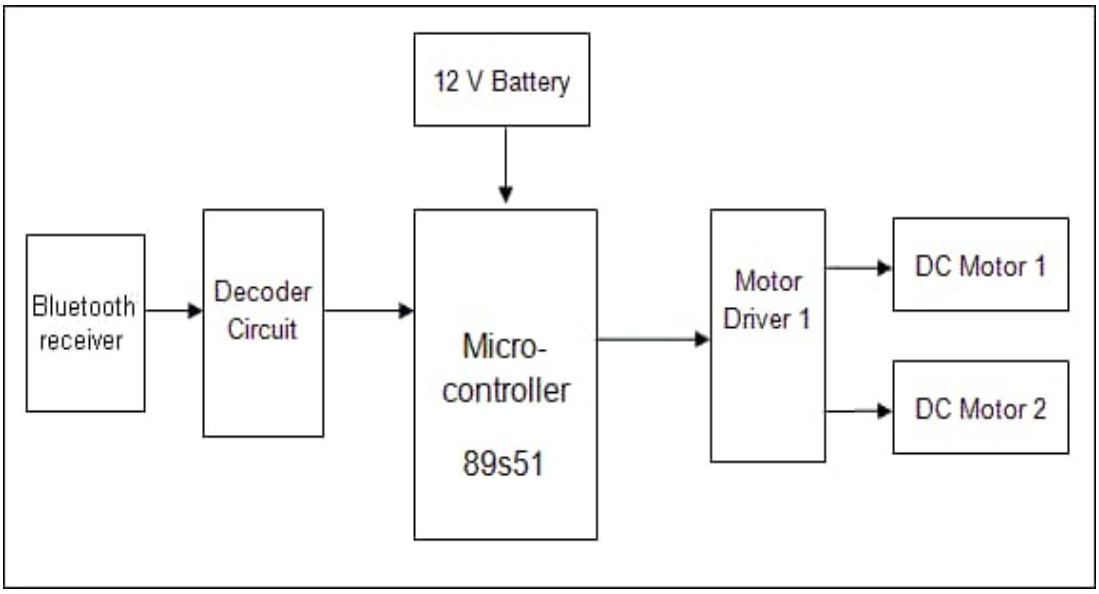


Figure 5.2 Block diagram of Bluetooth subsystem

5.9 Connecting Bluetooth module with Microcontroller [35]

- Connect Power Supply (based on datasheet of modules) for Bluetooth and Microcontroller which you are using.
- Connect TXD pin of Bluetooth module to RXD pin of Microcontroller.
- Connect RXD pin of Bluetooth module to TXD pin of Microcontroller.
- Common grounding should be needed for both modules.

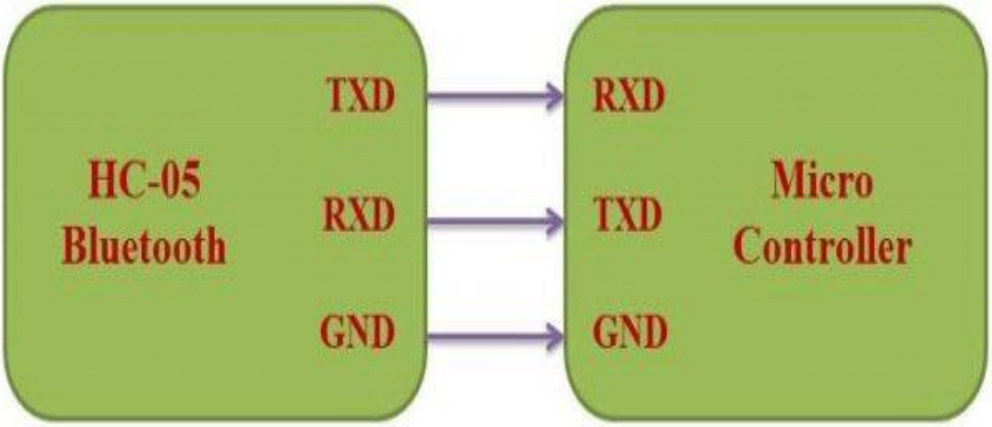


Figure 5.3 Connection between HC-05 and MCU

Module C
Subsystems

Chapter 6: ACC System.

Chapter 7: AEB System.

Chapter 8: Self-Parking System.

CHAPTER 6: ADAPTIVE CRUISE CONTROL SYSTEM

6.1 System Description [36]

Adaptive Cruise Control (ACC) is an automotive feature that allows a vehicle's cruise control system to adapt the vehicle's speed to the traffic environment. An ACC equipped vehicle has a radar or other sensor that measures the distance to other preceding vehicles (downstream vehicles) on the highway. In the absence of preceding vehicles, the ACC vehicle travels at a user-set speed, much like a vehicle with a standard cruise control system (see Figure 6.1). However, if a preceding vehicle is detected on the highway by the vehicle's radar, the ACC system determines whether or not the vehicle can continue to travel safely at the desired speed. If a slower moving vehicle is detected, the ACC system will slow the vehicle down and control the clearance, or time gap, between the ACC vehicle and the forward vehicle.

If the system detects that the forward vehicle is no longer in the ACC vehicle's path, the ACC system will accelerate the vehicle back to its set cruise control speed. This operation allows the ACC vehicle to autonomously slow down and speed up with traffic without intervention from the driver. The method by which the ACC vehicle's speed is controlled is via engine throttle control and limited brake operation.

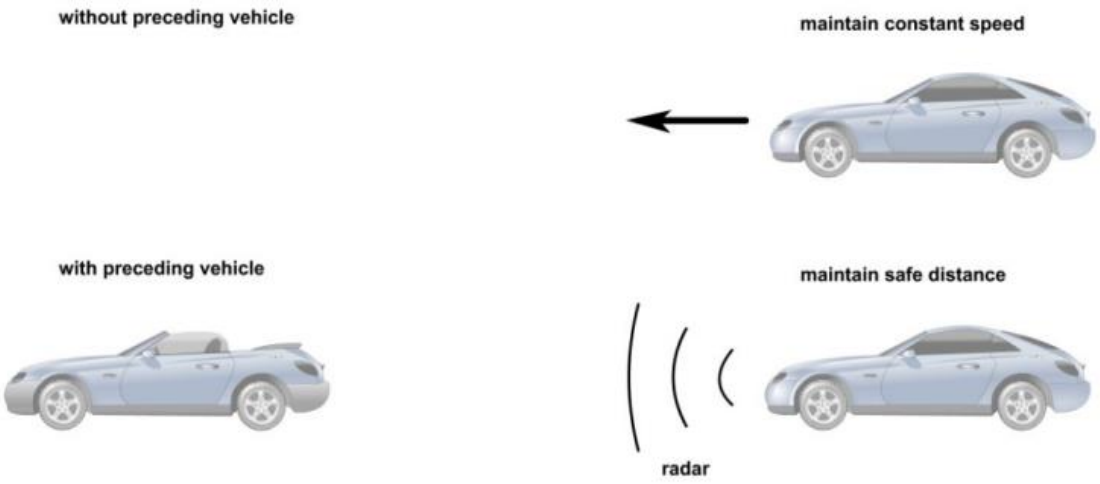


Figure 6.1 Adaptive cruise control

6.2 The advantages of system [37]

The advantages of the adaptive cruise control system are listed as follows:

1. The ACC system automatically controls the vehicle to replace a manual operation from the driver during long-distance driving and releases the driver from pressing the accelerator pedal or brake pedal for a long time.
2. The ACC system helps to avoid traffic accidents caused by drivers (fatigue driving, mis operation, etc.)
3. The ACC system improves ride comfort and reduces fuel consumption by smoothly controlling the speed and acceleration of the vehicle accurately, which cannot be achieved by human drivers.
4. The ACC system can avoid mechanical wear caused by frequent acceleration and deceleration of novice and prolong the service life of vehicle parts.
5. The ACC system has the potential to increase the traffic flow by applying a smaller distance between vehicles that can be realized due to a significant
   shorter response time of the adaptive cruise control system compared with human drivers.

6.3 The ACC system operating mode [38]

A vehicle (ego car) equipped with adaptive cruise control (ACC) has a sensor, such as radar, that measures the distance to the preceding vehicle in the same lane (lead car), D_rel. The sensor also measures the relative velocity of the lead car, V_rel. The ACC system operates in the following two modes:

- Speed control: The ego car travels at a driver-set speed.
- Spacing control: The ego car maintains a safe distance from the lead car.

The ACC system decides which mode to use based on real-time radar measurements. For example, if the lead car is too close, the ACC system switches from speed control to spacing control. Similarly, if the lead car is further away, the ACC system switches from spacing control to speed control. In other words, the ACC system makes the ego car travel at a driver-set speed as long as it maintains a safe distance.
The following rules are used to determine the ACC system operating mode:

- If D_rel>=D_safe, then speed control mode is active. The control goal is to track the driver-set velocity, V_set.
- If D_rel<D_safe, then spacing control mode is active. The control goal is to maintain the safe distance, D_safe

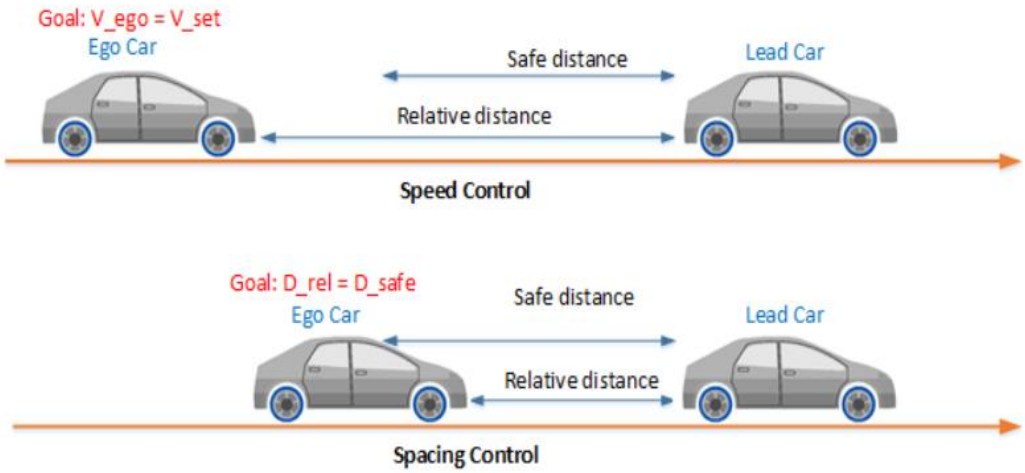


Figure 6.2 ACC system operating mode

6.4 Main Algorithm

- **Flow Chart**

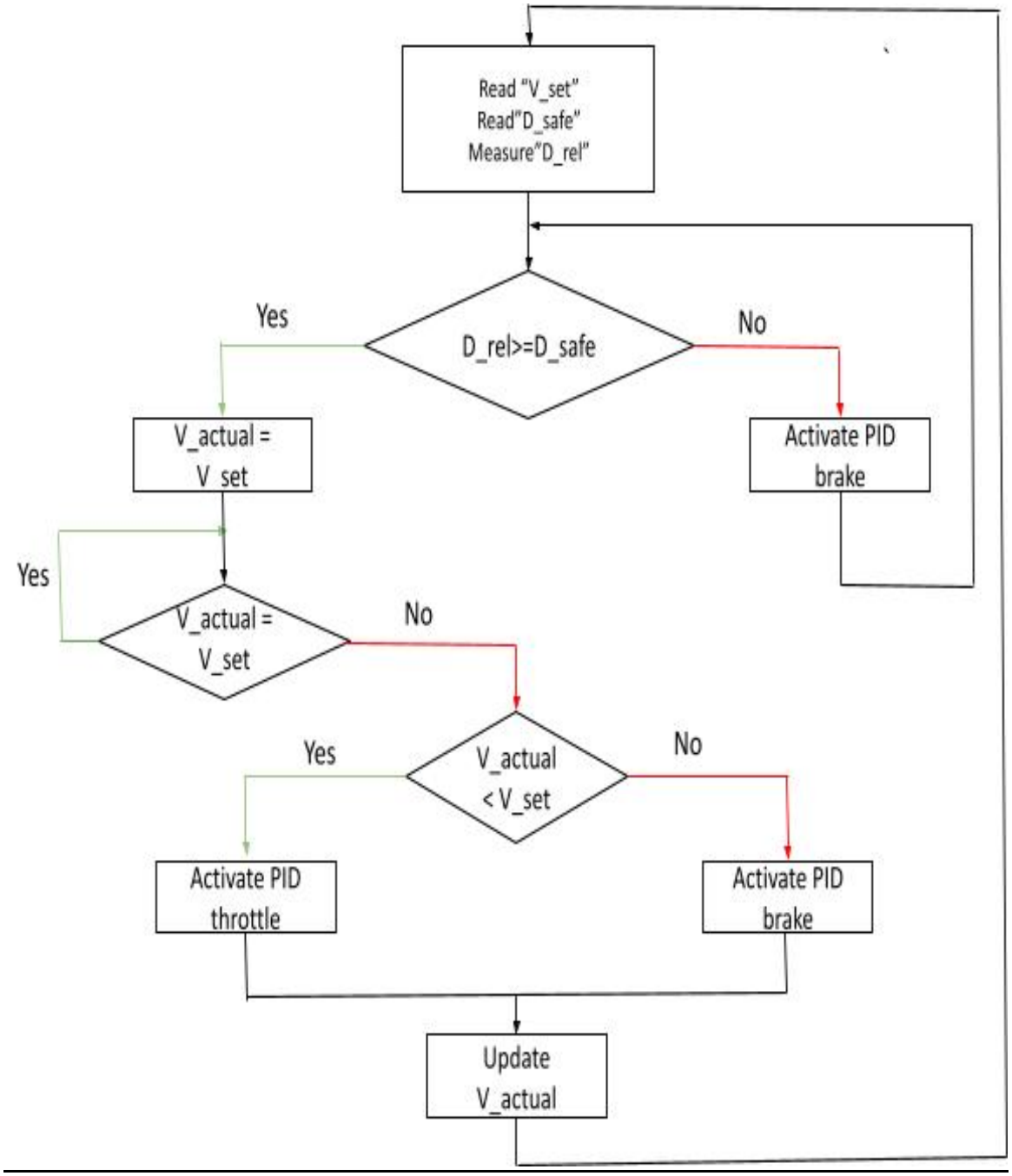


Figure 6.4 Flow Chart of Main Algorithm

- **Pseudo Code**

```
BEGIN
READ    Ultrasonic_Distance_value
IF      Ultrasonic_Distance_value >= SAFE_DISTANCE
CALL
ELSE    Ultrasonic_Distance_value < SAFE_DISTANCE
CALL
ENDIF
END
```

Figure 6.4 Pseudo Code of Main Algorithm

CHAPTER 7: AUTO EMERGENCY BRAKING SYSTEM

7.1 Introduction [39]

Vehicle technology has increased rapidly in recent years, particularly in relation to braking systems and sensing systems. Automatic braking may be a safety technology that automatically activates the vehicle's brake, to the point, when necessary.

Systems can vary from pre-charging brakes to slowing the vehicle to reduce damage. Nowadays, some advanced and updated systems completely take over and stop the vehicle before a collision happen. The precise capabilities of their car's automatic braking system. Regardless of a vehicle's autonomous technologies, drivers should remain conscious of their surroundings and maintain control in the least times. The automatic braking or brake assist is an integral component of crash avoidance technologies, including front crash prevention systems, back over prevention systems, and cross-traffic alert systems. Each automaker may have a special name for such technologies, but the rock bottom line is that the brake assist is supposed to attenuate accidents.

If no vehicle is ahead, the vehicle maintains the desired "set-speed", ACC can be ordered as an option for new vehicles: At least three heavy truck manufacturers offer this feature on their vehicles, Theoretically, a vehicle equipped with modern braking technology and adaptive cruise control is equipped with all of the necessary hardware to allow a simple (braking only no steering) collision: avoidance system that would be capable of detecting when a collision is likely to occur and applying emergency braking to avoid it. Collision mitigation systems are already on the market, providing limited braking capability Integrated safety systems based on these principles we implement this category.

Collision avoidance: Sensors detect a potential collision and take action to avoid it entirely, taking control away from the driver, In the context of braking this is likely to include applying emergency braking sufficiently early that the vehicle can be brought to a standstill before a collision occurs. In future, this could also include steering actions independent of the driver This category is likely to have the highest potential benefits but is the highest risk approach because a false activation of the system has the potential to increase the risk to another road [40].

Forward collision warning: Sensors detect a potential collision and take action to warn the driver. This is the least risky option since false detection of a collision only has impacts on the driver's reaction to, and perception of, the system. This type of system could also be used to optimize restraints. This type of system has been sold on some EU vehicles since 1999. The original focus of this project was to assess the technical requirements, costs and benefits relating to collision mitigation braking systems in their current form, however, as the project has developed the scope has expanded to consider, the potential benefits that could arise from automatic emergency braking systems (AEBS) that include avoidance capabilities [40].

7.2 Problem Definition [41]

Figure 1: Driver View. Figure 2: Car brakes suddenly

Figure 7.1 AEB System

Nowadays, the number of accidents is so high and occurs everywhere causing the worst damage, serious injuries and death. These accidents can be caused due to the delay of the driver to hit the brakes.

AEB is a safety system that can identify when a possible collision is about to occur and responds by autonomously activating the brakes to bring the vehicle to stop to avoid collision. This technology commonly uses ultrasonic sensor, radar, camera or lidar to get a good image and calculations for the surrounding environment and objects.

AEB also has access to the information on your car through the electronic control unit (ECU), by taking on your car's speed and calculate the distance from the object in front of you and behind you, if AEB detect a potential collision, it will check the braking system. If you have already engaged the brakes and are decelerating enough to avoid the collision, AEB won't intervene. However, if you have not hit the brakes with enough force to stop, AEB will take over and brake with enough force to avoid the collision.

The objective of this project is to design the automatic braking system in order to avoid the accident. To develop a safety vehicle braking system using Ultrasonic sensors and to design a vehicle with less human attention to the driving. This project is necessary to be attached to every vehicle. Mainly it is used when drive the vehicles in night time. Mostly the accident occurred in the night time due to long travel the driver may get tired, so the driver may hit the front side vehicle or road side trees. By using this project, the vehicle is stopped by automatic braking system, so we can avoid the accident.

7.3 System Description

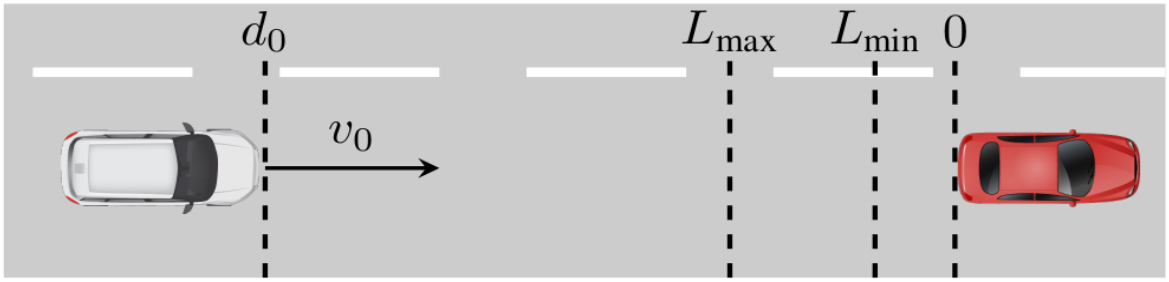


Figure 7.2 AEB Scenario 1

Active brake safety technology is mainly composed of three modules, including control module (ECU), distance measurement module and brake module. First use ultrasonic sensor to measure the distance to the vehicle or obstacle in front or behind to it, then compare the measured distance with the warning distance and safety distance. If the distance is less than the threshold of warning distance, a buzzer will be heard by the driver and surrounding cars to alert the driver that he is about to crash. When the distance is less than the safety distance, will make the car brake automatically, thus escorting the safe travel.

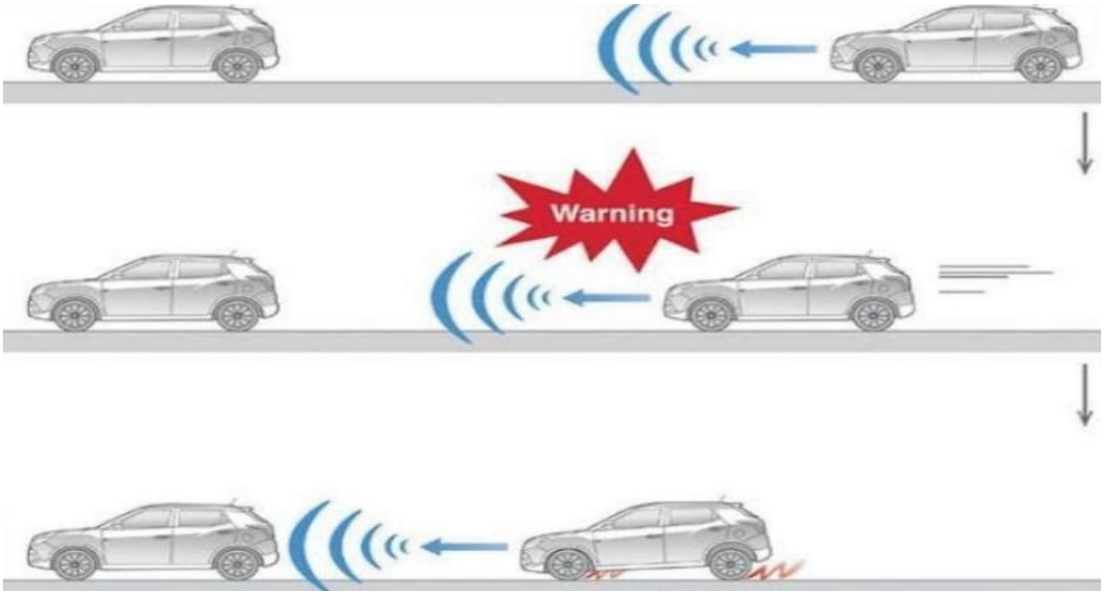


Figure 7.3 AEB Scenario 2

7.4 Algorithm Flow Chart

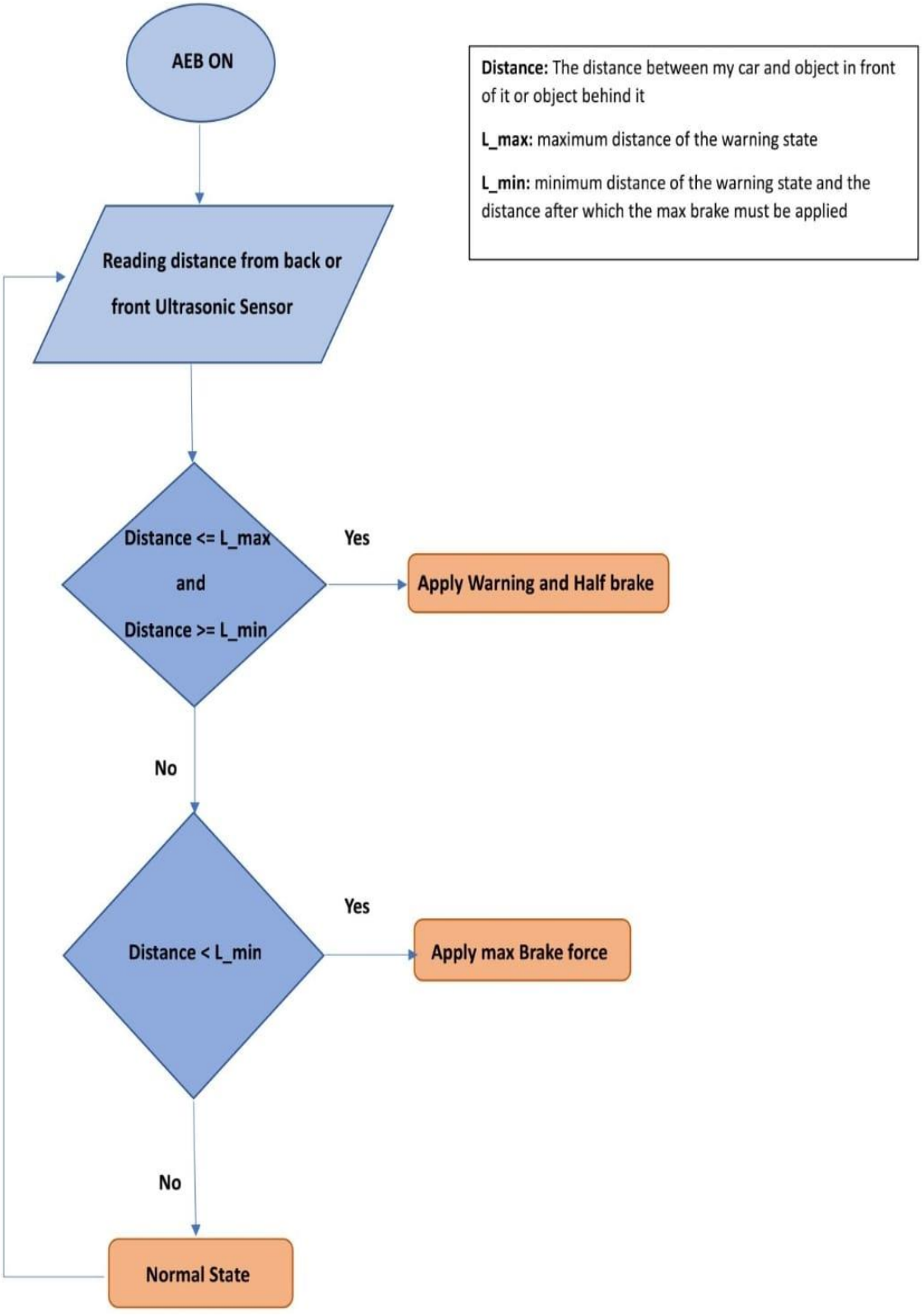


Figure 7.4 Algorithm Flow Chart

Normal State

In this state, our car is moving without any constrains. The car is fully controlled by the driver without any interaction of our system. If the car detects an opposite object that gets closer and may crash into, the system will transfer into the Warning State.

➢ `Distance From Ultrasonic > MAXIMUM DISTANCE`

Warning State

When ultrasonic sensor starts to detect an object that gets closer to our car and the driver did not take any action to slow down the car velocity, our system starts to raise a warning signal to warn the driver indicating that the danger is coming, if he did not take action. If the driver took the action and the distance became in the safe region again the system will return back to the Safe State.

➢ `MINIMUM DISTANCE <= Distance From Ultrasonic <= MAXIMUM DISTANCE`

Auto Braking State

In this state, the driver did not take any action to slow down the car to prevent it from crashing into the object. If the brake does not start immediately the car will crash the object, so the system takes the control and starts to apply the braking force till maximum the car fully stops then the control is back to the driver and the system returns into the Safe State again.

➢`Distance From Ultrasonic < MINIMUM DISTANCE`

CHAPTER 8: SELF PARKING SYSTEM

8.1 Introduction [42]

Micro-controllers are used to develop intelligent systems that can be found in different applications in our daily lives. In this project, we aim to design Self-Parking Car using stm32f103C8T6 "Blue-pill". Self-car driving we are developed autonomous cars using the ultrasonic sensor. In this car, we used three ultrasonic sensors to find the obstacles sides. If anyone side obstacles detected means the car will move in another direction.

The main motivation for building this system is how self-parking works. This system makes the car park between two cars without hitting any surrounding car and it reduces many of problems that people suffer from.

The goal of this system is to:

1. Exploiting narrow spaces.
2. Helping drivers who do not have high driving skills.
3. Helping drivers with special needs.
4. Reducing traffic accidents.

8.2 The advantages of this project are [42]:

1. No need to worry about the cars from accidents.
2. Reducing the worry of people who do not have high skills in driving.
3. Helping people with special needs.

8.3 Problem Statement:

Most of the time people suffer from the problem of parking the car in the appropriate places due to high traffic demand, Following the rapid increase of

car ownership, many cities are lacking car parking areas. So, this project was chosen, which aims to stop the car between two cars independently.

8.4 System Description

A parallel self-parking system is designed to using a combination of ultrasonic sensors, H-bridge circuits, and DC motors. This system simplifies the parking process, ensuring precise and safe maneuvering into parking spaces with minimal driver intervention. Ultrasonic sensors, placed on the front, rear, and sides of the vehicle, continuously scan for available parking spaces and detect obstacles.

Once a suitable space is identified, the system calculates the dimensions and confirms if the space is adequate for parking. The H-bridge circuit, which controls the direction and speed of the DC motors, directs the motors to maneuver the vehicle into the parking space by adjusting the steering angle and wheel speed. This precise control is essential for executing the parallel parking maneuver.

The ultrasonic sensors provide continuous feedback to ensure the vehicle avoids obstacles and maintains the correct trajectory. An LCD display offers real-time information about the parking process, including distances to nearby obstacles and step-by-step guidance, ensuring the driver remains informed and can intervene if necessary.

The system makes fine adjustments to the vehicle’s position to ensure it is properly aligned within the parking space, resulting in a perfect park. By integrating these components, the parallel self-parking system enhances the convenience and safety of parking, making the process intuitive and stress-free for the driver.

8.5 System Requirements [43]:

The system must achieve the following functionality:

1. The system must be user-friendly to be used by a beginner and advanced users.
2. Design a suitable car that achieves the goal of the project.
3. Checking all project parts.
4. High accuracy in the driver.
5. High accuracy in the sensors.

8.6 Expected Results:

We expect to accomplish the following at the end of the project:

1. A simple, low-cost system used to Self-Parking Car.
2. Stops the car properly.
3. The system reducing the number of collisions between cars.
4. Sensor data should be highly accurate.
5. Install the appropriate code on the board.

8.7 Algorithm Flow Chart

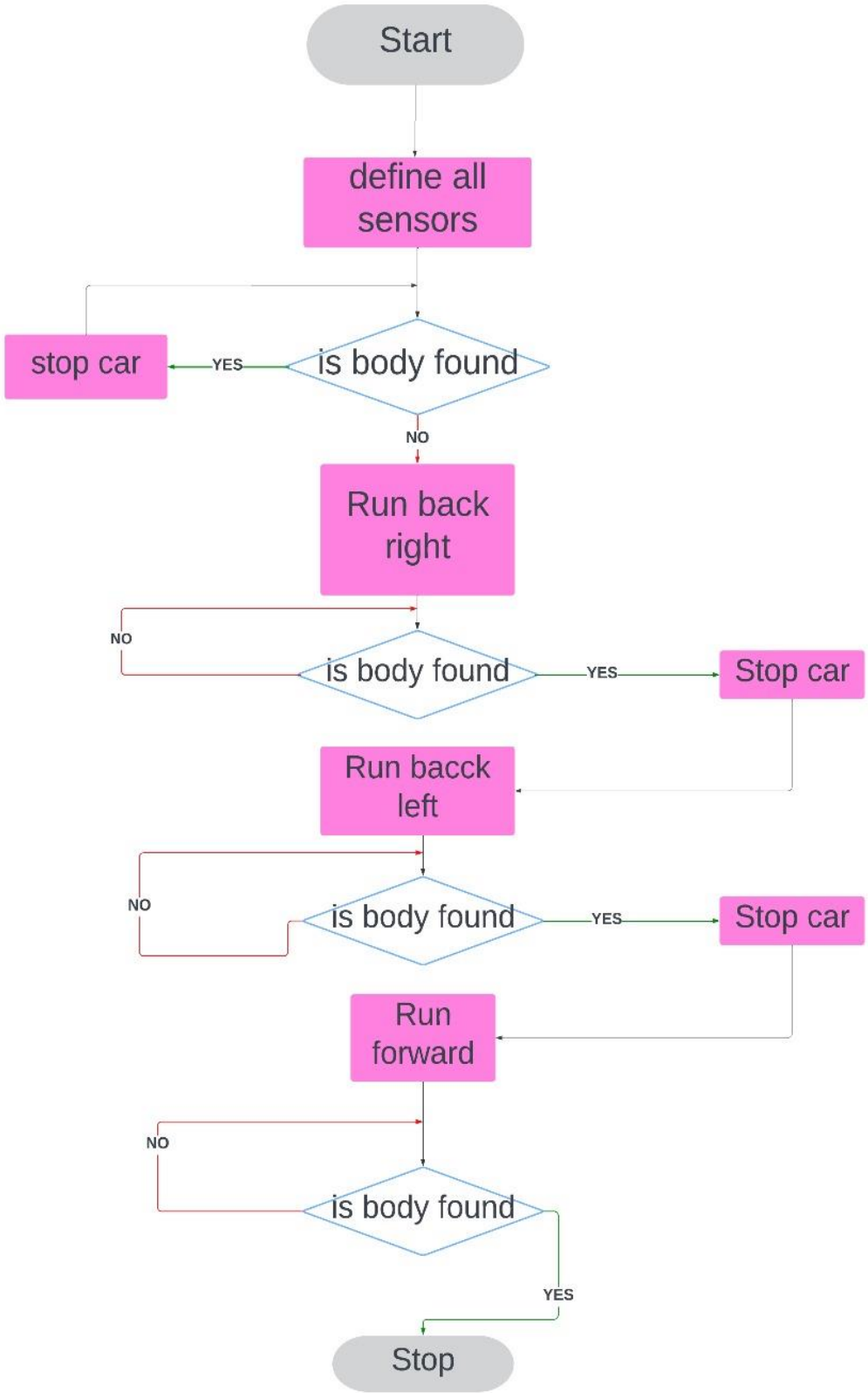


Figure 8.1 Algorithm Flow Chart

Module D
Embedded Linux Application with IOT

Chapter 9: Embedded Linux.

Chapter 10: Internet of Things (IoT).

Chapter 11: Embedded Linux &Internet of Things (IoT) tasks.

CHAPTER 9: EMBEDDED LINUX

In this chapter, we will talk about Linux and its features. Also, will talk about Embedded Linux, its Architecture, and explain its component individually. Finally, we will explain what the Build systems is and explain Yocto in detail.

9.1 Introduction [44]

Nowadays, Technology development increases rapidly, so we need applications more complicated with a huge number of features. To deal with these expectations we need powerful tools, like the Linux operating system to control these complex applications of embedded systems devices. Ten years ago, embedded processors had limitations and didn't have the same capabilities as today's processors. Now microcontrollers or system on chip (SOC) generally provide interfaces with high application and suitable memory size and high speed. Undoubtedly, Embedded Linux will be applied to more and more devices.

Linux provides the flexibility and power of open-source development and offers greater security. Also, in a bare-metal environment, we are limited to a single application image. As we build out the application, we'll notice things get hard if our system must do a few totally different things simultaneously. If we're developing for Linux, we can break this functionality into separate processes, where we can develop, debug, and deploy separately as separate binary images. So, let's know more details about Linux.

9.2 Linux

9.2.1 Linux History

In the late 1960s, Bell Labs was involved in a project with MIT and General Electric to develop a time-sharing system, called Multiplexed Information and Computing Service (Multics). Multics doesn't make a good success, so Bell labs decided to pull out [45].

In 1969, a team of programmers in Bell Labs who worked in Multics decided to make a better OS, they called it Unimplied Information and Computing Service (UNICS).
The team was led by great programmers:

- **Ken Thompson**
- **Dennis Ritchie**

In 1971, the First Edition of Unics was released. In 1973, Ritchie rewrote B and called the new language the C language. In 1975, Unix V6 became very popular. Unix V6 was free and was distributed with its source code to many universities. Hence, many organizations had modified the Unix source code to build their own distributions. The most popular distributions are:

- Berkeley Software Distribution (BSD)

In 1983, Bell Labs found that there are many distributions of Unix that are uncontrolled and untracked. They decided to make a closed commercial version of Unix and called it System V.

In 1983, Richard Stallman started the development of 100% free software. Not 90%
and not 99.5%. It is totally free. Free not mean free in cost, but free from freedom!
It means that this system would give the user:

- The freedom to run the program as you wish
- The freedom to copy the program and give it
- away to your friends and co-workers
- The freedom to change the program as you wish
- The freedom to distribute an improved version and thus help build the community.

GNU is a recursive acronym meaning GNU's Not Unix, A Unix-like operating system includes a kernel, compilers, editors, text formatters, mail software, graphical interfaces, libraries, games, and many other things. Thus, writing a whole operating system is a very large job. Stallman started in January 1984.
One of these most important components is the GNU C Compiler GCC. By 1990 we had either found or written all the major components except one the kernel.
In 1991, Linus Torvalds build a free, open-source kernel name its Linux.

Combining Linux with the almost-complete GNU system resulted in a complete operating system: **GNU/Linux system** [45].

Estimates are that tens of millions of people now use GNU/Linux systems. Nowadays, there are many distributions of GNU/Linux systems such as:

- Ubuntu
- Gentoo
- CentOS
- Debian
- Linux mint Fedora

9.2.2 Why Linux? [45]

Linux is like other operating systems such as Windows, macOS, or IOS. Like them, Linux can have a graphical interface and the types of desktop software that you are used to, such as word processors, photo editors, video editors, etc. But Linux also differs in many important ways.

First, and perhaps its most important feature, it is open-source software. The code used to create Linux is free and available for the public to view, edit, and for users to contribute to it. Although the core pieces of the Linux operating system are generally widespread, many Linux distributions include different software options. That means that Linux is incredibly customizable. Linux is present in the software of many devices that we use daily. Linux is secure, flexible, and can receive excellent support from a large community of users.

Linux is Multitasking and Multiuser. IN Linux, you get superuser access and privileges (for real), as Linux OS will not take any step without the consent of the superuser. While Linux installation, you don't need to install drivers for Wi-Fi, Bluetooth, mouse, touchpad, etc. explicitly as they can be installed during installation with little patience.
Linux community is loyal to all the Linux users so it would give Long Term Support. Linux covers many device drivers. Linux is ported on different platforms and is platform-independent.

For other Oss, usually, a user would have to go to the manufacturer's website to get driver support for different types of hardware. The Linux kernel supports most of the hardware automatically via plug and-play (largely in part because of

the open-source community). Some manufacturers also develop Linux versions of their proprietary drivers which could be easily installed via the software repository of distribution or by manually installing the provided binaries.

For these and other reasons, we are seeing an accelerated adoption rate of Linux in many Applications.

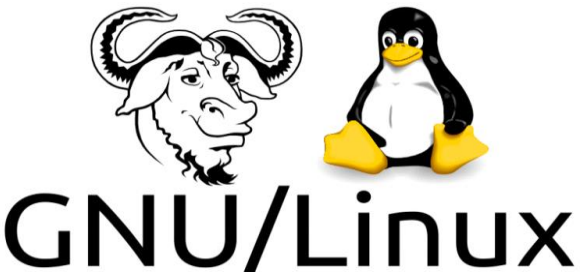


Figure 9.1 GNU/Linux OS

9.2.3 The difference between Linux and Windows

Key	Linux	Windows
Open Source	Linux is Open Source and is free to use.	Windows is not open source and is not free to use.
Case sensitivity	Linux file system is case-sensitive.	Windows file system is case insensitive.
Kernel type	Linux uses a monolithic kernel.	Windows uses a microkernel.
Efficiency	Linux is more efficient in operations as compared to Windows.	Windows is less efficient in operations.
Path Separator	Linux uses a forward slash as a path separator between directories.	Windows uses backward slash as a path separator.
Security	Linux is highly secure as compared to Windows.	Windows provides less security as compared to Linux.

Table 9.1 Linux vs Windows

9.3 Embedded Linux

Embedded Linux is a type of Linux operating system/kernel that was designed to be installed and used in embedded devices or systems. An embedded system is a set of computer hardware and software based on a microcontroller or microprocessor, controlled by a real-time operating system or RTOS, with limited memory, and that can vary both in size and complexity [46].

Although it uses the same kernel, embedded Linux is quite different from the standard operating system. First, it gets Customized for embedded systems and, therefore, is much smaller in size, requires less processing power, and has minimal features. The Linux kernel is modified and optimized as an embedded Linux version. Such a Linux instance can only run applications created specifically for the device [46].

9.4 Embedded Linux Architecture

Every Embedded Linux project begins by obtaining, customizing, and deploying these Elements (Components):

- Toolchain
- Bootloader
- Kernel
- Root filesystem
- Specific Application

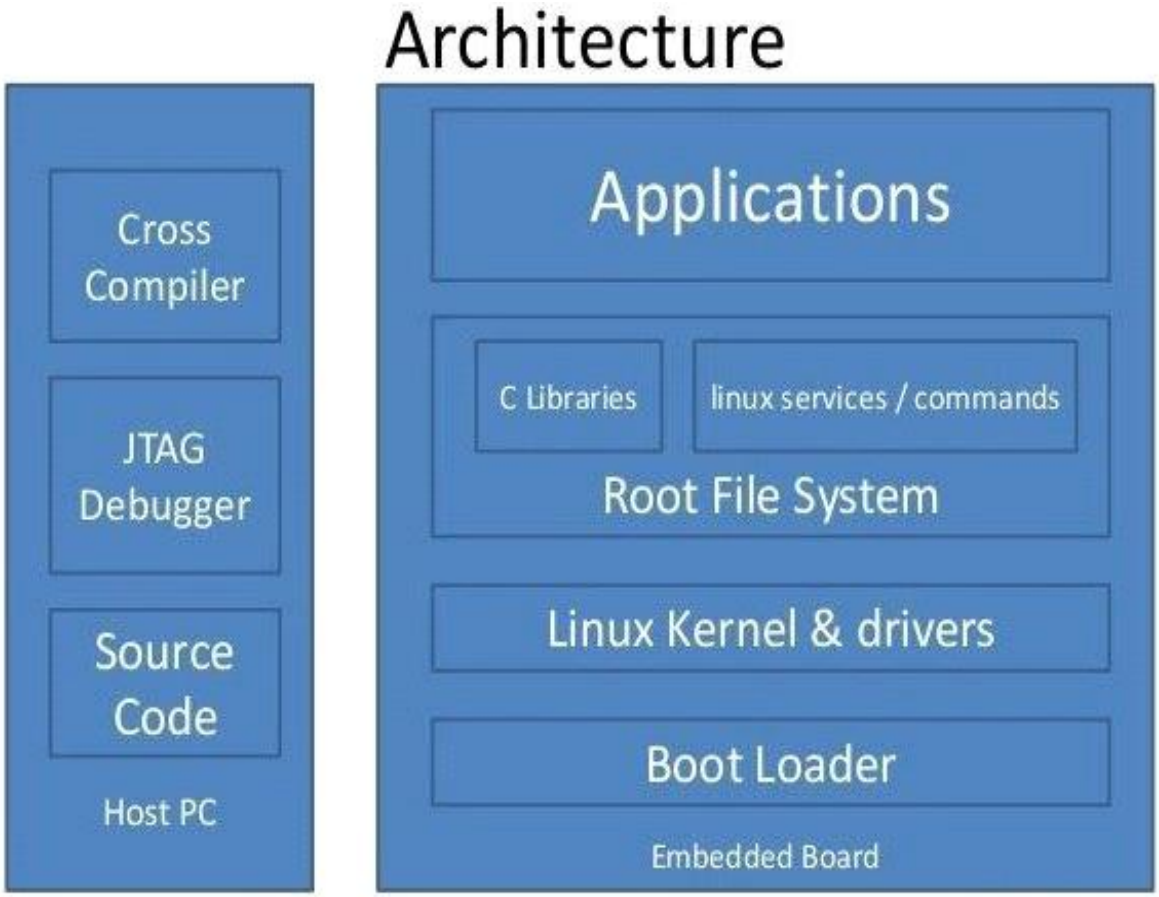


Figure 9.2 Embedded Linux Architecture

The embedded system build process is usually done on the host PC using cross compilation tools. Because target hardware does not have enough resources to run tools that are used to generate a binary image for target embedded hardware. The process of compiling code on one system (host system) and generating source code runs on the other system is known as cross-compilation [47].

9.4.1 Toolchain

A toolchain is a set of tools that compiles source code into executables that can run on the target device and includes a compiler, a linker, and run-time libraries. Initially, need one to build the other three elements of an embedded Linux system: the bootloader, the kernel, and the root filesystem. It has to be able to compile code written in assembly, C, and C ++ since these are the languages used in the base open-source packages [47].

-There are two types of toolchains:

- **Native toolchain:** This toolchain runs on the same type of system, sometimes the same actual system, as the programs it generates. This is the usual case for desktops and servers.
- **Cross toolchain:** This toolchain runs on a different type of system than the target, allowing the development to be done on a fast desktop PC and then loaded onto the embedded target for testing.

Almost all embedded Linux development is done using a cross-development toolchain, partly because most embedded devices are not well suited to program development since they lack computing power, memory, and storage, but also because it keeps the host and target environments separate [48].

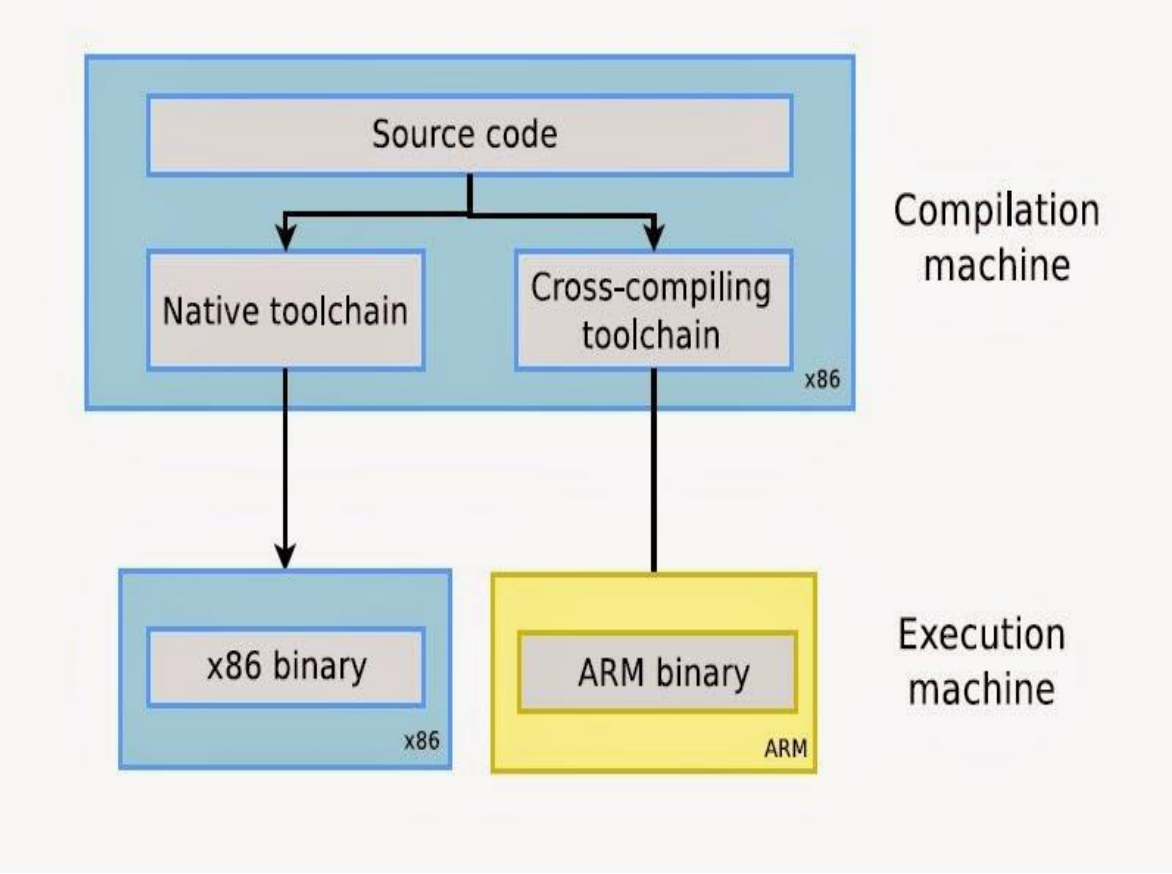


Figure 9.3 Types of Toolchains

Toolchain consists of:

- Binutils: is a set of tools to generate and manipulate binaries for a given CPU architecture. Ex: as, ld, ar, and ranlib.

- Kernel headers: The C library and compiled programs need to interact with the kernel. For example, to get available system calls and their numbers or for available data structures. That's why compiling the C library requires kernel headers, and many applications also require them.

- GCC: GNU Compiler Collection, the famous free software compiler. Can compile C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, and generate code for a large number of CPU architectures, including ARM, AVR, Black n, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86 64, IA64, and Xtensa.

- C library: The C library is a very important component of a Linux system Interface between the applications and the kernel. It provides the well-known standard C API to ease application development. Several C libraries are available: glibc, uClibc, eglibc, dietlibc, newlib, etc. The choice of the C library for our toolchain must be made at the time of the cross-compiling toolchain generation.

There are three choices for our cross-development toolchain: a ready-built toolchain that matches our needs, use the one generated by an embedded build tool, selecting a Build System, or can create one using cross tool-NG [47].

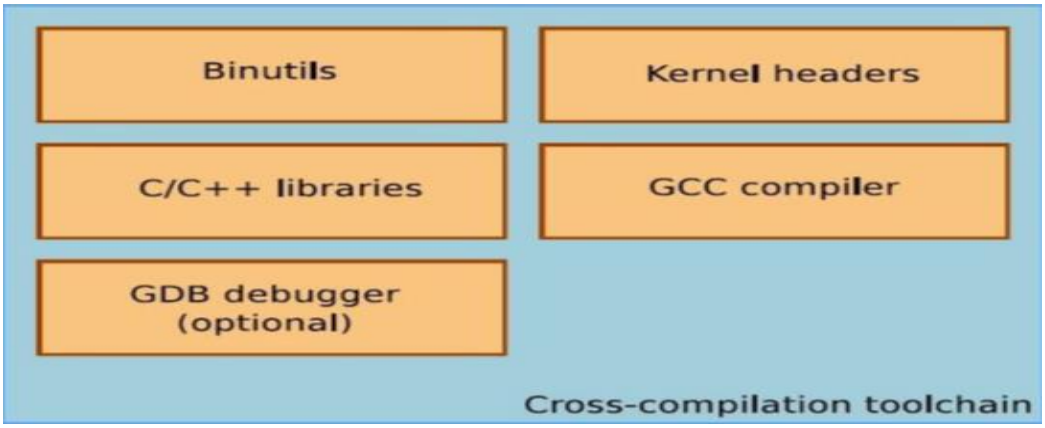


Figure 9.4 Cross-compilation toolchain

9.4.2 Bootloader

The bootloader has two main jobs: basic system initialization and the loading of the kernel. In fact, the first job is somewhat subsidiary to the second in that it is only necessary to get as much of the system working as is needed to load the kernel. When the computer is powered on, after performing some initial setup, it will load a bootloader into memory and run that code. The bootloader's main job is to find the operating system's binary program, load that binary into memory, and run the operating system [49].

In our case, this is the Linux kernel. The bootloader is done at this point, and all of its code and data in RAM are usually overwritten by the operating system. The bootloader won't run again until the computer is reset or power cycled again [49].
The bootloader in embedded systems is different from a typical laptop, desktop, or server computer. A typical PC usually boots into what we call the BIOS first and then runs Grub as the bootloader. Embedded Linux systems boot using Das-Uboot or Uboot for short as the bootloader. So, Let's Talk about boot sequence firstly for Linux machines in general then for Raspberry pi.

Linux machine booting sequence: the time between system power on and login is called the booting time, there are 6 high-level stages of a typical Linux boot process.

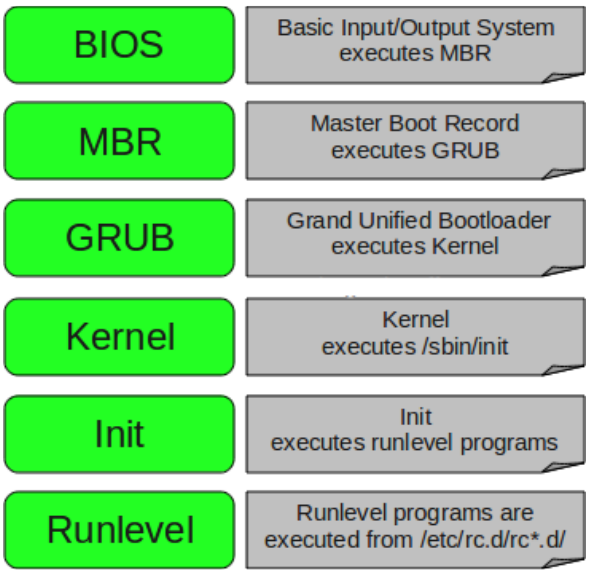


Figure 9.5 Booting Sequence

BIOS

- BIOS stands for Basic Input/Output System
- Performs some system integrity checks
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, CD-ROM, or hard drive. You can press a key during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.

So, in simple terms, BIOS loads and executes the MBR boot loader.

MBR

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk. Typically, /dev/had, or /dev/sda
- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
- It contains information about GRUB.

So, in simple terms MBR loads and executes the GRUB boot loader.

GRUB

- GRUB stands for Grand Unified Bootloader.
- GRUB is the typical boot loader for most modern Linux systems.
- If there are multiple kernel images installed on the system, we can choose which one to be executed.
- GRUB displays a splash screen, and waits for a few seconds, if we don't enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB knows about the filesystem.

So, in simple terms GRUB just loads and executes Kernel [50].

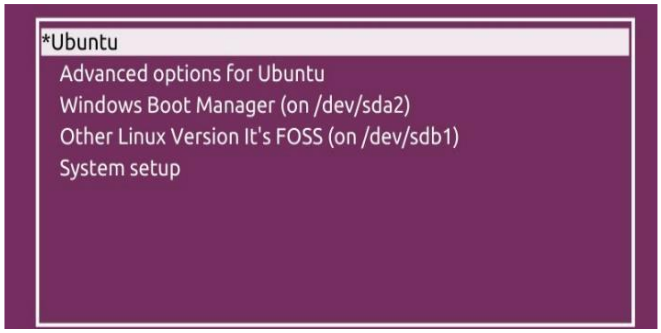


Figure 9.6 GNU GRUB

Kernel

- The kernel is often referred to as the core of any operating system, Linux included.
- It has complete control over everything in your system.
- Mounts the root file system that's specified in the grub.conf file.
- It executes the /sbin/init program, which is always the first program to be executed. You can confirm this with its process id (PID), which should always be 1.
- The kernel then establishes a temporary root file system using Initial RAM Disk (initrd) until the real file system is mounted.

Init

- System executes runlevel programs.
- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
    - o 0 – halt
    - o 1 – Single user mode
    - o 2 – Multiuser, without NFS
    - o 3 – Full multiuser mode
    - o 4 – unused
    - o 5 – X11
    - o 6 – reboot
- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate programs.
- Execute 'grep initdefault /etc/inittab' on your system to identify the default run level.
- System will then begin executing run level programs [51].

Run level programs

- When the Linux system is booting up, you might see various services getting started. For example, it might say “starting send mail …. OK”. Those are the run level programs, executed from the run level directory as defined by your run level.
- Depending on your default init level setting, the system will execute the programs from one of the following directories.
  - O Run level 0 – /etc/rc.d/rc0.d/
  - o Run level 1 – /etc/rc.d/rc1.d/
  - o Run level 2 – /etc/rc.d/rc2.d/
  - o Run level 3 – /etc/rc.d/rc3.d/
  - o Run level 4 – /etc/rc.d/rc4.d/
  - o Run level 5 – /etc/rc.d/rc5.d/
  - o Run level 6 – /etc/rc.d/rc6.d/

- Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
- Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.
- Programs that start with S are used during startup. S for startup.
- Programs start with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.

For example, S12syslog is to start the Syslog daemon, which has the sequence number 12. S80sendmail is to start the Send mail daemon, which has the sequence number 80. So, the Syslog program will be started before send mail [44].

-As explained previously the bootloader in embedded systems is different from a typical laptop, Let’s Know about the Raspberry pi boot sequence:

1. When the Raspberry Pi is first turned on, the ARM core is off, and the GPU core is on. At this point the SDRAM is disabled.
2. The GPU starts executing the first stage bootloader, which is stored in ROM on the SoC. The first stage bootloader reads the SD card loads the second stage bootloader (bootcode.bin) into the L2 cache and runs it.
3. bootcode.bin enables SDRAM, reads the third stage bootloader (loader.bin) from the SD card into RAM, and runs it.

4. loader.bin reads the GPU firmware (start.elf).
5. start.elf reads config.txt, cmdline.txt, and kernel.img [52].

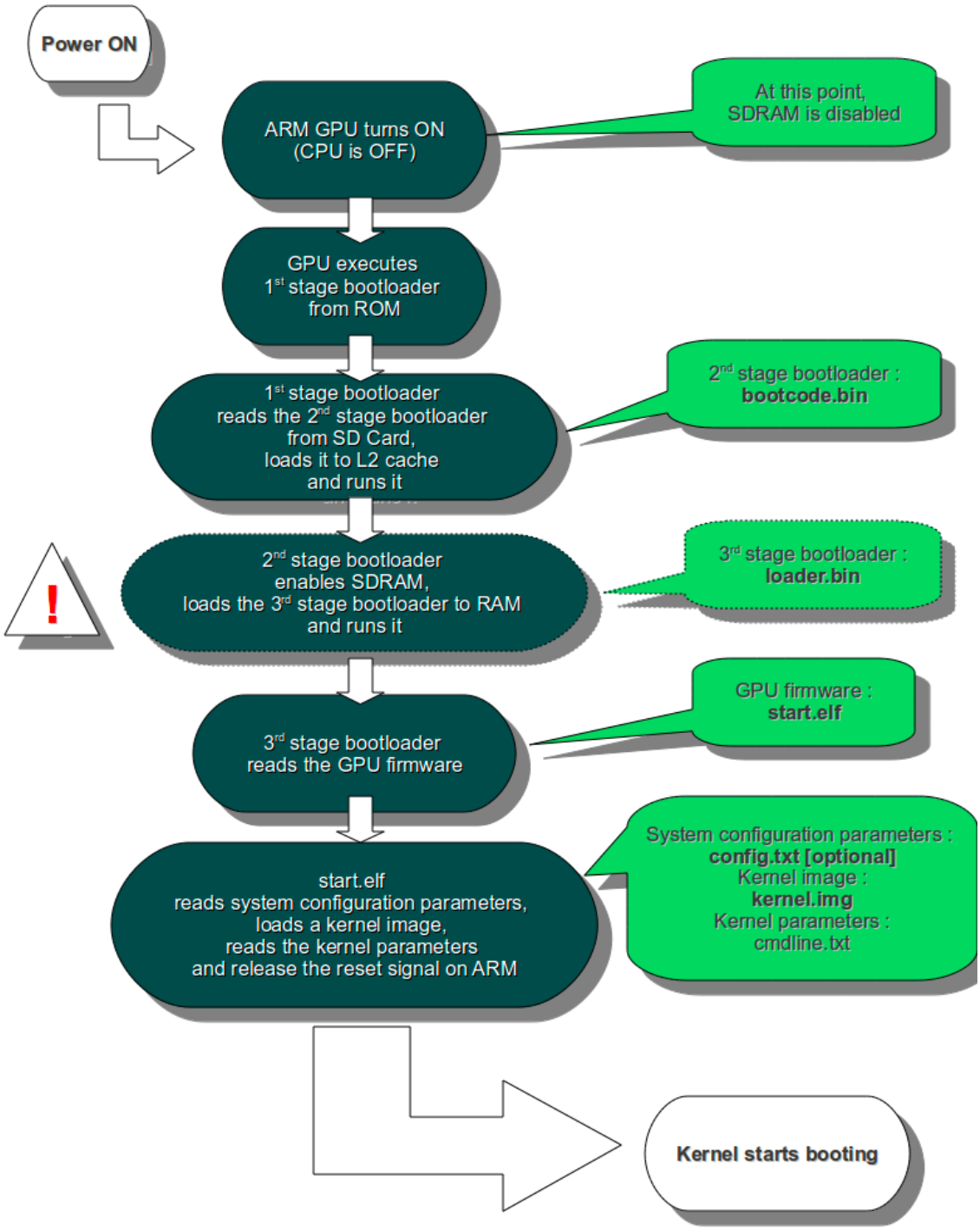


Figure 9.7 Raspberry pi boot sequence

After explaining the booting sequence, let's know how to choose a bootloader. Bootloaders come in all shapes and sizes. The kind of characteristics we want from a bootloader is that they are simple and customizable with lots of sample configurations for common development boards and devices.

The following table shows a number of them that are in general use:

Name	Architectures
Das U-Boot	ARM, Blackfin, MIPS, PowerPC, SH
Barebox	ARM, Blackfin, MIPS, PowerPC
GRUB 2	X86, X86_64
RedBoot	ARM, MIPS, PowerPC, SH
CFE	Broadcom MIPS
YAMON	MIPS

Table 9.2 Architectures of different bootloaders

9.4.3 Kernel [47]

The kernel is the component that is responsible for managing resources and interfacing with hardware, and so affects almost every aspect of our final software build. It is usually tailored to our particular hardware configuration. Once the bootloader loads the Linux kernel into memory and runs it, the Kernel will begin running its startup code. This startup code will initialize the hardware, initialize system critical data structures, initialize the scheduler, initialize all the hardware drivers, initialize filesystem drivers, mount the first filesystem, and launch the first program, among other things.

The Linux kernel's main job is to start applications and provide coordination among these applications (or programs, as they're usually called in Linux). The Linux kernel doesn't know about all programs that are supposed to run. So, the Linux kernel starts only one program and lets that program launch all the other programs that are needed. This very first program is called the init program, or sometimes just "init" for short. If the kernel can't find the init program, the kernel's purpose is gone and the kernel crashes.

The main difference in the Linux kernel for embedded systems is that it is built to run on a different CPU architecture. Otherwise, the way the kernel operates is consistent with a typical PC, which is one of its strengths.

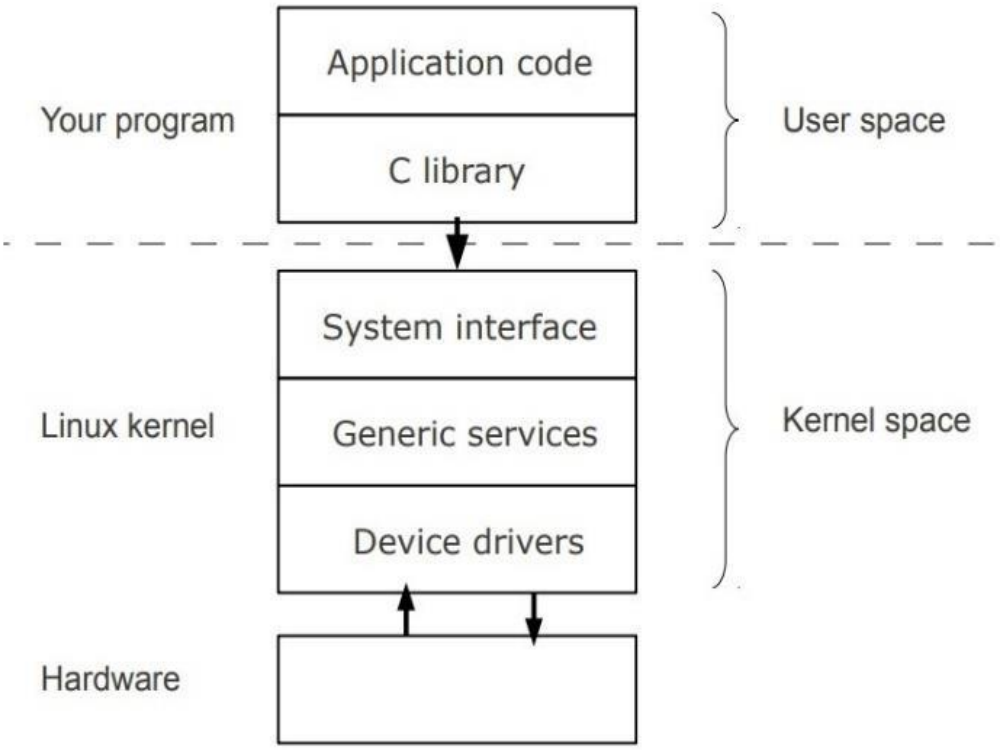


Figure 9.8 Kernel main jobs

Applications running in User space run at a low CPU privilege level. They can do very little other than making library calls. The primary interface between the User space and the Kernel space is the C library, which translates user level functions, such as those defined by POSIX, into kernel system calls. The system call interface uses an architecture specific method, such as a trap or a software interrupt, to switch the CPU from low privilege user mode to high privilege kernel mode, which allows access to all memory addresses and CPU registers.

The System call handler dispatches the call to the appropriate kernel subsystem: memory allocation calls go to the memory manager, filesystem calls to the filesystem code, and so on. Some of those calls require input from the underlying hardware and will be passed down to a device driver. In some cases, the hardware itself invokes a kernel function by raising an interrupt.

To choose the kernel for the project, Balancing the desire to always use the latest version of software against the need for vendor-specific additions and an interest in the long-term support of the code base is required.

9.4.4 Root Filesystem

In Linux, the kernel loads programs into memory separately, and the kernel expects these programs to be stored on some medium organized into files and directories. This organization of files and directories is called a filesystem. As is true of many operating systems, Linux has filesystems on media, the data actually stored on a storage medium, and filesystem drivers, the code that knows how to interpret and update the filesystem data on the medium.

In Linux, this medium is often a hard disk. However, embedded systems often don't have a hard drive, so the medium can be other hardware devices like SD cards, flash memory, or even RAM to name a few. Unlike Windows, Linux filesystems get associated with a directory rather than a drive letter. Filesystems can be associated with any directory, even one that is several layers down in a path. This associating a filesystem with a directory is called "mounting." Linux first starts with an empty directory called / (slash) [44].

During Linux startup, the top most filesystem gets associated with (or mounted to) this directory, and all the contents of that filesystem appear under /. This topmost filesystem is called the root filesystem.

Linux systems expect the root filesystem to be laid out a certain way. So, this filesystem is special and can't just be some random set of directories and files. This is where directories like bin, sbin, etc. and more come from.

The main point here is that Linux looks for this first program, this init program, to reside in the filesystem. The root filesystem needs to be created in advance and be mounted to '/' before the kernel can launch the init program. Because embedded systems have different hardware constraints, often Linux embedded systems use special filesystem formats rather than the typical EXT3, EXT4, btrfs, or xfs used on desktop or laptop computers [53].

The first objective is to create a minimal root filesystem that can give us a shell prompt. Then using that as a base we will add scripts to start other programs up, and to configure a network interface and user permissions. Knowing how to build the root filesystem from scratch is a useful skill.

-To make a minimal root filesystem, we use these components:

- init
- Shell
- Daemons
- Shared libraries
- Configuration files
- Device nodes
- Kernel modules

9.4.5 Application

When the kernel finds, loads and runs the init program, that program then is responsible for bringing up the rest of the system. The init program is also responsible for starting regular programs. These programs do have user interaction. Embedded systems often have just a few user programs, sometimes just one. In an embedded system, this set of programs makes the device do what it's supposed to do [52].

9.5 Build Systems

As explained previously, an embedded Linux system has several major components, all of which are derived from freely available source code and all of which may require varying levels of customization.

We can build individually a toolchain, a bootloader, a kernel, and a root filesystem, and then combine them into a basic embedded Linux system. It has the advantage that we are in complete control of the software, and we can tailor it to do anything we like. If we want it to do something truly odd but innovative, or if we want to reduce the memory footprint to the smallest size possible, building our own is the way to go. But, in the vast majority of situations, building manually is a waste of time and produces inferior, unmaintainable systems. The idea of a build system is to automate all the steps of building the Embedded Linux major components. A capable embedded Linux build system helps us create an embedded Linux distribution tailored to our unique requirements. This must include a cross-toolchain and all the packages required for our project. Our build system should be able to generate root file systems in our choice of binary formats, our embedded Linux kernel image with our configuration, a bootloader image, and any other necessary files and utilities so that these can be properly

deployed. There are several build systems available, each has its own ideologies and focuses on building the Linux operating system for the target. Below given are the most popular among them:

- **Buildroot**

This is a build system based on make. This is simple as the GNU build system. Lightweight build system, easy to edit and configure using menu configuration. It supports building several embedded applications with minimal foot-print.

- **Yocto / Open Embedded**

This build system is based on bit bake instead of make. Supports a huge list of packages. Supports several target boards and architectures. Supports a Layered approach to segregate and maintain the build scripts. Became de-facto build system for embedded Linux. And it is also the base for other Distributions like Peta Linux of Xilinx and RDK for broadband/video (RDK-B). Open Embedded has several derivatives which have several features like IDE support, debugging with emulators, etc [44].

- **OpenWRT / LEDE**

This build system is a popular build system to build custom distributions for networking devices like routers and access points. It is a derivative of build root. Has its own lightweight software components as a replacement for the init system(procd), the network manager(netifd), interposes communication(ubus), etc. it Has seven network- relate dated software components natively supported. It is also the base of a few other distributions like QSDK for Qualcomm network SoCs.

- **AOSP / Soong**

The Android Open-Source Project has built its own build system called Soong which is based on the blueprint (meta build system) and Kati (Modified Make) and ninja (alternative for make). Soong builds an android system with lots of Android-specific components like its own init system, HAL codes, system services, Java packages, and external opensource packages.

9.6 The Yocto Project

The Yocto Project is more general in the way it defines the target system, so it can build complex embedded devices. Every component is generated as a binary package, by default, using the RPM format, and then the packages are combined together to make the filesystem image. Furthermore, you can install a package

manager in the filesystem image, which allows you to update packages at runtime. In other words, when you build with the Yocto Project, you are, in effect, creating your own custom Linux distribution.

The Yocto Project is primarily a group of recipes, similar to Build root packages but written using a combination of Python and shell script, together with a task scheduler called Bit Bake that produces whatever you have configured, from the recipes.

The Yocto Project is structured with cross-platform tools, and metadata, to enable developers rapidly to create customized Linux distributions from source code, which simplifies the development process. Compared to a full Linux distribution, the customized system will reserve the software you need to make the system much more specific to your application [54].

Yocto Project has advantages in system and application development, archiving, and management. Developers can customize their systems in terms of speed, memory footprint, and even memory utilization. Yocto Project allows software customization and construction exchange for multiple hardware platforms, and also maintains a software stack in scale.

-The Yocto Project collects together several components, the most important of which are the following:

- OE-Core: is metadata composed of basic recipes, classes, and related files. These metadata are designed to be common in many different Open Embedded derived systems (including Yocto projects).
- Bit Bake: The core tool of the Open Embedded build system. Bit Bake plays the role of build a system engine and is responsible for parsing metadata, generating task lists from it, and then performing these tasks.
- Poky: a reference distribution. Poky is the name of the reference distribution or reference OS of the Yocto Project. Poky includes the Open Embedded Build System (Bit bake and Open Embedded-Core) and a set of metadata to help us start building our own distribution. Poky uses Open Embedded Build System to build a small embedded operating system. Poky is an integration layer on top of OE-Core. Poky provides the following:
    - A basic level of distro infrastructure to illustrate how to customize the distro.
    - A means to verify the Yocto Project components.

- Documentation: This is the user's manuals and developer's guides for each component.
- Toaster: This is a web-based interface to Bit bake and its metadata.
- ADT Eclipse: This is a plugin for Eclipse.

The Yocto Project provides a stable base, which can be used as it is or can be extended using meta layers. When downloading the build system, the Poky build 'file' is called a recipe and a layer. We can modify, assign, or anyway we need to create our own customized embedded Linux [47].

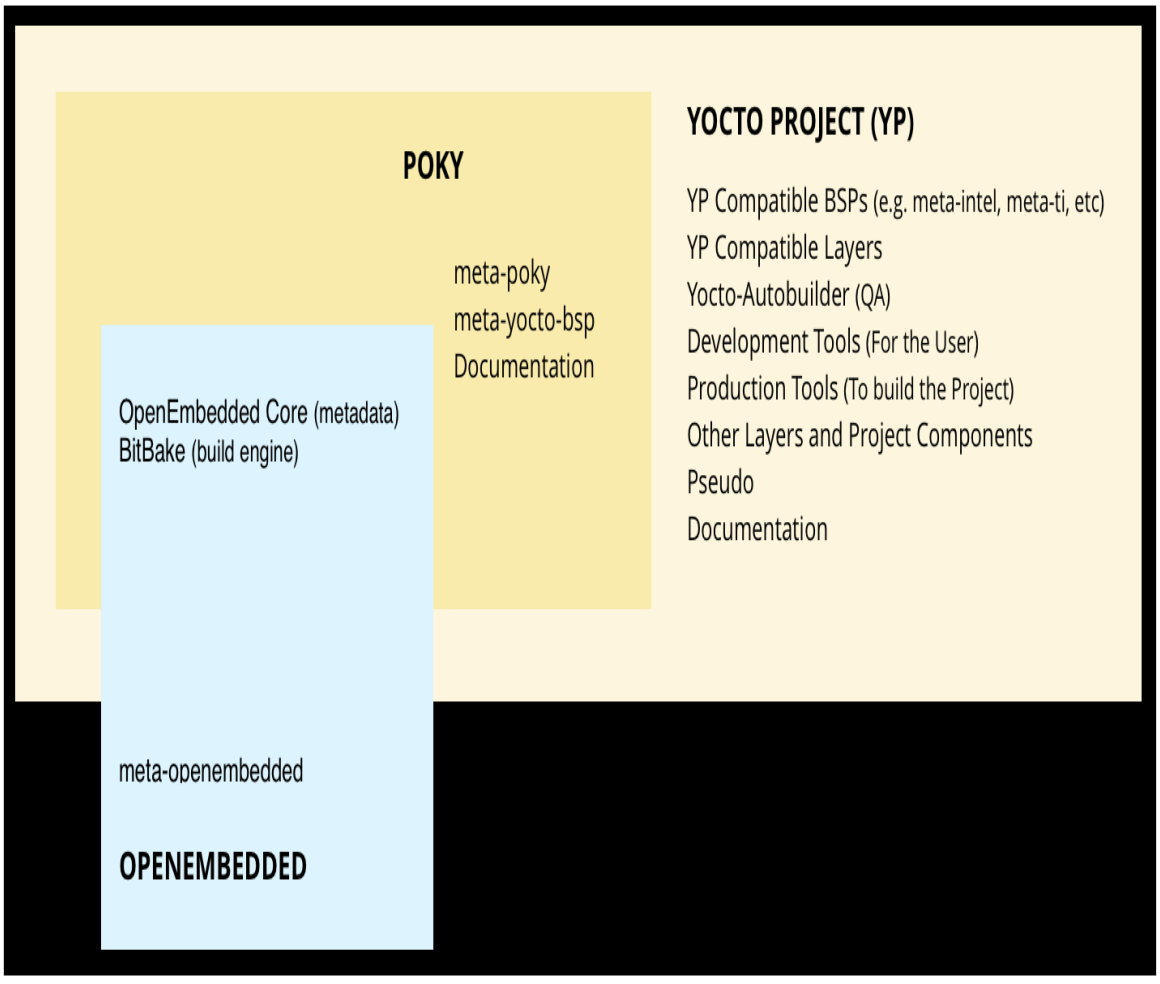


Figure 9.9 Yocto project components

-Some Yocto Project Terminology

- Recipe: The most common form of metadata. Recipe contains a list of settings and tasks used to build a binary image file. Recipe describes where you get the code and which (code) patch you need to apply. At the same time, Recipe also describes the dependencies on other recipes or

libraries, as well as configuration and compilation options. Recipe is stored in Layer.

- Layer: A collection of related recipes. A Layer is a repository containing relevant metadata that tells the Open Embedded build system how to build the target. Yocto Project's layer model promotes collaboration, sharing, customization, and reuse in the Yocto Project development environment. Layers logically separate the information of your project.

- Metadata: Layer contains the recipe files, patches, and additional files provided by the user, other information referring to the build instructions, and data that controls what and how to build. A good example of the software layer might be the meta-Qt5 Layer from the Open Embedded Layer Index.

- A Board Support Package (BSP): is a collection of information that defines how to support a particular hardware device, set of devices, or hardware platform. The BSP includes information about the hardware features present on the device and kernel configuration information along with any additional hardware drivers required.

- Configuration Files: Files that hold global definitions of user-defined variables and hardware configuration information. They tell the build system what to build and put into the image to support a particular platform.

- Packages: The output of the build system used to create your final image [54].

9.7 Summary

Embedded hardware will continue to get more complex, following the trajectory set by Moore's Law. Linux has the power and the flexibility to make use of hardware in an efficient way. Linux is just one component of open-source software out of the many that you need to create a working product.

Linux is open-source software. The code used to create Linux is free and available for the public to view, edit, and for users to contribute to it. Linux is present in the software of a large number of devices that we use on a daily basis. Linux is secure, flexible, and can receive excellent support from a large

community of users. Linux is Multitasking and Multiuser.

Embedded Linux is a type of Linux operating system/kernel that was designed to be installed and used in embedded devices or systems.

When an embedded computer starts, the Linux system will perform these steps:

- Jump into the bootloader.
- Jump into the kernel.
- Mount the root filesystem.
- Load and run init.
- Load and run background services (or daemons).
- Load and run applications.

Each of these steps invokes a component that is needed in the system.

Building embedded systems, which was a complex process earlier, has been simplified a lot by open-source build frameworks, and there exists a variety of choices and flexibilities for embedded Linux developers.

CHAPTER 10: INTERNET OF THINGS (IOT)

In this chapter, we will talk about IoT and its main applications. We will explain what the IoT Value Chain in detail is. Finally, we will discuss the MQTT Protocol in detail.

10.1 Introduction [55]

Over the past few years, IoT has become one of the most important technologies of the 21st century. Now that we can connect everyday objects (kitchen appliances, cars, thermostats, baby monitors) to the internet via embedded devices, seamless communication is possible between people, processes, and things. The Internet of Things (IoT) describes the network of physical objects(things), that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. These devices range from ordinary household objects to sophisticated industrial tools. The term “Things” in the Internet of Things refers to anything and everything in day-to-day life which is accessed or connected through the internet. As we have a platform such as a cloud that contains all the data through which we connect all the things around us.

For example, a house, where we can connect our home appliances such as air conditioner, light, etc. through each other, and all these things are managed on the same platform. Since we have a platform, we can connect our car, track its fuel meter, and speed level, and track the location of the car.

So, IoT refers to the billions of physical devices around the world that are connected to the internet all collecting and sharing data.

Figure 10.1 IoT Applications

10.2 Main categories of IoT applications [55]

The applications of IoT technologies are multiple because it is adjustable to almost any technology that can provide relevant information about its own operation, the performance of an activity, and even about the environmental conditions that we need to monitor and control at a distance.

-The main categories are:

- Human consumer
  - Human appliances, Washing machine, fridge, lights, etc.
  - Serving many devices.
  - Free of charge (Revenue from advertising).
- Smart cities
  - Roads, Lights, Parking, Sensors, etc.
  - High revenues.
- Health body
  - Medical devices, Artificial parts of the body, etc.
  - Devices require a long-life battery.
  - Serving a few devices.
- Transport/Mobility
  - Cars, trains, shoes, planes, etc.
  - Challenge: Location-based services.
  - Always connectivity (On board diagnostics board in cars connected to the internet).
- Buildings/Infrastructure

    - Access controlling (Gates), lights, cooling, air sensors, etc.
    - Observing: stability, leaning, load, tension, etc.
    - Permanent source of power (long-life battery).
- Other many applications
    - Farming
    - Wearables
    - Smart Grids
    - Water and Waste management.

10.3 IoT Value Chain

An IoT solution is formed of several building blocks or components, and each of these building blocks forms part of the IoT value chain. The IoT value chain illustrates how the different components, in combination with one another or separately, add value to the overall IoT solution and, in turn, to the end-user. Furthermore, each component is developed by a range of companies, some of which play several roles in the IoT value chain. The following components form part of the IoT value chain [56].

-IoT Value Chain Components:

- Sensor Device.
- Wireless Sensor Network.
- IoT Gateways.
- Carrier Networks.
- Application Enablement Platforms.
- Big Data Analytics Platforms.
- Device and Gateway remote management.
- End-End System Integrators.

Figure 10.2 IoT Value Chain

10.3.1 Devices

This category includes existing devices such as smart meters or vehicles in which the connectivity component has been integrated into the product design. This could also include new devices that would not have existed without IoT, such as pet trackers. Such a device must have a sensor and an actuator, as well as communications hardware, but it will also have other elements (for example, a power source such as a battery or mains electricity). In addition, depending on the type of device, it may have a screen and other ways for the user to interact with it directly (such as buttons or a keyboard) [57].

Sensors and actuators are connected to the device. Sensors are able to capture data from the environment (for example, temperature). Actuators respond to instructions and make changes in the device (for example, adjusting the temperature on a thermostat). The instructions for an actuator can come from sensors on the same device, or from other sources (for example, a thermostat can be activated by mobile phone while the homeowner is on their way home). A device can have sensors, actuators, or both. Communications hardware

enables the device to connect to the network to send the data from the sensors to the backend systems [57].

10.3.2 Wireless Sensor Network

We need to use wireless technologies with low battery consumption as we can, there is an inverse relationship between the bandwidth and the battery consumption. So, we will use small bandwidth to save the battery.

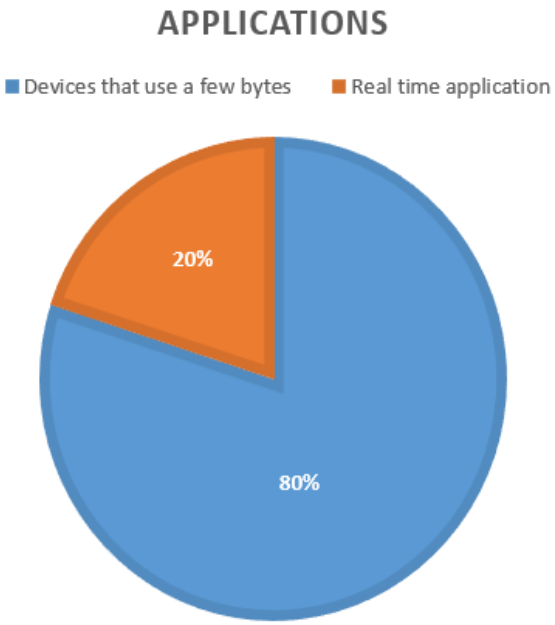


Figure 10.3 IoT Applications Bandwidth

As shown in Figure10.3, 80% of IoT application devices use a few bytes for bandwidth, and 20% of applications which Real-time applications use a huge bandwidth [58].

-Types of Wireless Technologies:

- NFC
- Bluetooth Low Energy (BLE)
- ZigBee
- Lora

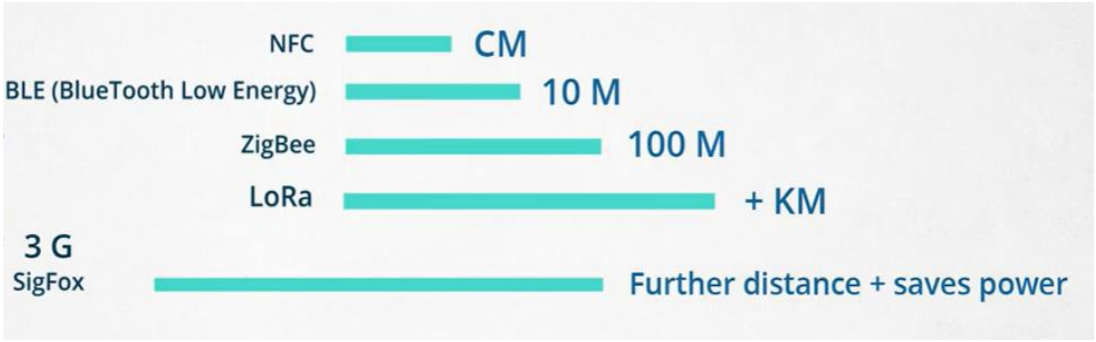


Figure 10.4 Distances of Wireless Technologies

10.3.3 Gateways [59]

Gateways Can Communicate with the devices through the different wireless technologies and get the data from the devices by it and send the data to the internet. Using Carrier Networks.

The optimum gateway will be one that supports numerous wireless technologies and carrier networks. A gateway with good Memory and CPU, have the ability to connect more device at the same time.
Gateway manufacturers are:

- CISCO
- DELL

10.3.4 Carrier Networks

Carrier Networks send the data from the gateways to the Internet.
Types of Carrier Networks:

- 3G Ethernet
- Fixed Cable (ADSL)
- SigFox
- 4G
- Iridium Next
- Narrowband IoT Satellites (NB-IoT)

-Carrier Networks Comparison:

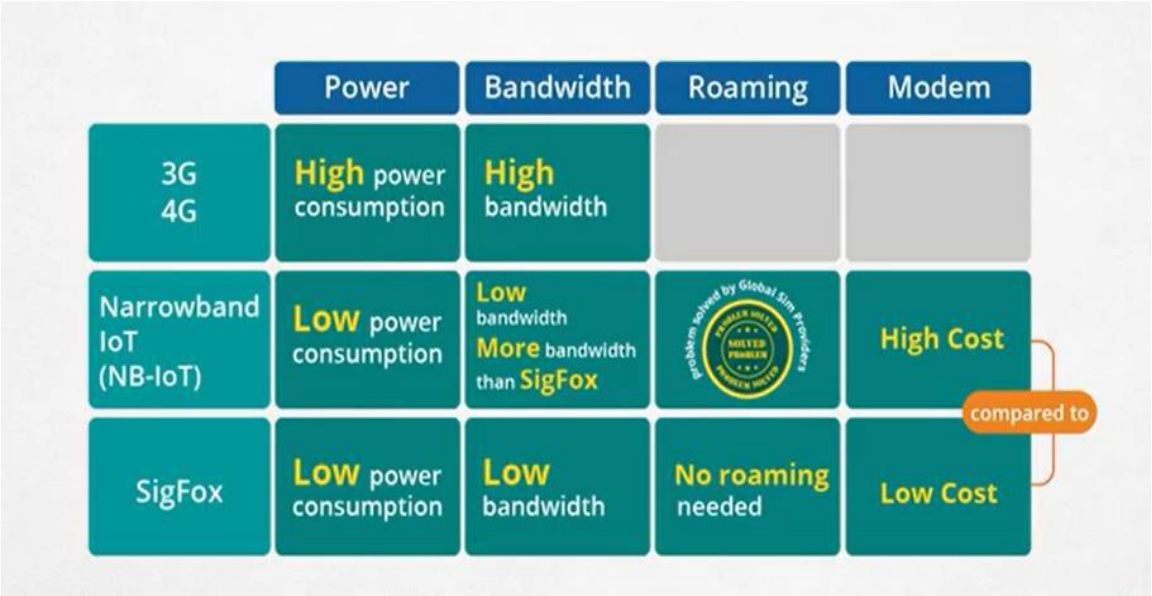


Figure 10.5 Carrier Networks Comparison

10.3.5 Application Enablement Platform

An Application Enablement Platform (AEP) can help provide a clear vision of a successful IoT solution.
There are certain components that every IoT solution needs: device management, visualizations, a rules engine, etc. Providing these crucial components are the main reason AEPs exist.

-There are many challenges interfacing the programmer [60]**:**

- Connected sensors in different domains
  - Sometimes customers ask what is the device needed?
- Connectivity and Data Collection application protocols
  - Internet application protocol on top of Wi-Fi and Ethernet: HTTP, FTP, and SMTP.
  - IoT application protocol: HTTP, MQTT, CoAP, and Modbus.
  - The challenge is when our application connects with many devices with different application protocols.
- Data storage
  - We need a huge storage increase with time (Elastic data storage).

- Data Visualization and Real-time Action execution

    - The difficulty comes from having to monitor numerous boilers and sound an alarm when they reach a certain temperature.
- Internet Based End-User Application
    - There are many Devices for the end-user: laptops, Mobile phones, Ipad, tablets, etc.
    - We can use HTML5, to be suitable for all devices.
- Other Challenges
    - Low Budget.
    - Timing.

-The AEP can solve all these challenges with:

- Reliability.
- Scalability.
- Technical Complexities are perfectly managed.
- Performance.
- Fast solutions development.

-Application Enablement Platforms Categories [44]**:**

- Ease of use & faster development (non-programmers).
- Ease of use & faster development (programmers).
- Industrial Integration.
- Database Engine Providers.
- Protocols Converters.
- Messaging Bus.
- Open Source.

10.3.6 Analytics Platforms

- Data Analytics platform: Analyze the data (Big data) to get results.
- Services Companies: Use the Big data analytics platforms to Get you it as a Service, Analyze the data and get the information of which Choice we will use.

10.3.7 Device and Gateway remote management

It helps monitor the uptime and troubleshoot problems with the devices, distribute new versions, and notify the upgrade is complete [58].

-Below are procedures for IoT device management:

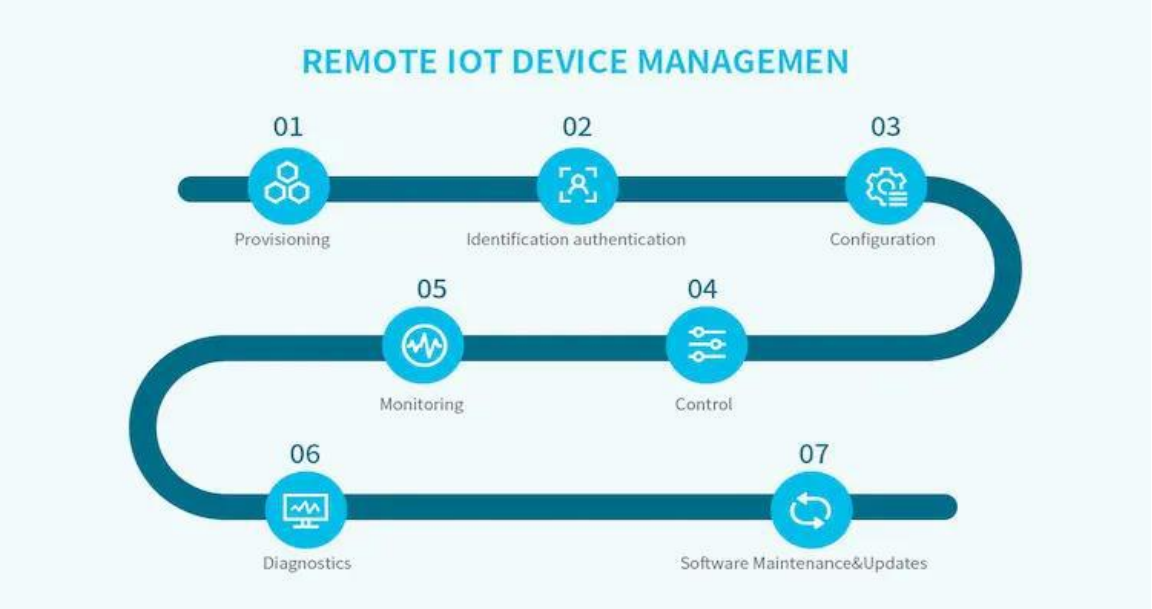


Figure 10.6 Procedures for IoT Device Management

10.3.8 End-End System Integrators

An IoT system integrator is a professional service company that offers solutions for deploying and managing Internet of Things (IoT) solutions, both hardware and software. Realizing the value of IoT requires a holistic approach to business. Even though some companies are developing capabilities within their IT and OT teams to deploy and manage internal IoT solutions themselves, most enterprises won't have the staff or the experience to select, integrate, and develop IoT solutions on their own. Hence, implementing IoT solutions requires a strong network of resources to be organized around the needs of an organization. For that reason, an ecosystem of IoT system integrators should emerge to fulfill the potential of IoT applications.

10.3.9 Compare between IoT Value Chain and Mobile Phone Value Chain

IoT Value Chain	Mobile Phone
Sensor Device with Embedded Application	The Phone with WhatsApp
Wireless Sensor Network: Lora, ZigBee, etc.	Wireless Network: Wi-Fi
IoT Gateways: Router.	Wi-Fi Access Point: Wi-Fi Router.
Carrier Networks: ADSL, 4G, Satellite, etc.	Carrier Networks: ADSL, 4G, Fiber, etc.
Application Enablement Platforms: MoT, etc.	The server of The Application: Google Play for new versions, etc.
Big Data Analytics Platforms.	
Device and Gateway remote management.	

Table 10.1 Compare betwwen IOT & Mobile phone

10.4 MQTT Protocol

MQTT (MQ Telemetry Transport) is a Client-Server publish/subscribe messaging transport protocol. It is lightweight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium. MQTT is extremely easy to implement on the client-side. Ease of use was a key concern in the development of MQTT and makes it a perfect fit for constrained devices with limited resources today.

10.4.1 MQTT Main Features and Characteristics [59]

Main Features:

- IoT messaging protocol
- Built upon (on Top of) TCP
- Minimal overhead (low cost)
- Simple DE
- Designed for reliable Communication over unreliable channels

Characteristics:

- Binary: binary protocols like MQTT are built for machines and are superefficient.
- Efficient: Smallest MQTT packet actually has two bytes.
- Bi-directional
- Data agnostic: protocol doesn't care what we're sending over it we can send like Xml Piles, JSON files, and Pictures.
- Scalable
- Built for Push Communication: we have the lowest latency we Can have as soon as the device sends data to the broker which will push data to all devices that need the data.
- Suitable for constrained devices: if we don't have a lot of Computing Power and you're very restricted with memory we can still use MQTT.

10.4.2 MQTT Publish/Subscribe Architecture

-The typical MQTT architecture can be divided into:

- Client
- Broker
- Message
- Topic

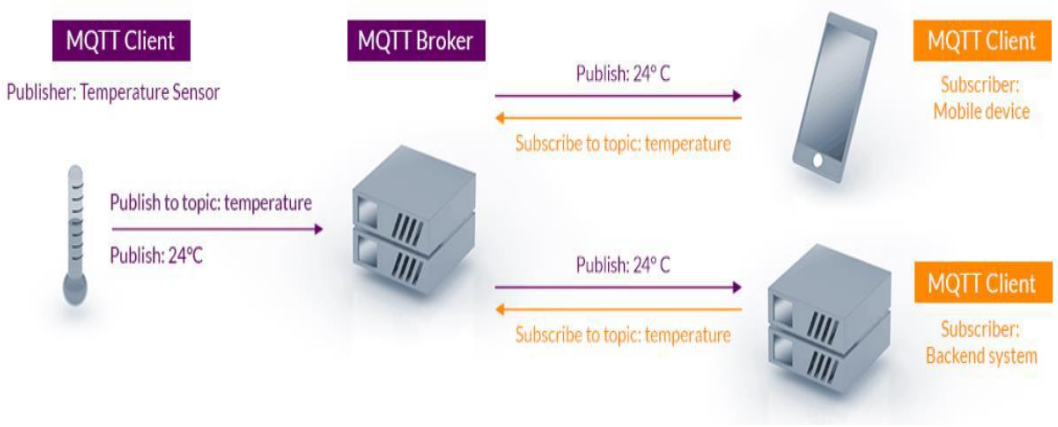


Figure 10.7 MQTT Publish/Subscribe Architecture

The publish/subscribe pattern (also known as pub/sub) provides an alternative to a traditional client-server architecture. In the client-server model, a client communicates directly with an endpoint. The pub/sub model decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers). The publishers and subscribers never contact each other directly. In fact, they are not even aware that the other exists. The connection between them is handled by a third component (the broker). The job of the broker is to filter all incoming messages and distribute them correctly to subscribers [63].

The most important aspect of pub/sub is the decoupling of the publisher of the message from the recipient (subscriber). This decoupling has several dimensions:

- **Space decoupling:** Publisher and subscriber do not need to know each other (for example, no exchange of IP address and port).
- **Time decoupling:** Publisher and subscriber do not need to run at the same time.
- **Synchronization decoupling:** Operations on both components do no need to be interrupted during publishing or receiving.

Now that we've explored the publish/subscribe model in general, let's focus on MQTT specifically.
Depending on what we want to achieve, MQTT embodies all the aspects of pub/sub that we've mentioned:

MQTT decouples the publisher and subscriber spatially. To publish or receive messages, publishers and subscribers only need to know the hostname/IP and

port of the broker. MQTT decouples by time. Although most MQTT use cases deliver messages in near-real time, if desired, the broker can store messages for clients that are not online. (Two conditions must be met to store messages: the client had connected with a persistent session and subscribed to a topic with a Quality of Service greater than 0).

MQTT works asynchronously. Because most client libraries work asynchronously and are based on callbacks or a similar model, tasks are not blocked while waiting for a message or publishing a message. In certain use cases, synchronization is desirable and possible. To wait for a certain message, some libraries have synchronous APIs. But the flow is usually asynchronous. Another thing that should be mentioned is that MQTT is especially easy to use on the client side.

Most pub/sub systems have the logic on the broker-side, but MQTT is really the essence of pub/sub when using a client library and that makes it a lightweight protocol for small and constrained devices. MQTT uses subject-based filtering of messages. Every message contains a topic (subject) that the broker can use to determine whether a subscribing client gets the message or not [62].

10.4.3 MQTT Client

Both publishers and subscribers are MQTT clients. The publisher and subscriber labels refer to whether the client is currently publishing messages or subscribed to receive messages (publish and subscribe functionality can also be implemented in the same MQTT client). An MQTT client is any device (from a microcontroller up to a full-fledged server) that runs an MQTT library and connects to an MQTT broker over a network.

10.4.4 MQTT Broker

The broker is at the heart of any publish/subscribe protocol. Depending on the implementation, a broker can handle up to millions of concurrently connected MQTT clients. The broker is responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the message to these subscribed clients. The broker also holds the session data of all clients that have persistent sessions, including subscriptions and missed messages.

Another responsibility of the broker is the authentication and authorization of clients. Usually, the broker is extensible, which facilitates custom authentication, authorization, and integration into backend systems [61].

10.4.5 MQTT Connection

The MQTT protocol is based on TCP/IP. Both the client and the broker need to have a TCP/IP stack.

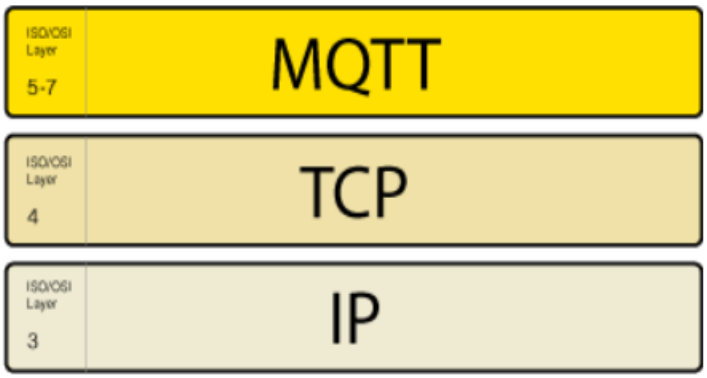


Figure 10.8 TCP/IP Stack

The MQTT connection is always between one client and the broker. Clients never connect to each other directly. To initiate a connection, the client sends a CONNECT message to the broker. The broker responds with a CONNACK message and a status code. Once the connection is established, the broker keeps it open until the client sends a disconnect command or the connection breaks [63].

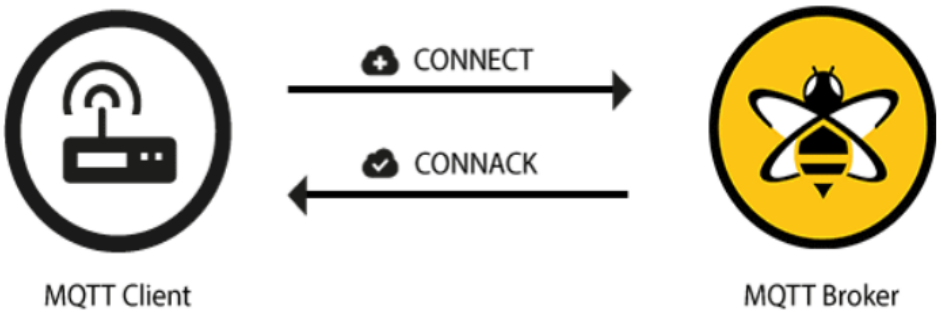


Figure 10.9 MQTT Connection

10.4.6 MQTT Messages

MQTT CONNECT command message:
To initiate a connection, the client sends a command message to the broker

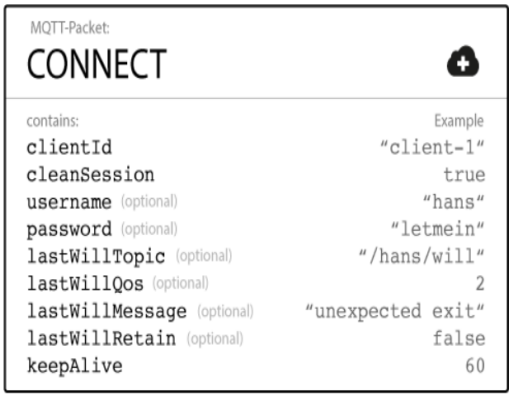


Figure 10.10 CONNECT Packet

CONNACK message: is the connect acknowledge flag. This flag contains a return code that tells the client whether the connection attempt was successful or not.

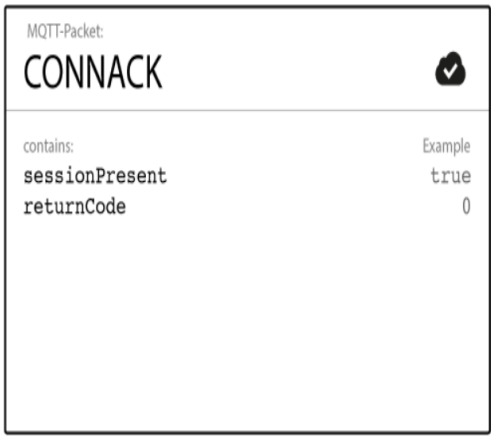


Figure 10.11 CONNACK Packet

PUBLISH message: Each message must contain a topic that the broker can use to forward the message to interested clients. Typically, each message has a payload that contains the data to transmit in byte format.

Figure 10.12 PUBLISH Packet

SUBSCRIBE message: To receive messages on topics of interest, the client sends a subscribe message to the MQTT broker. This subscribe message is very simple, it contains a unique packet identifier and a list of subscriptions.

Figure 10.13 SUBSCRIBE Packet

SUBACK message: To confirm each subscription, the broker sends a suback acknowledgment message to the client. This message contains the packet identifier of the original Subscribe message (to clearly identify the message) and a list of return codes.

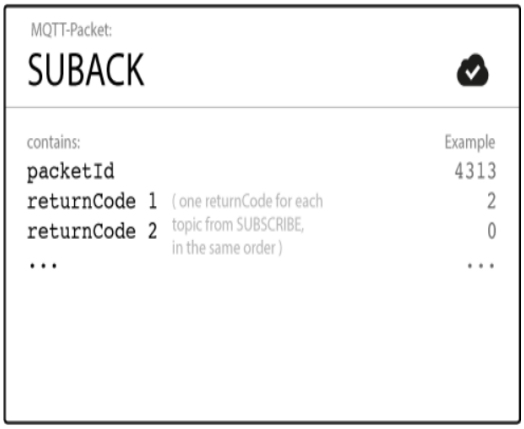


Figure 10.14 SUBACK Packet

UNSUBSCRIBE message: This message deletes existing subscriptions of a client on the broker.

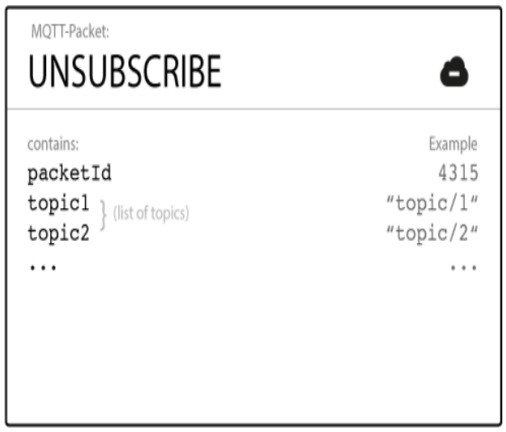


Figure 10.15 UNSUBSCRIBE Packet

UNSUBACK message: To confirm the unsubscribe, the broker sends an unsuback acknowledgment message to the client.

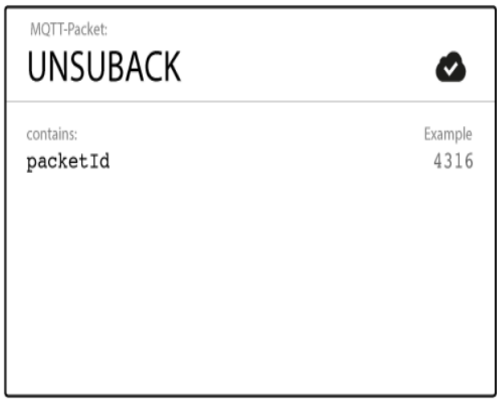


Figure 10.16 UNSUBACK Packet

10.4.7 MQTT Topics [63]

The MQTT broker uses the topic of a message to decide which client receives which message. In MQTT, the word topic refers to a UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

In comparison to a message queue, MQTT topics are very lightweight. The client does not need to create the desired topic before they publish or subscribe to it. The broker accepts each valid topic without any prior initialization.

Figure 10.17 MQTT Topic

When a client subscribes to a topic, it can subscribe to the exact topic of a published message, or it can use wildcards to subscribe to multiple topics simultaneously. A wildcard can only be used to subscribe to topics, not to publish a message. There are two different kinds of wildcards: single-level and multi-level.

10.5 Summary

IoT refers to the billions of physical devices around the world that are connected to the internet all collecting and sharing data.

The IoT value chain explains the building blocks of IoT, how value is created, who the players are, and how they interact with each other to deliver value. Looking at the IoT value chain as a pyramid, at the base is all the connected devices:

phones, fitness bands, connected cars, smart homes, and other devices on the consumer side; in industry, you have things like building sensors, smart cities, and connected factories, for example.

Stepping up a level from the base brings in the network and connectivity—how devices are connected and communicate. It's also where service providers collect device and network data and upload it to the cloud.

Finally, at the top of the value chain, are applications and services that are closest to the eventual end users—enterprises and consumers.

MQTT is an extremely lightweight and publish-subscribe messaging transport protocol. It is a publish and subscribe system where we can publish and receive the messages as a client. It makes it easy for communication between multiple devices.

It is a simple messaging protocol designed for constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications.

CHAPTER 11: EMBEDDED LINUX & INTERNET OF THINGS (IOT) TASKS

In the preceding chapters, we Explained what Embedded Linux Systems is and what is IoT, now it is time to build the application related to their meaning. The application will be divided into two main categories, Device application and the Application Enablement platform, in this chapter we will Explain the device application and the next chapter will explain the Application Enablement platform.

11.1 Introduction

The Device application is divided into four main Tasks:

- Patients interface with the Embedded Linux development board to start the measurement.
- The Communication between the Embedded Linux development board and the master controller to get the measurement results.
- Publishing the Measurement results on the MQTT Broker.
- The first car asks the Second car to move to a specific location.

we will use the Raspberry pi as my Embedded Linux development board to Build the application.

11.2 Raspberry Pi

The Raspberry Pi is a credit-card-sized fully functioning computer (System on chip) created by the non-profit Raspberry Pi Foundation in the UK. It's considered as a SOC (system on chip) device that runs on a Linux OS specially designed for it, called Raspbian.

Raspbian is the official OS for Raspberry Pi, whereas other third-party OSes like Firefox OS, Android, RISC OS, and Ubuntu Mate. can be installed on Pi. Like a computer, it has memory, processor, USB ports, audio output, and a graphic

driver for HDMI output [52].

The Raspberry Pi is amazing at two levels—the advanced functionality that you get in a credit card-sized SBC (Single Board Computer) and its price. Even with today's Pi competitors, the Raspberry Pi reigns supreme because few can beat its price. Further, it enjoys great software and community support.

I used Raspberry Pi 3B+ for The Project and its specifications are:

- SoC: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- GPU: Broadcom Videocore-IV
- RAM: 1GB LPDDR2 SDRAM
- Networking: Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
- Bluetooth: Bluetooth 4.2, Bluetooth Low Energy (BLE)
- Storage: Micro-SD
- GPIO: 40-pin GPIO header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- Dimensions: 82mm x 56mm x 19.5mm, 50g

One of the most widely used programming languages on the Raspberry Pi is Python. Python is a high-level, general-purpose, very popular programming language, Interpreted, Platform Independent, Free, Open Source, and supports many libraries.

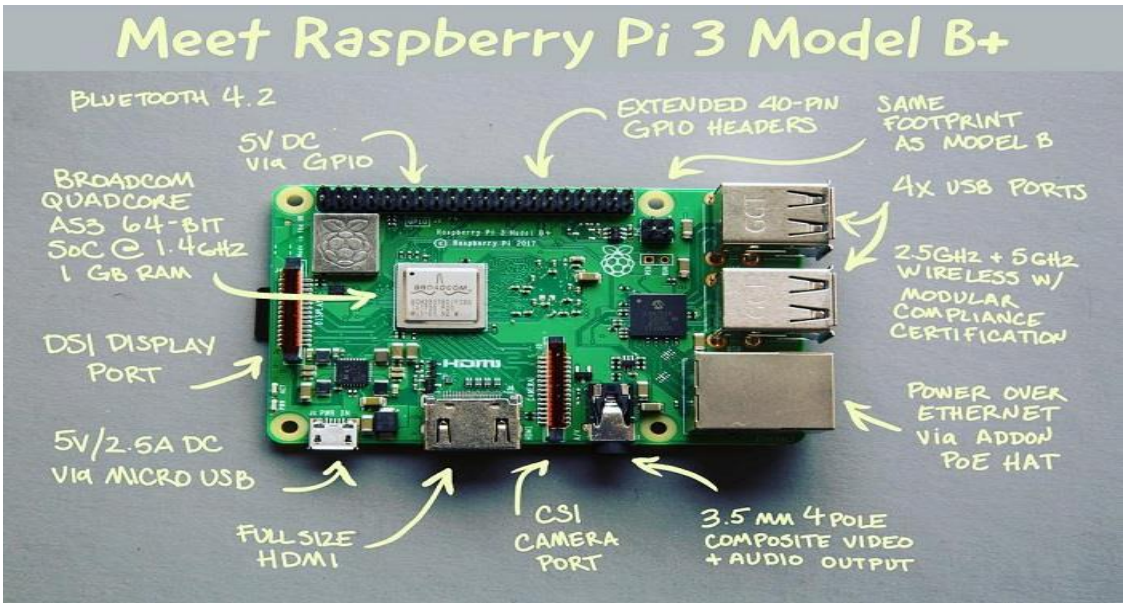
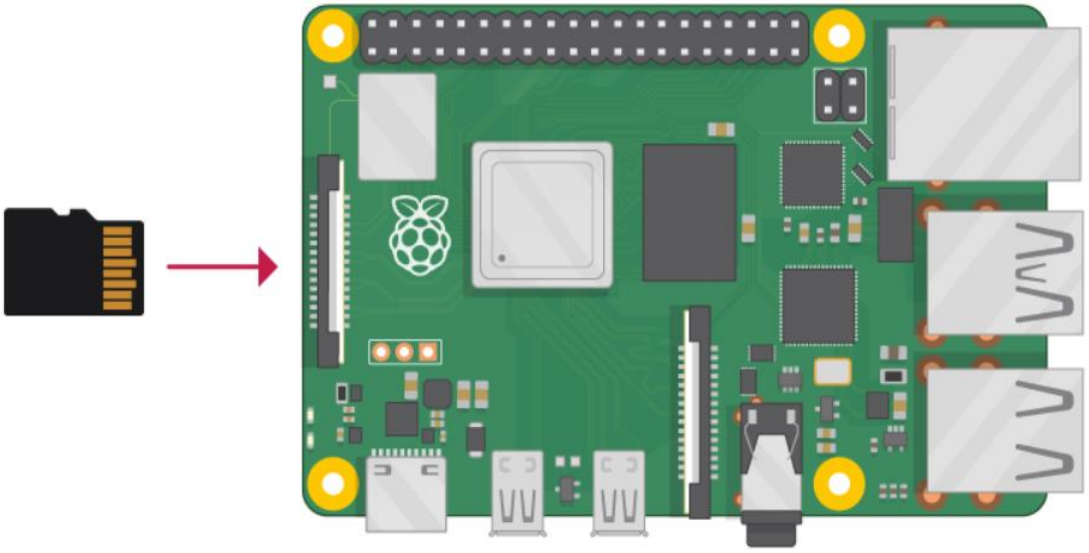


Figure 11.1 Raspberry Pi 3B+

First, before starting to make the tasks, you must do the following first [64]:

A microSD card

Your Raspberry Pi needs an SD card to store all its files and the Raspberry Pi OS operating system.



You need a microSD card with a capacity of at least 8GB.

Many sellers supply SD cards for Raspberry Pi that are already set up with Raspberry Pi OS and ready to go.

A keyboard and a mouse

To start using your Raspberry Pi, you need a USB keyboard and a USB mouse. Once you've set up your Raspberry Pi, you can use a Bluetooth keyboard and mouse, but you'll need a USB keyboard and mouse for the first setup.

A TV or computer screen

To view the Raspberry Pi OS desktop environment, you need a screen, and a cable to link the screen and your Raspberry Pi. The screen can be a TV or a computer monitor. If the screen has built-in speakers, Raspberry Pi is able to use these to play sound.

HDMI

Your Raspberry Pi has an HDMI output port that is compatible with the HDMI port of most modern TVs and computer monitors. Many computer monitors may also have DVI or VGA ports.

Raspberry Pi 4 has two micro-HDMI ports, allowing you to connect two separate monitors.
But Raspberry Pi 1, 2, and 3 have a single full-size HDMI port, so you can connect them to a screen using a standard HDMI to HDMI cable. has one micro-HDMI ports, allowing you to connect one separate monitors.

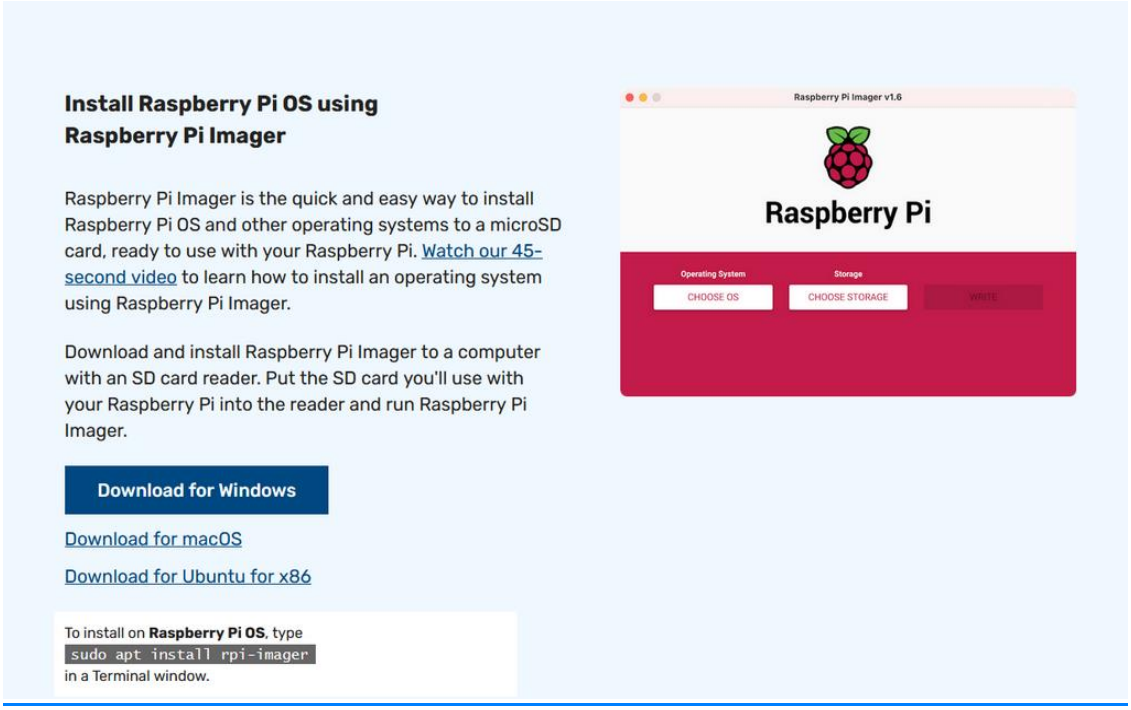
- **Set up your SD card**

If you have an SD card that doesn't have the Raspberry Pi OS operating system on it yet, or if you want to reset your Raspberry Pi, you can easily install Raspberry Pi OS yourself. To do so, you need a computer that has an SD card port-most laptop and desktop computers have one.

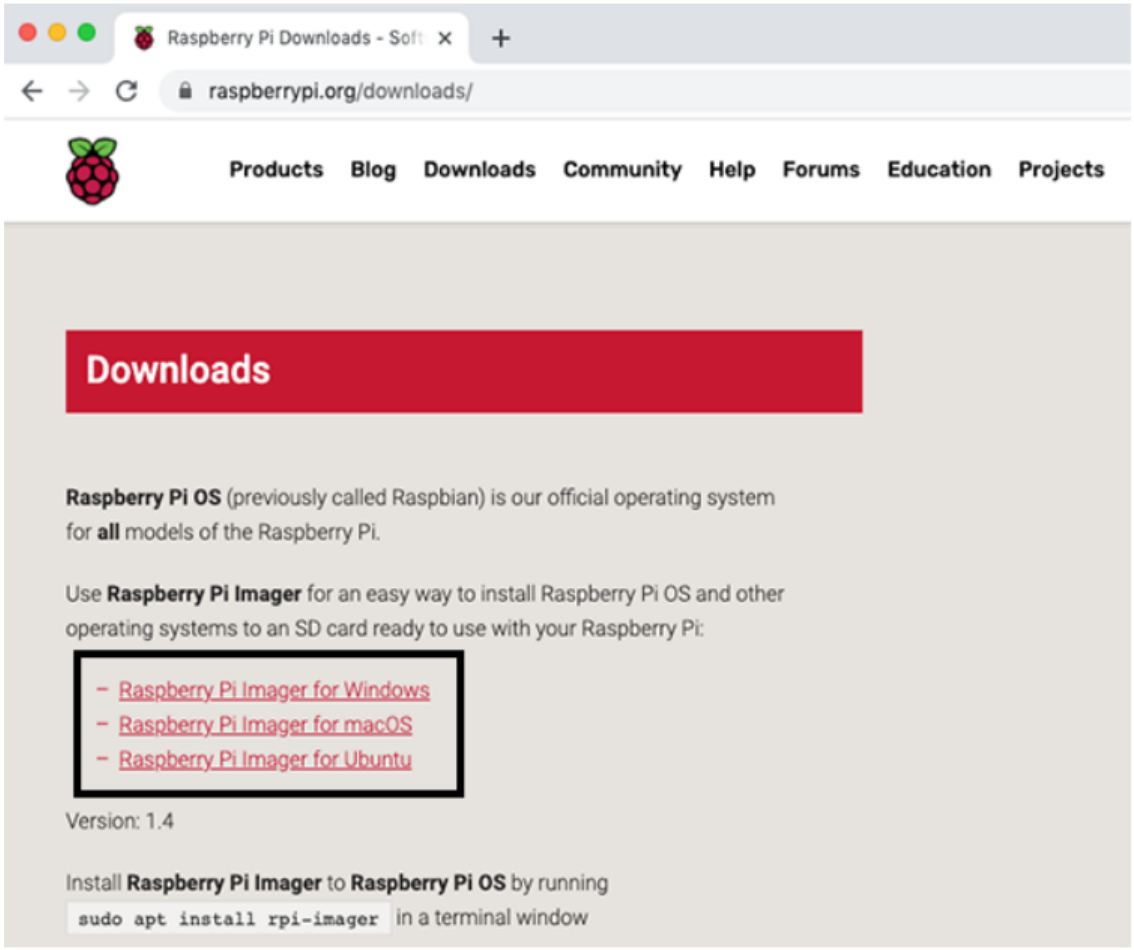
The Raspberry Pi OS operating system via the Raspberry Pi Imager
Using the Raspberry Pi Imager is the easiest way to install Raspberry Pi OS on your SD card.

Note: More advanced users looking to install a particular operating system should use this guide to installing operating system images.
Download and launch the Raspberry Pi Imager

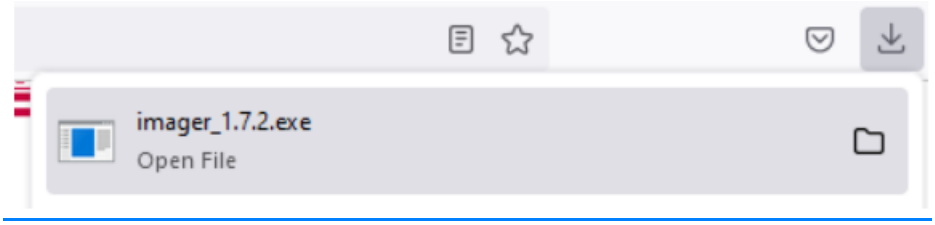
Visit the Raspberry Pi downloads page



- Click on the link for the Raspberry Pi Imager that matches your operating system



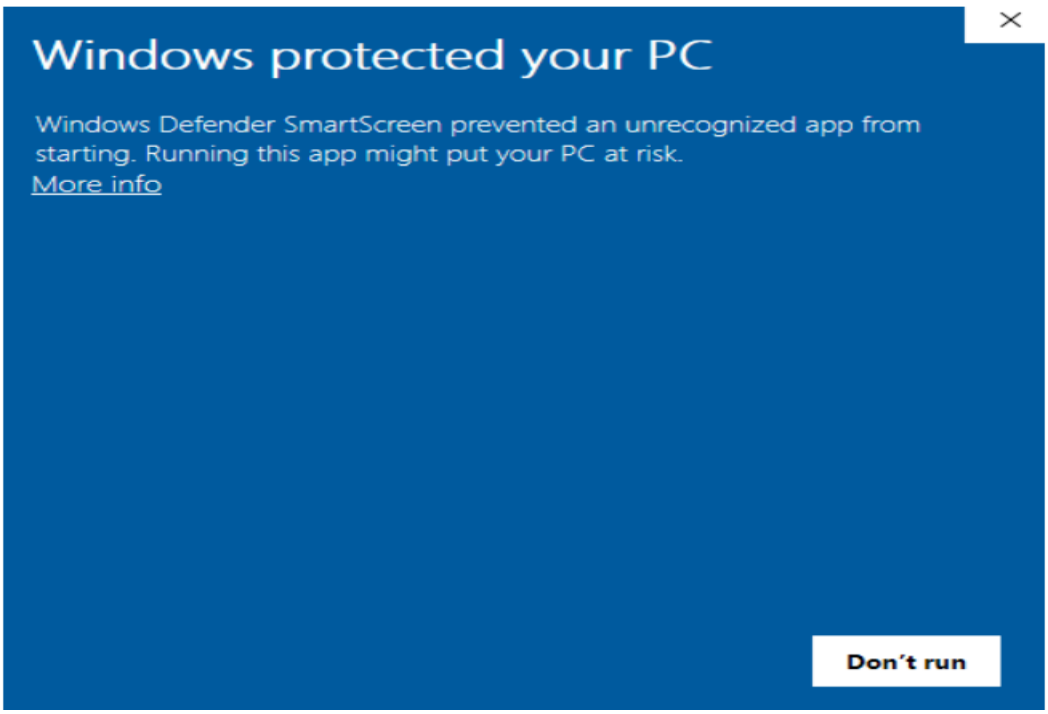
- When the download finishes, click it to launch the installer



Using the Raspberry Pi Imager

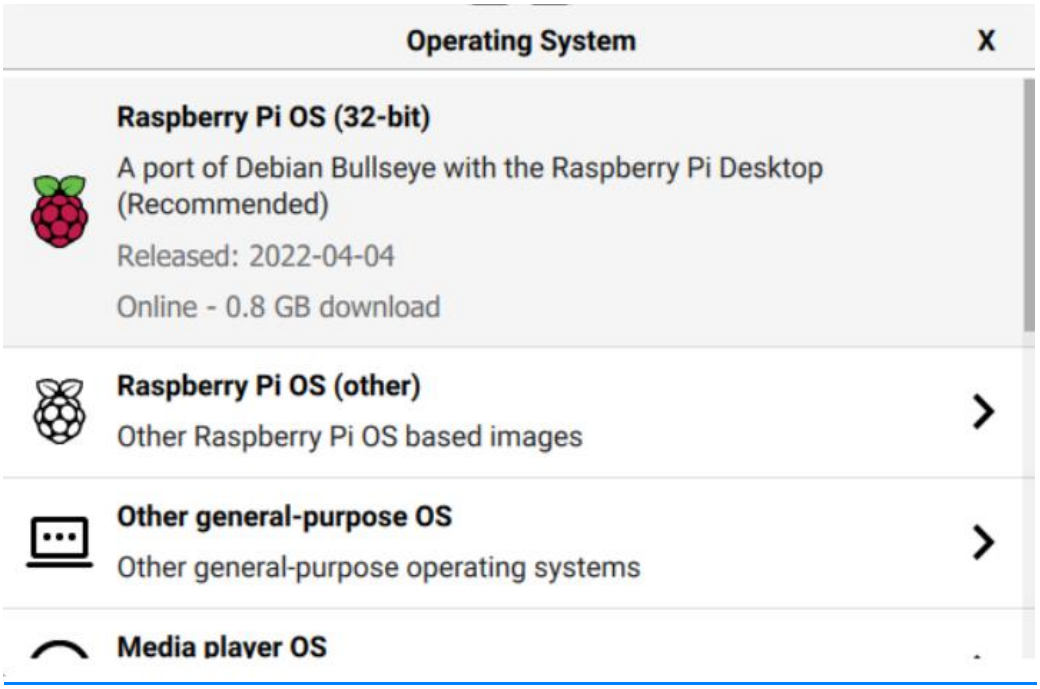
Anything that's stored on the SD card will be overwritten during formatting. If your SD card currently has any files on it, e.g. from an older version of Raspberry Pi OS, you may wish to back up these files first to prevent you from permanently losing them.

When you launch the installer, your operating system may try to block you from running it. For example, on Windows I receive the following message:
Windows warning



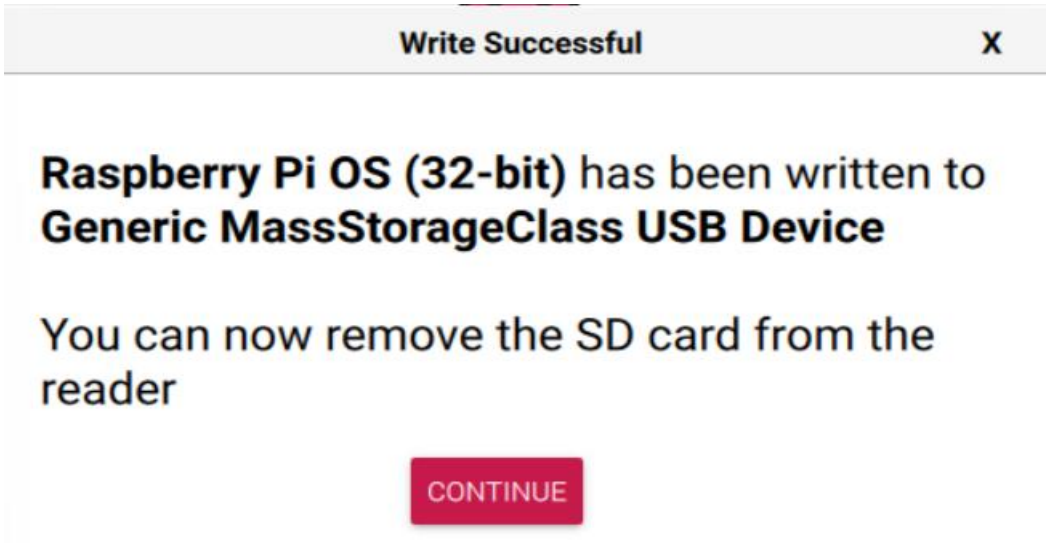
- If this pops up, click on More info and then Run anyway
- Follow the instructions to install and run the Raspberry Pi Imager
- Insert your SD card into the computer or laptop SD card slot
- In the Raspberry Pi Imager, select the OS that you want to install and the SD card you would like to install it on.

Note: You will need to be connected to the internet the first time for the Raspberry Pi Imager to download the OS that you choose. That OS will then be stored for future offline use. Being online for later uses means that the Raspberry Pi imager will always give you the latest version.


Operating System
X
Raspberry Pi OS (32-bit)
A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended)
Released: 2022-04-04
Online - 0.8 GB download
Raspberry Pi OS (other)
Other Raspberry Pi OS based images
Other general-purpose OS
Other general-purpose operating systems
Media player OS

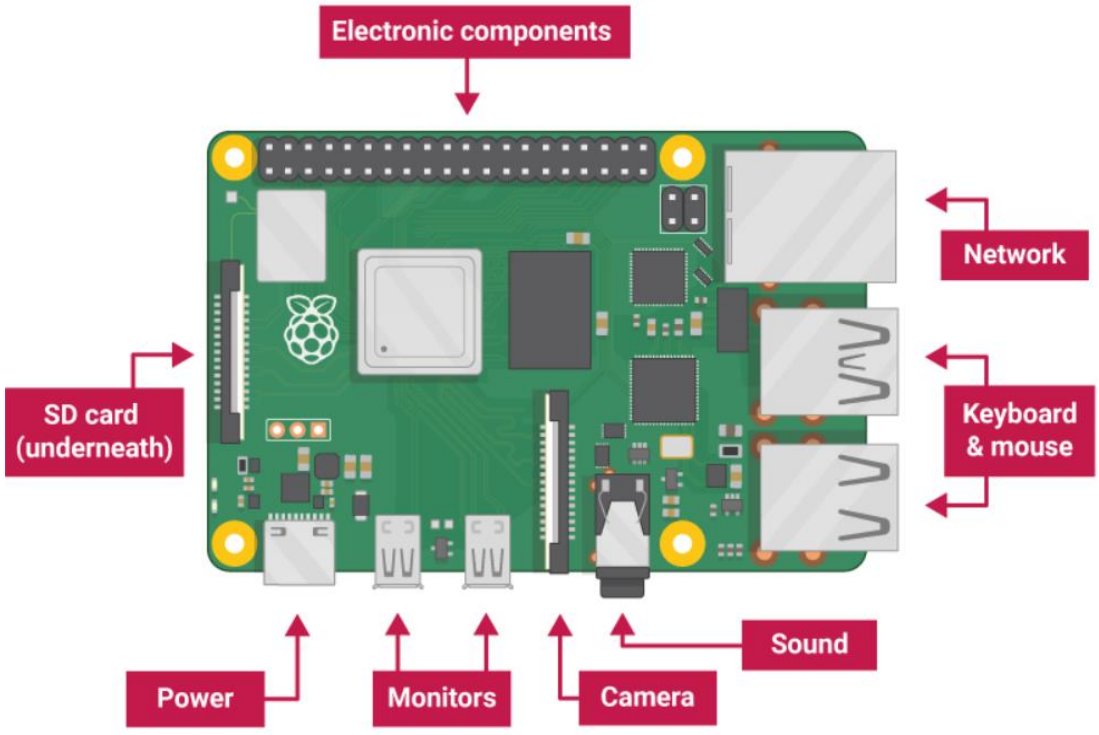

Raspberry Pi Imager v1.7.2
Raspberry Pi
Operating System
Storage
RASPBERRY PI OS (32-BIT)
GENERIC MASSST...
WRITE

- Then simply click the WRITE button
- Wait for the Raspberry Pi Imager to finish writing
- Once you get the following message, you can eject your SD card

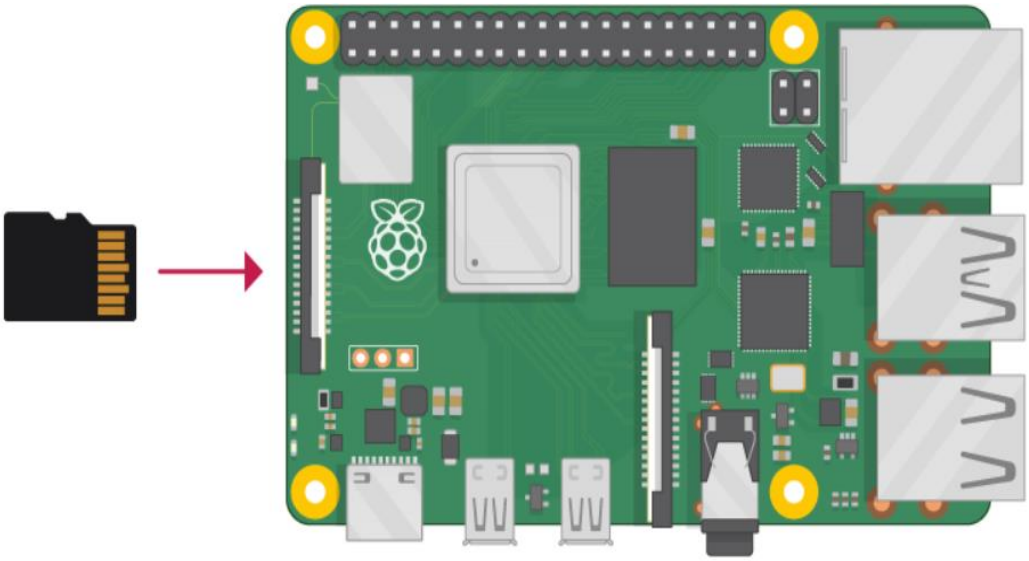


Connect your Raspberry Pi

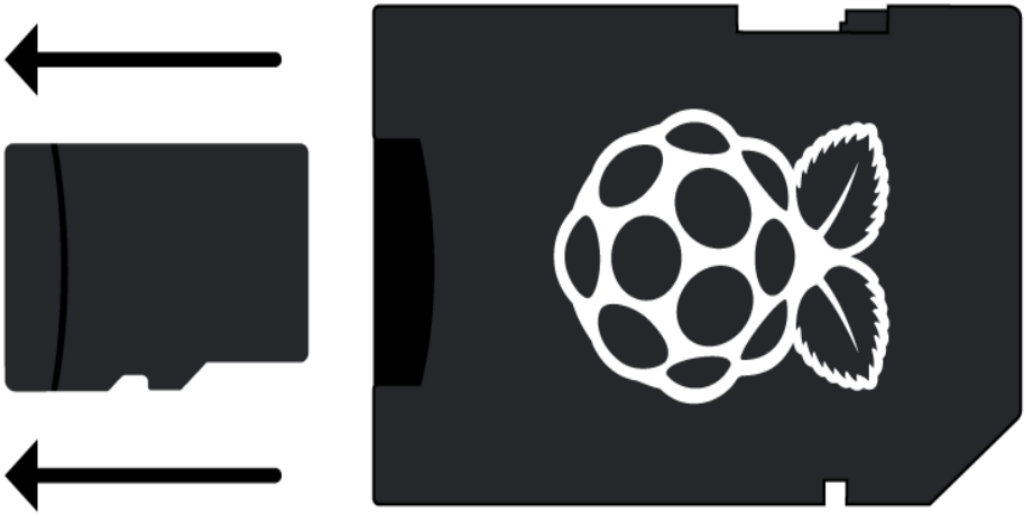
Now get everything connected to your Raspberry Pi. It's important to do this in the right order, so that all your components are safe.



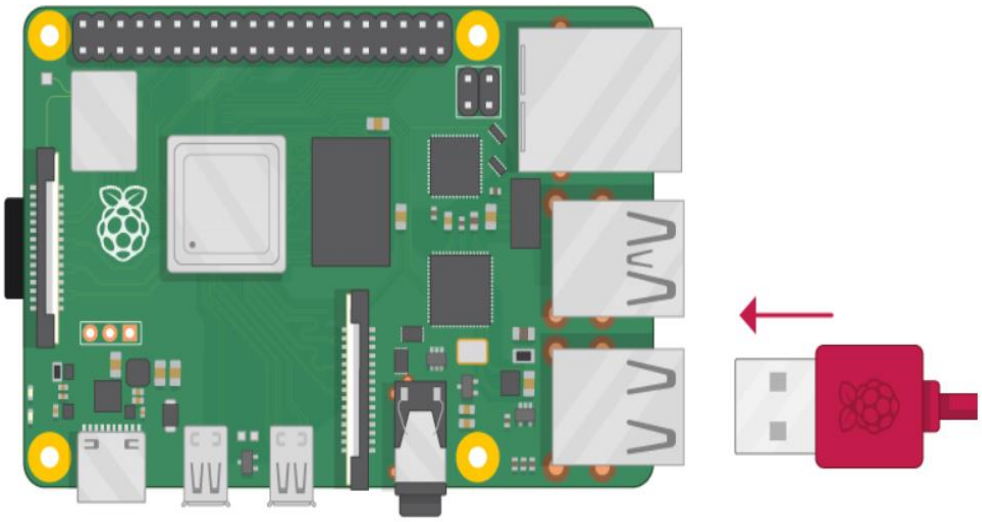
- Insert the SD card you've set up with Raspberry Pi OS into the microSD card slot on the underside of your Raspberry Pi.



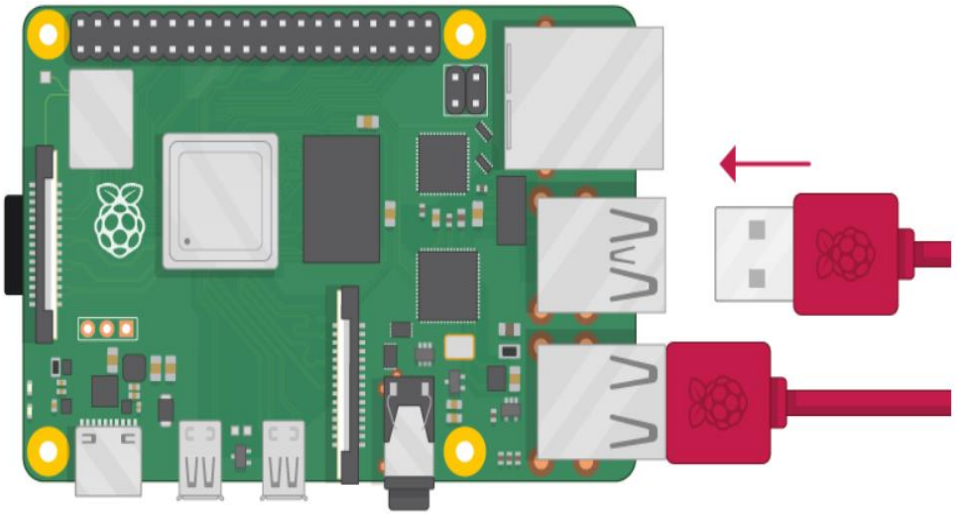
Note: Many microSD cards come inside a larger adapter — you can slide the smaller card out using the lip at the bottom.



- Find the USB connector end of your mouse's cable, and connect the mouse to a USB port on Raspberry Pi (it doesn't matter which port you use).



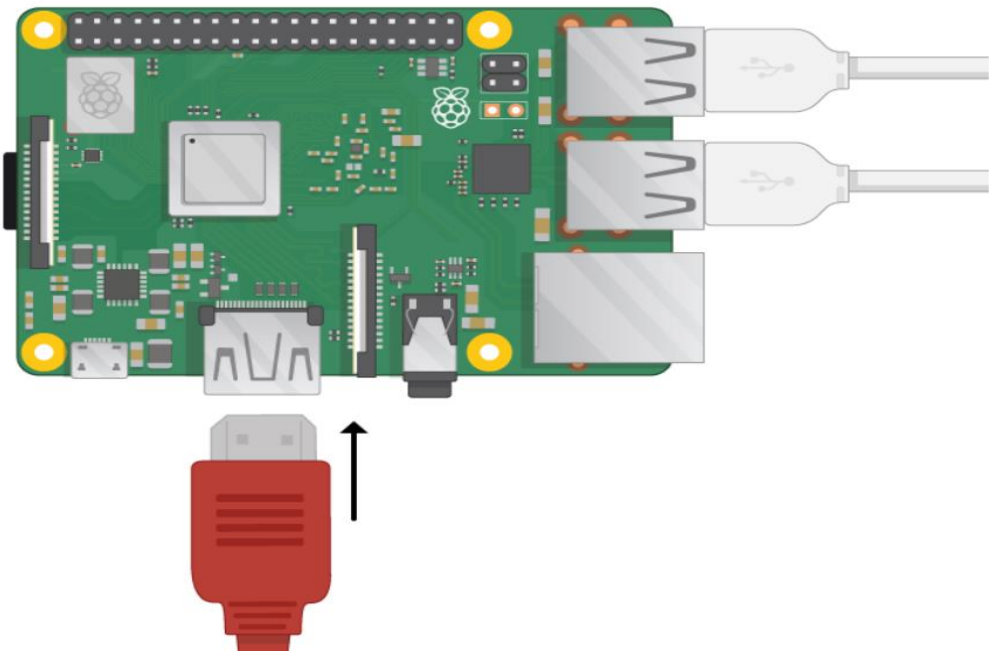
- Connect the keyboard in the same way.



- Make sure your screen is plugged into a wall socket and switched on.
- Look at the HDMI port(s) on your Raspberry Pi — notice that they have a flat side on top.
- Use a cable to connect the screen to Raspberry Pi's HDMI port — use an adapter if necessary.

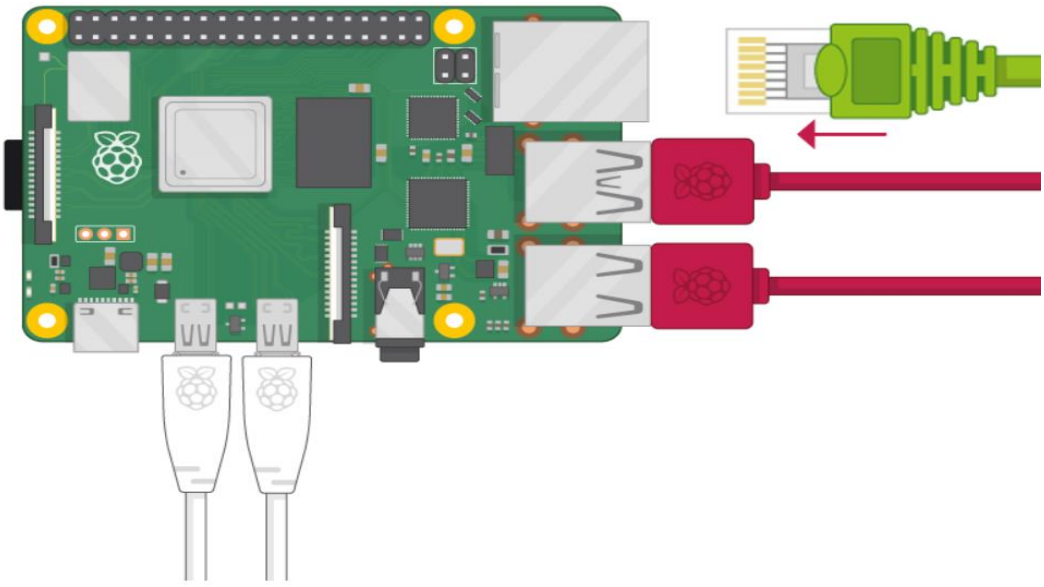
Raspberry Pi 3

- Connect your screen to the single HDMI port.

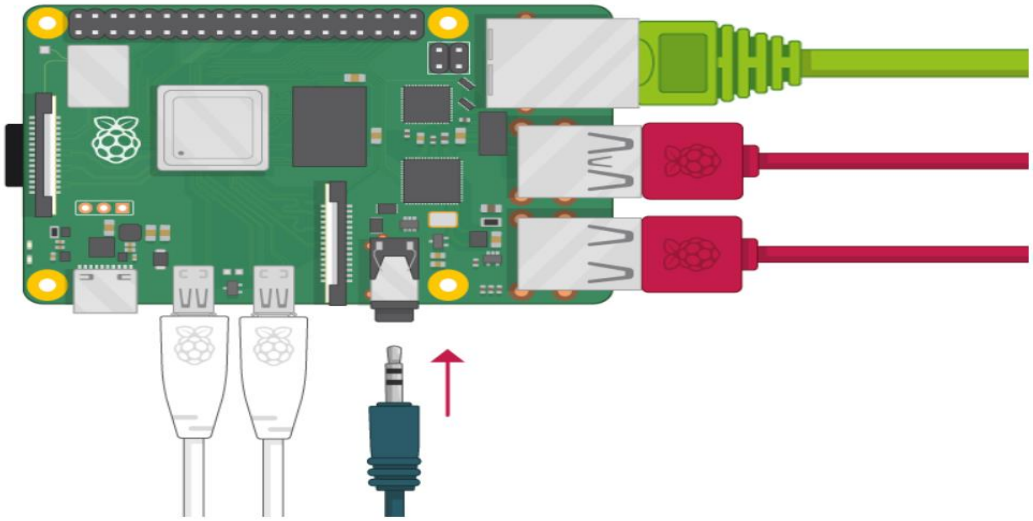


Note: Nothing will display on the screen, because your Raspberry Pi is not running yet.

- If you want to connect your Raspberry Pi to the internet via Ethernet, use an Ethernet cable to connect the Ethernet port on Raspberry Pi to an Ethernet socket on the wall or on your internet router. You don't need to do this if you want to use wireless connectivity, or if you don't want to connect to the internet.



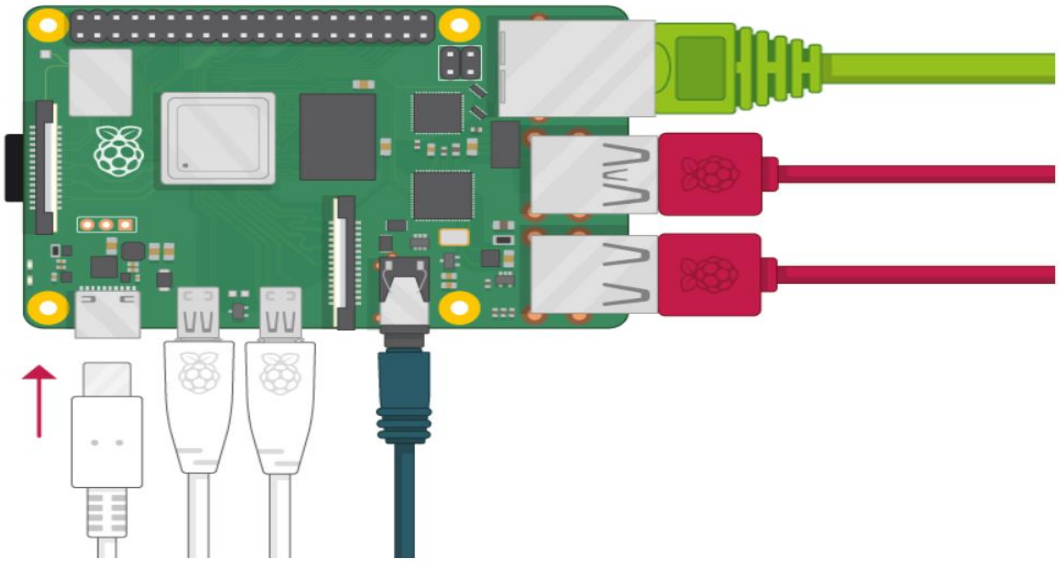
- If the screen you are using has speakers, sound will play through those. Alternatively, connect headphones or speakers to the audio port if you prefer.



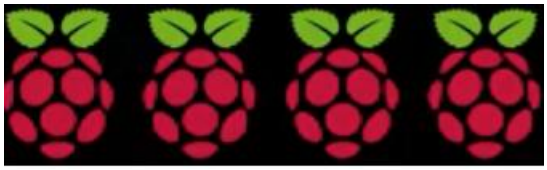
Start up your Raspberry Pi

Your Raspberry Pi doesn't have a power switch. As soon as you connect it to a power outlet, it will turn on.

- Plug the power supply into a socket and connect it to your Raspberry Pi's power port.



You should see a red LED light up on the Raspberry Pi, which indicates that Raspberry Pi is connected to power. As it starts up (this is also called booting), you will see raspberries appear in the top left-hand corner of your screen.



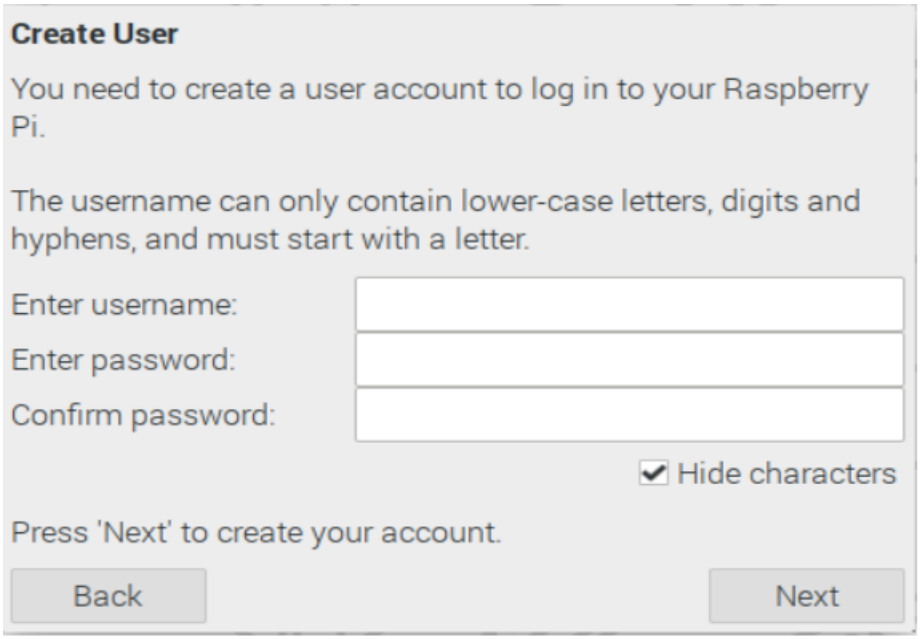
After a few seconds the Raspberry Pi OS desktop will appear.

Finishing the setup

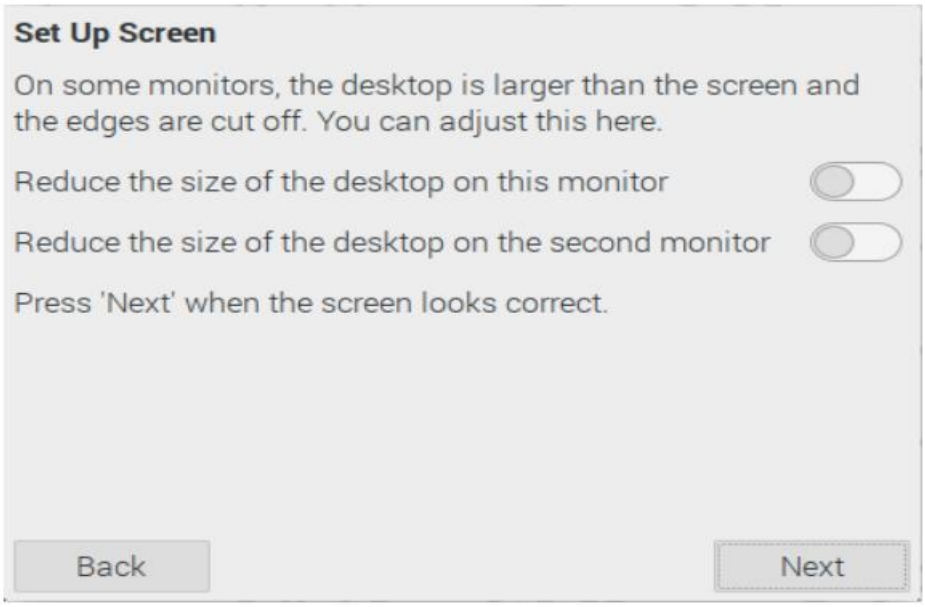
When you start your Raspberry Pi for the first time, the Welcome to Raspberry Pi application will pop up and guide you through the initial setup.

- Click on Next to start the setup.
- Set your **Country**, **Language**, and **Time zone**, then click on **Next** again.

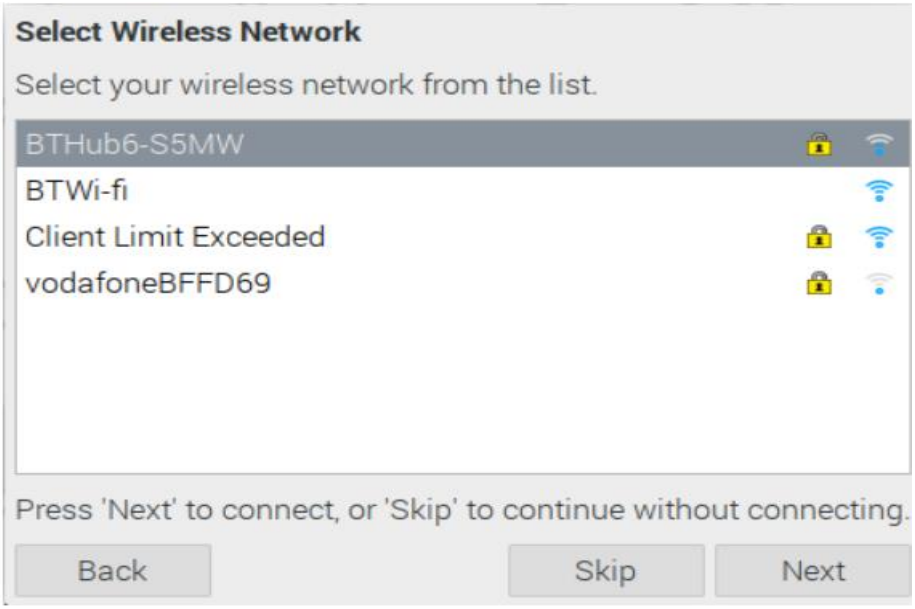
- Enter a new username and password for your Raspberry Pi and click on **Next**



- Set up your screen so that the Desktop completely fills your monitor

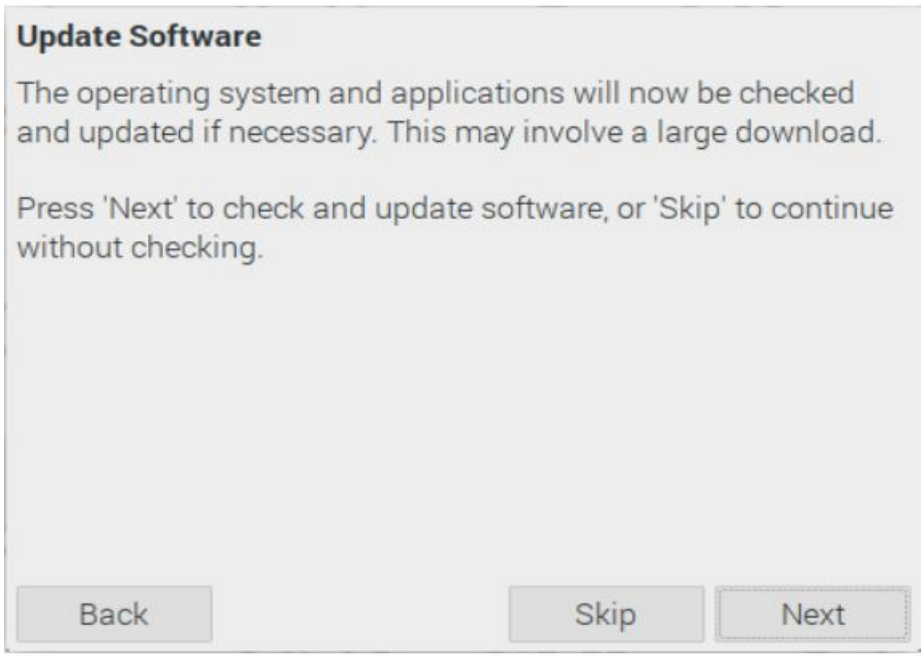


- Connect to your wireless network by selecting its name, entering the password, and clicking on **Next**.



Note: If your model of Raspberry Pi doesn't have wireless connectivity, you won't see this screen.
Note: Wait until the wireless connection icon appears and the correct time is shown before trying to update the software.

- Click on Next, and let the wizard check for updates to Raspberry Pi OS and install them (this might take a little while).



- Click on Restart to finish the setup.

Finally pi wizard complete.

11.3 Task 1

The task: The Blue Pill Interface with Raspberry Pi to Start the Measurements using UART.

11.3.1 Description

For this task, to establish a UART connection between a Raspberry Pi and a Blue Pill (STM32 microcontroller), you need to connect the appropriate UART pins on both devices, configure the hardware and software for UART communication, and understand the potential applications of this setup.

11.3.2 Hardware Components

- Raspberry Pi (e.g., Raspberry Pi 3 Model B or 3 Model B+)
- Blue Pill (STM32F103C8T6)
- Jumper wires (female-to-female or male-to-female depending on connectors)
- A common ground connection
- USB to TTL converter (optional, for initial debugging)
- Serial communication software (e.g., minicom, screen, picocom, etc.)

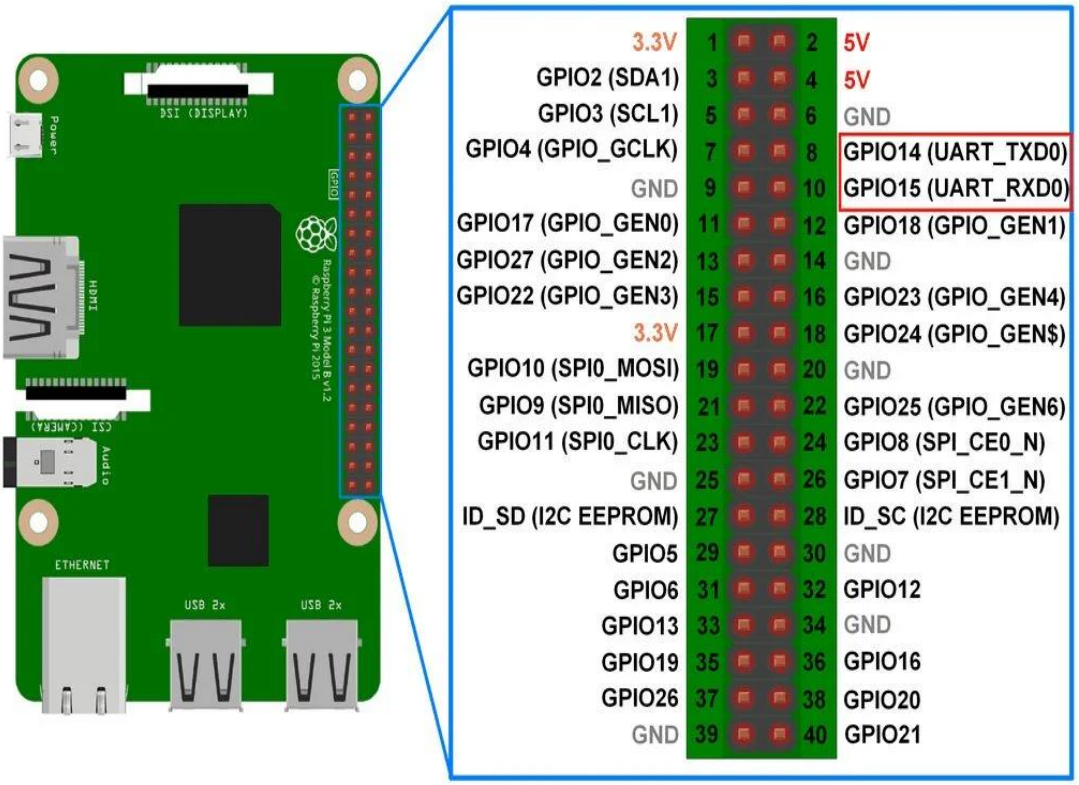


Figure 11.2 Raspberry Pi 3B+ UART Pins

11.3.3 Steps:

1. Identify UART Pins

- **Raspberry Pi GPIO Pins:**
- TX (Transmit): GPIO 14 (Pin 8)
- RX (Receive): GPIO 15 (Pin 10)
- Ground: Any ground pin (e.g., Pin 6)

- **Blue Pill Pins:**
- TX (Transmit): PA9 (USART1_TX)
- RX (Receive): PA10 (USART1_RX)
- Ground: Any ground pin

2. Wiring

- **Connect the pins as follows:**
- Raspberry Pi TX (GPIO 14) to Blue Pill RX (PA10)
- Raspberry Pi RX (GPIO 15) to Blue Pill TX (PA9)
- Ground (GND) on the Raspberry Pi to Ground (GND) on the Blue Pill
- Make sure the connections are secure.

3. Configure Raspberry Pi UART

By default, the Raspberry Pi UART might be used for console messages. Disable this to use UART for your own applications.

a. Disable Console over UART:
   - Edit the file /boot/cmdline.txt and remove any references to ttyAMA0 or serial0.
   - Edit /boot/config.txt and add the following lines.

b. Reboot the Raspberry Pi.

4. Configure Blue Pill UART

Ensure that the Blue Pill's firmware is configured to use UART. This typically involves:

- Initializing UART peripheral in your code.
- Setting baud rate and other UART parameters.

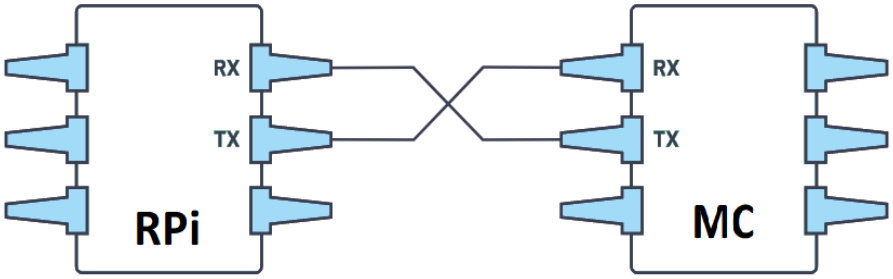


Figure 11.3 UART Connection

11.3.4 Code:

1- python

First, make sure pyserial is installed on the Raspberry Pi. You can install it using pip.

Raspberry Pi Python Code:

This script will set up the Raspberry Pi to send and receive messages over UART.

```
import serial
import time

# Configure the serial port
serial_port = '/dev/serial0'  # Serial port for Raspberry Pi UART
baud_rate = 9600              # Baud rate (should match the Blue Pill's setting)

# Initialize the serial connection
ser = serial.Serial(serial_port, baud_rate, timeout=1)

def send_message(message):
    """Send a message to the Blue Pill."""
    ser.write(message.encode())
    print(f"Sent: {message}")

def receive_message():
    """Receive a message from the Blue Pill."""
    if ser.in_waiting > 0:
        message = ser.readline().decode().strip()
        return message
    return None
```

```
try:
    while True:
        # Send a test message every 2 seconds
        send_message("Hello from Raspberry Pi")
        time.sleep(2)

        # Check for any incoming messages
        received = receive_message()
        if received:
            print(f"Received: {received}")

except KeyboardInterrupt:
    print("Program interrupted. Closing serial connection.")
finally:
    # Close the serial connection
    ser.close()
```

Blue Pill Code (C using STM32 HAL Library):

To complement the Python script on the Raspberry Pi, here's the corresponding Blue Pill code to send and receive messages using the STM32 HAL library.

11.3.5 Steps to Run code:

1.Setup Wiring:

- Connect Raspberry Pi GPIO 14 (TX) to Blue Pill PA10 (RX).
- Connect Raspberry Pi GPIO 15 (RX) to Blue Pill PA9 (TX).
- Connect Raspberry Pi GND to Blue Pill GND.

2.Configure Raspberry Pi UART:

- Edit /boot/cmdline.txt and /boot/config.txt as mentioned in previous steps.
- Reboot the Raspberry Pi.

3.Upload Blue Pill Code:

- Use an STM32 development environment (e.g., STM32CubeIDE) to

upload the provided code to the Blue Pill.

4.Run Raspberry Pi Python Script:

- Save the Python script to a file (e.g., uart_communication.py).
- Run the script using:

This setup will enable UART communication between the Raspberry Pi and the Blue Pill, allowing them to send and receive messages to each other.

11.3.6 Flow chart for connect raspberry pi and blue pill using UART

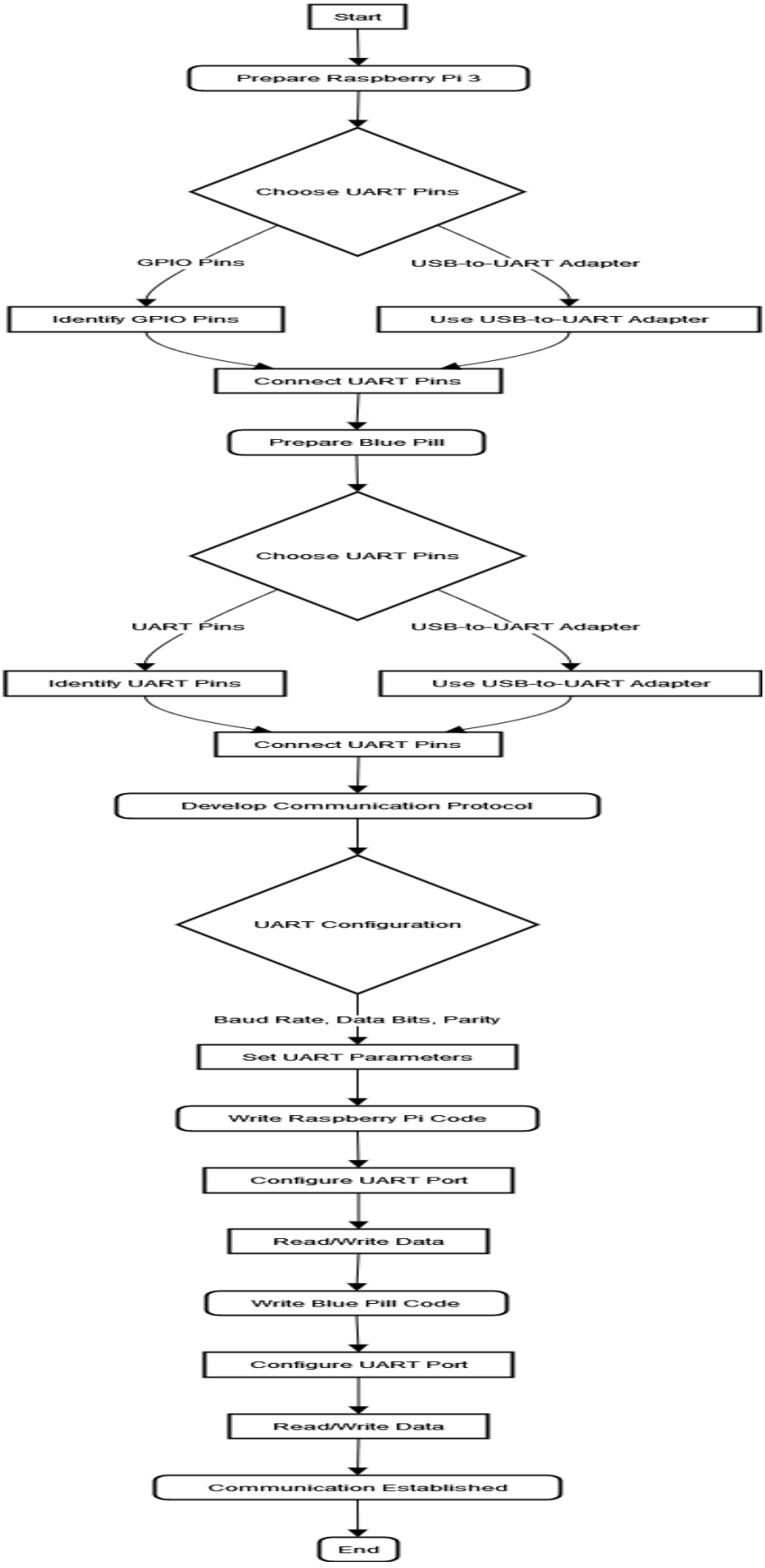


Figure 11.4 Flow Chart

11.4 Task 2

The task: Connect between Two Raspberry Pi by using MQTT.

11.4.1 Description

For this task, the process involves setting up an MQTT broker on one Raspberry Pi, then configuring both Raspberry Pi devices to use MQTT clients to publish and subscribe to messages. The steps below describe what happens at each stage of this setup.

To connect two Raspberry Pi devices using MQTT, you'll need to set up an MQTT broker on one of the Raspberry Pis and use MQTT clients on both devices to publish and subscribe to messages.

11.4.2 Requirements

- Two Raspberry Pi devices (e.g., Raspberry Pi 3 Model B and Raspberry Pi 3 Model B+)
- MQTT broker software (e.g., Mosquitto)
- MQTT client software (e.g., Paho MQTT)

11.4.3 Steps:

1. Install Mosquitto MQTT Broker on One Raspberry Pi

- Choose one Raspberry Pi to act as the MQTT broker.

  1- Update the package list and install Mosquitto

  2- Enable and start the Mosquitto service.

2. Install MQTT Client on Both Raspberry Pis.

On both Raspberry Pi devices, install the Paho MQTT client:

1-Update the package list and install the Paho client.

3. Set Up Python Scripts for Publishing and Subscribing

Create Python scripts to publish and subscribe to MQTT topics.

Publisher Script (Raspberry Pi 1):

```python
import paho.mqtt.client as mqtt
import time

# Define the MQTT broker address and port
broker_address = "BROKER_IP_ADDRESS"  # Replace with the IP address of the broker
broker_port = 1883
topic = "test/topic"

# Create a new MQTT client instance
client = mqtt.Client()

# Connect to the MQTT broker
client.connect(broker_address, broker_port, 60)

while True:
    message = "Hello from Raspberry Pi 1"
    client.publish(topic, message)
    print(f"Published: {message}")
    time.sleep(2)
```

Subscriber Script (Raspberry Pi 2):

```python
import paho.mqtt.client as mqtt

# Define the MQTT broker address and port
broker_address = "BROKER_IP_ADDRESS"  # Replace with the IP address of the broker
broker_port = 1883
topic = "test/topic"

# Callback function to handle incoming messages
def on_message(client, userdata, message):
    print(f"Received message: {message.payload.decode()} on topic {message.topic}")

# Create a new MQTT client instance
client = mqtt.Client()

# Attach the on_message callback function to the client
client.on_message = on_message
```

```
# Callback function to handle incoming messages
def on_message(client, userdata, message):
    print(f"Received message: {message.payload.decode()} on topic {message.topic}")

# Create a new MQTT client instance
client = mqtt.Client()

# Attach the on_message callback function to the client
client.on_message = on_message

# Connect to the MQTT broker
client.connect(broker_address, broker_port, 60)

# Subscribe to the topic
client.subscribe(topic)

# Start the client loop to listen for messages
client.loop_forever()
```

Replace BROKER_IP_ADDRESS with the IP address of the Raspberry Pi running the Mosquito broker.

4. Run the Scripts

1- On Raspberry Pi 1 (Publisher).

2- On Raspberry Pi 2 (Subscriber).

11.4.4 Testing the steps:

1. **Publisher (Raspberry Pi 1):**

- This script will continuously publish the message "Hello from Raspberry Pi 1" to the test/topic every 2 seconds.

2. **Subscriber (Raspberry Pi 2):**

- This script will subscribe to test/topic and print any received messages.

If everything is set up correctly, the subscriber should print "Hello from Raspberry Pi 1" every 2 seconds.

11.4.5 Troubleshooting

1. Network Connectivity: Ensure both Raspberry Pis are on the same network and can ping each other.
2. Firewall: Make sure no firewall is blocking port 1883.
3. Broker Status: Verify that the Mosquito broker is running using sudo systemctl status mosquito.
4. Logs: Check logs for any errors:

   a- Mosquito logs: /var/log/mosquito/mosquitto.log

   b- Python script errors: Ensure no syntax or runtime errors.

11.4.6 Diagram or Flow chart of MQTT Setup

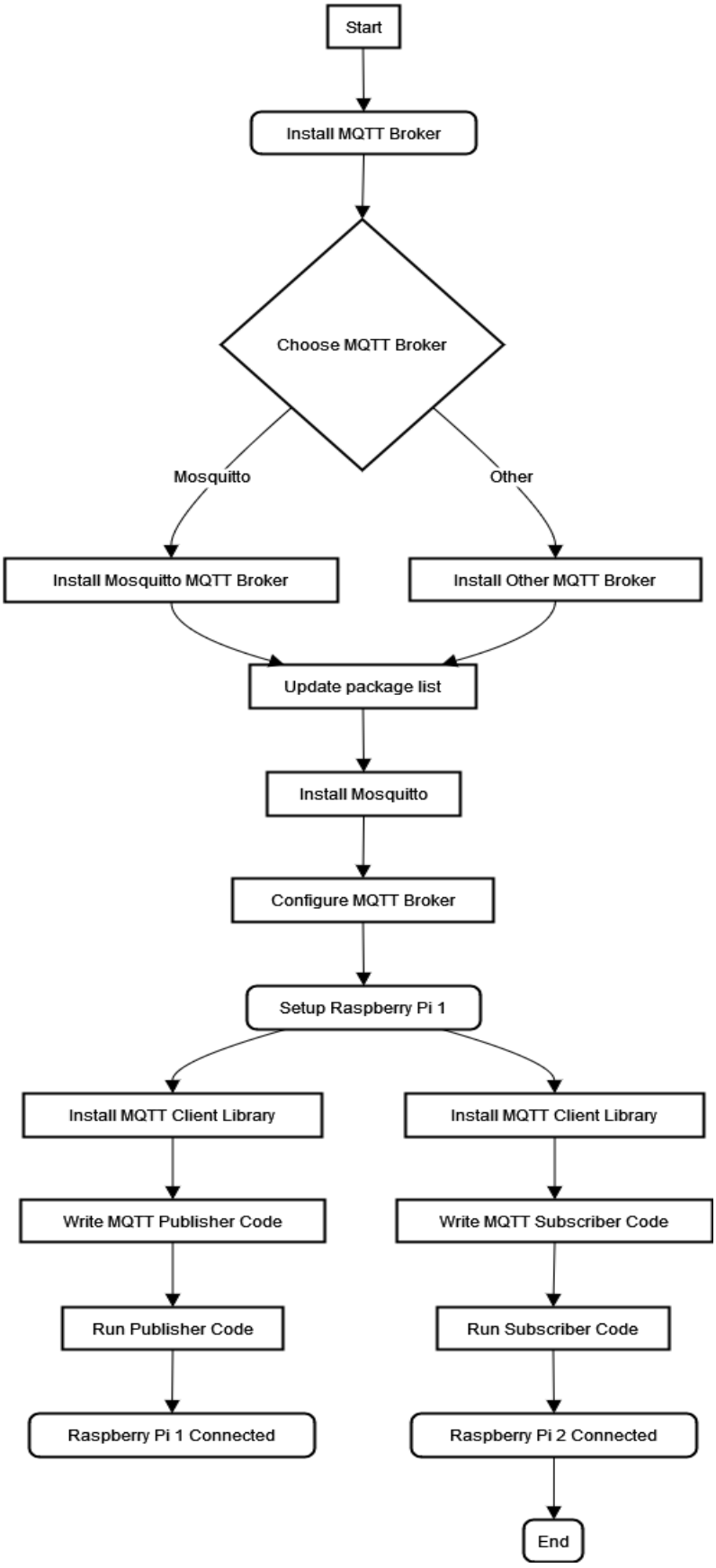


Figure 11.5 Flow Chart

By following these steps and using this diagram, you can set up a communication channel between two Raspberry Pi devices using MQTT [61].

11.5 Task 3

11.5.1 Description

For this task, to create a graphical user interface (GUI) on a Raspberry Pi 3 B using both Qt and Tkinter, including a brief overview of each toolkit.

Overview of Qt and Tkinter

Qt:

- Qt is a powerful, cross-platform C++ framework for creating GUI applications. It includes a wide range of tools and modules, such as Qt Widgets for creating traditional desktop interfaces and Qt Quick for building fluid and dynamic UIs using QML.
- Qt Creator is the official integrated development environment (IDE) for developing applications with the Qt application framework.

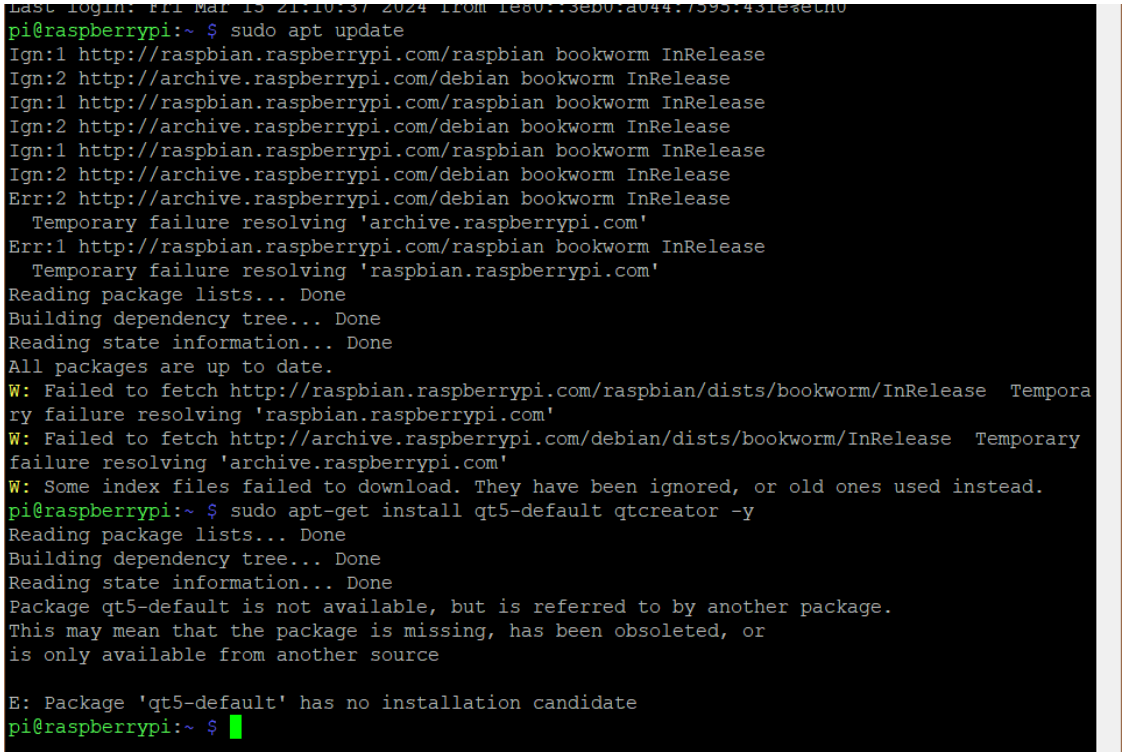
Tkinter:

- Tkinter is the standard Python interface to the Tk GUI toolkit. It is one of the simplest ways to create GUIs in Python and comes pre-installed with Python on most systems.
- Tkinter is a lightweight toolkit suitable for smaller applications and rapid prototyping.

11.5.2 Steps to Create a GUI with Qt on Raspberry Pi

1. Install Qt and Qt Creator

First, update your package list and install Qt and Qt Creator:



2. Create a New Qt Project

1. Open Qt Creator:

- Launch Qt Creator by typing qtcreator in the terminal.

2.Create a New Project:

- Go to File -> New File or Project.
- Choose Application -> Qt Widgets Application.
- Follow the prompts to set up your project. Provide a name and a location for your project.

3.Design the Interface:

- In the Project Explorer, open the mainwindow.ui file.
- Use the widget box to drag and drop elements (buttons, labels, etc.) into the design area.

4.Implement the Logic:

- Open mainwindow.cpp and add the logic for your application. For example, to display a message box when a button is clicked:

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    connect(ui->pushButton, &QPushButton::clicked, this, &MainWindow::on_pushButton_cl
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{
    QMessageBox::information(this, "Title", "Button clicked!");
}
```

5.Build and Run the Project:

- Click the green play button to build and run your application. This will compile your code and launch the application, showing the GUI you designed.

11.5.3 Steps to Create a GUI with Tkinter on Raspberry Pi

1. Install Tkinter

Tkinter typically comes pre-installed with Python. If it's not installed, you can install it using:

```
pi@raspberrypi:~ $ sudo apt-get install python3-tk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-tk is already the newest version (3.11.2-3).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $
```

2. Create a Tkinter Application

1. Write the Tkinter Code:

- Open a text editor and create a new Python file, such as app.py.
- Add the following code to create a simple GUI with a button:

```
# app.py
import tkinter as tk
from tkinter import messagebox

def on_button_click():
    messagebox.showinfo("Title", "Button clicked!")

root = tk.Tk()
root.title("Tkinter App")

button = tk.Button(root, text="Click Me", command=on_button_click)
button.pack(pady=20)

root.mainloop()
```

2. Run the Application:

- Open the terminal and navigate to the directory where you saved app.py.
- Run the application using:

11.5.4 Running the Applications

- Qt Application: After running your Qt application, you will see a window with the interface you designed. Clicking the button will show a message box with the text "Button clicked!".
- Tkinter Application: After running your Tkinter application, you will see a window with a button labeled "Click Me". Clicking the button will display a message box with the text "Button clicked!".

11.5.5 Flow chart for GUI

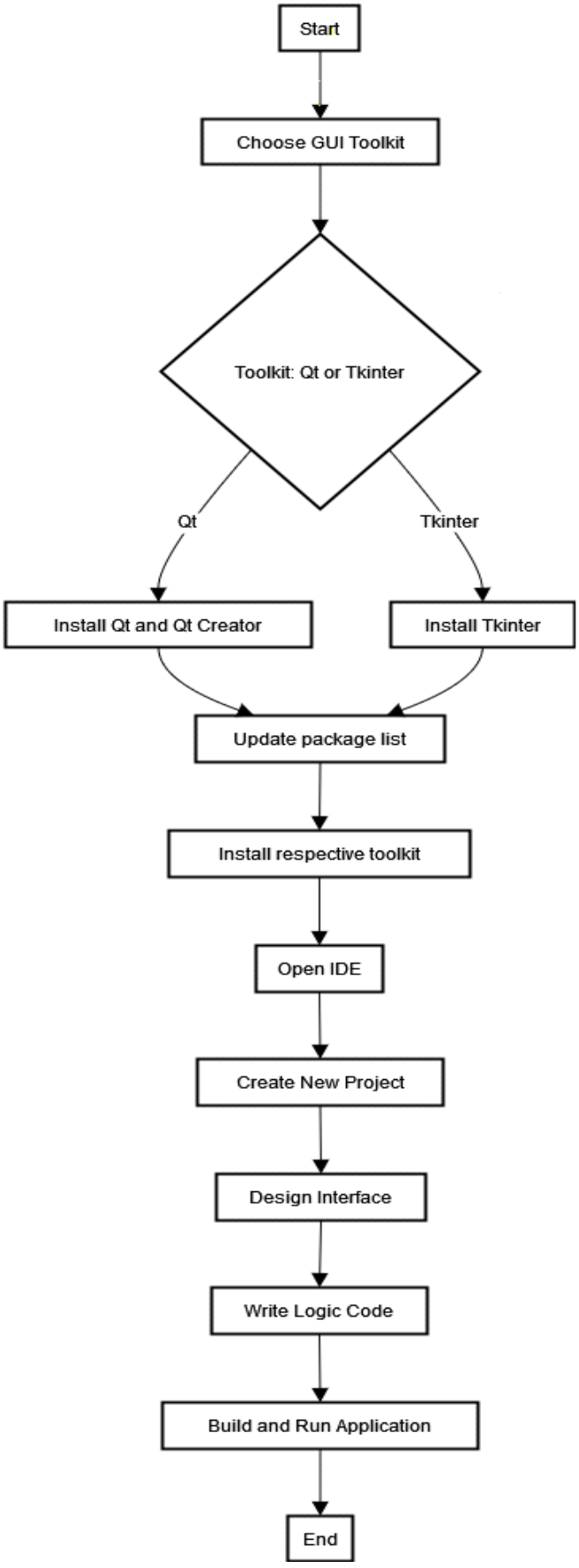


Figure 11.6 Flow Chart

11.5.6 Summary

Qt:

- More powerful and flexible, suitable for complex applications.
- Requires installation of Qt and Qt Creator.
- Uses C++ for development.
- Provides extensive tools and modules for various functionalities.

Tkinter:

- Simpler and easier to use, suitable for smaller applications and rapid prototyping.
- Comes pre-installed with Python.
- Uses Python for development.
- Lightweight and quick to set up.

By following these steps, you can create rich graphical user interfaces on a Raspberry Pi 3 B using either Qt or Tkinter, depending on my project needs and programming preferences.

Future Work

- The future of your ADAS (Advanced Driver Assistance System) AEB (Automatic Emergency Braking) system using ultrasonic technology holds promise in several areas:

  1. **Advancements in Sensing Technology:** Ultrasonic sensors are likely to become more sophisticated, offering higher accuracy and reliability in detecting objects and distances. This could improve the overall performance and safety of your ADAS.

  2. **Integration with AI and Machine Learning:** Future developments may involve integrating ultrasonic sensors with AI algorithms for better object recognition, predictive analysis, and decision-making capabilities. This can enhance the system's ability to anticipate and respond to potential hazards on the road.

  3. **Cost Reduction and Accessibility:** As technology matures and economies of scale kick in, the cost of ultrasonic sensors may decrease, making ADAS systems more affordable and accessible to a broader range of vehicles and consumers.

  4. **Regulatory and Safety Standards**: There is an increasing focus on safety and regulatory standards in automotive technology. Your ADAS AEP system will need to stay updated with these standards to ensure compliance and market acceptance.

  5. **User Experience Enhancements:** Future developments may also focus on improving the user interface and experience of ADAS systems, making them more intuitive and user-friendly.

- The future of your ADAS (Advanced Driver Assistance System) ACC (Adaptive Cruise Control) project using ultrasonic sensors sounds promising. Here are some potential directions it could take:

  1. **Enhanced Features:** You could expand the capabilities to include features like lane-keeping assist, collision avoidance, or automated parking, depending on the sensor capabilities and integration.

  2.**Sensor Upgrades**: Consider upgrading sensors to more advanced technologies like LIDAR or radar for better performance in various weather conditions and at different speeds.

  3. **User Interface Improvements**: Enhancing the user interface for better interaction and feedback could improve user experience and safety.

  4. **Regulatory Compliance**: Keeping up with and adapting to evolving automotive safety regulations and standards will be crucial for the project's future viability.

  5. **Testing and Validation:** Continuously testing and validating the system under various scenarios will be essential to ensure reliability and safety.

  6. **Market Adoption**: If feasible, exploring opportunities for commercialization or partnerships with automotive manufacturers or technology integrators could be a path forward.

  Overall, the future success of your project will likely depend on further development in sensor technology, integration with AI for smarter decision-making, and meeting regulatory standards for automotive safety.

- The future work for a self-parking Advanced Driver Assistance System (ADAS) could involve several advancements and enhancements:

    1. **Improved Sensor Technology:** Upgrading sensors like cameras, and LiDAR to enhance accuracy and reliability in detecting obstacles and surroundings.

    2. **Enhanced AI Algorithms**: Developing more sophisticated algorithms for path planning, obstacle avoidance, and decision-making in complex parking scenarios.

    3.integration with V2X Communication: Utilizing Vehicle-to-Everything (V2X) communication for better coordination with other vehicles and infrastructure, such as receiving real-time information about available parking spaces.

    4.**Machine Learning for Personalization:** Implementing machine learning models to adapt the parking behavior based on individual driving styles, preferences, and parking environment characteristics.

    5.**Cybersecurity Measures:** Strengthening cybersecurity protocols to protect the system from potential cyber threats and ensure the safety and privacy of user data.

    6.**Testing and Validation:** Conducting extensive testing and validation in various real-world scenarios to ensure the reliability and safety of the self-parking ADAS system.

These advancements aim to make self-parking ADAS systems more efficient, reliable, and user-friendly, ultimately contributing to the broader goal of enhancing vehicle automation and improving overall driving experience.

REFRENCES

[1] Global, regional, and national age-sex-specific mortality for 282 causes of death in 195 countries and territories, 1980–2017: a systematic analysis for the Global Burden of Disease Study 2017.

[2] Global health estimates 2015: deaths by cause, age, sex, by country and by region, 2000–2015. World Health Organization, Geneva2017.

[3] Ronald Jurgen," V2V/V2I Communications for Improved Road Safety and Efficiency", SAE International, Warrendale, 2012.

[4] Li, Zekai, et al. "A genuine V2V market mechanism aiming for maximum revenue of each EV owner based on non-cooperative game model." *Journal of Cleaner Production* 414 (2023): 137586.

[5] https://opentransportationjournal.com/VOLUME/14/PAGE/86

[6] https://www.st.com/resource/en/datasheet/cd00161566.pdf

[7] https://www.wellpcb.com/STM32-pinout.html

[8]https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[9]https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d1fbdf8a3a1b32fd9c951f44f0f5403232412795

[10] Bräunl, Thomas. *Embedded robotics*. Springer, 2003.

[11] https://www.freertos.org/index.html

[12] Brown A. E, E. G. Richardson, "Ultrasonic Physics", Elsevier, 2013

[13] "HC-SR04 User's Manual." Docs. Google. Cytron Technologies, May 2013 Web. 5 Dec 2009.

[14] Tracey Scott Wilson,"Buzzer", Dramatist's Play Service,2016

[15] Hemati, Neyram, and Ming C. Leu. "A complete model characterization of brushless DC motors." *IEEE Transactions on Industry Applications* 28.1 (1992): 172-180.

[16] Keream, Settar S., Khalid G. Mohammed, and Mayyada Sahib Ibrahim. “Analysis study in principles of operation of DC machine.” Journal of Advanced Research in Dynamical and Control Systems 10.02 (2018): 2323-2329.

[17] Muruganandam, M., and M. Madheswaran. “Modeling and simulation of modified fuzzy logic controller for various types of DC motor drives.” 2009 International Conference on Control, Automation, Communication and Energy Conservation. IEEE, 2009.

[18] Niapour, S. A., et al. “Review of permanent-magnet brushless DC motor basic drives based on analysis and simulation study.” International Review of Electrical Engineering 9.5 (2014): 930-957.

[19] Sivakumar, S., et al. “An assessment on performance of DC–DC converters for renewable energy applications.” Renewable and Sustainable Energy Reviews 58 (2016): 1475-1485.

[20] https://www.electrical4u.com/what-is-servo-motor/

[21] Baballe, Muhammad Ahmad, et al. “Different Types of Servo Motors and Their Applications.” *1st International Conference on Engineering and Applied Natural Sciences*. 2022.

[22] Sedra, A. S., & Smith, K. C. (2015). *Microelectronic Circuits*. Oxford University Press.

[23] Mohan, N., Undeland, T. M., & Robbins, W. P. (2003). Power Electronics: Converters, Applications, and Design. Wiley.

[24] Mechatronics and H-bridge circuits. (n.d.). Retrieved from https://example.com/mechatronics_hbridge

[25] Anderson, D. P. (2000). *Build Your Own Robot*. McGraw-Hill.

[26]https://www.techtarget.com/whatis/definition/USART-Universal-Synchronous-Asynchronous-Receiver-Transmitter

[27] RM0008 Reference manual

[28]https://fastbitlab.com/exploring-uart-functional-block/

[29]https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

[30]https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-UART.html#google_vignette

[31] Sairam, K. V. S. S. S. S., N. Gunasekaran, and S. Rama Redd. “Bluetooth in wireless communication.” *IEEE Communications Magazine* 40.6 (2002): 90-96.

[32] https://www.geeksforgeeks.org/bluetooth/

[33]https://www.rajguruelectronics.com/ProductView?product=HC05%20CORE%20MODULE&tokDatRef=NzA1&tokenId=NDA=

[34] https://electronicsmaker.com/bluetooth-controlled-robot-using-android-mobile

[35] https://electrosome.com/hc-05-serial-bluetooth-module/

[36] Choi, S.B. and Hedrick, J.K., "Vehicle Longitudinal Control Using an Adaptive Observer for Automated Highway Systems", Proceedings of American Control Conference, Seattle, Washington, 1995.

[37] HEARST AUTOS RESEARCH, "What Is Adaptive Cruise Control?" https://www.caranddriver.com/research/a32813983/adaptive-cruise-control/, Jun. 2020, accessed: 2022-Oct-04.

[38] Adaptive Cruise Control System Using Model Predictive Control – MATLAB & Simulink (mathworks.com)

[39] Lee, J., Kim, G. and Kim, B. (2019) Study on the improvement of a collision avoidance system for curves, MDPI. Available at: https://www.mdpi.com/2076-3417/9/24/5380/htm (Accessed: 10 July 2023).

[40] (2008) Automated Emergency Brake System: Technical requirements, costs and benefits. (1st ed., pp. 1-117). Published Project Report

[41] Drowsiness and distraction detection: Pooja D.C., Sara Aziz, Shakuntala Koujalagi, Shilpa B. H., Mr. Vasanth Kumar N.T. (2022). Driver Drowsiness Detection Using OpenCV and Raspberry Pi. International Journal for Research in Applied Science & Engineering Technology (IJRASET), Volume 10 (Issue VII). 1-7. https://doi.org/10.22214/ijraset.2022.45288.

[42] Huang, S. J., and G. Y. Lin. "Parallel auto-parking of a model vehicle using a self-organizing fuzzy controller." Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering 224.8 (2010): 997-1012.

[43] Rashid, Md Mamunur, et al. "Autonomous 4wd smart car parallel self-parking system by using fuzzy logic controller." American International Journal of Sciences and Engineering Research 2.2 (2019): 1-31.

[44] Hallinan, Christopher. Embedded Linux primer: a practical real-world approach. Pearson Education India, 2011.

[45] Raghavan, Pichai, Amol Lad, and Sriram Neelakandan. Embedded Linux system design and development. Auerbach Publications, 2005.

[46] Abbott, Doug. Linux for embedded and real-time applications. Elsevier, 2011.

[47] Simmonds, Chris. Mastering Embedded Linux Programming. Packt Publishing Ltd, 2017.

[48] Bi, Chun-yue, Yun-peng Liu, and Ren-fang Wang. “Research of key technologies for embedded Linux based on ARM.” 2010 International Conference on Computer Application and System Modeling (ICCASM 2010). Vol. 8. IEEE, 2010.

[49] Vaduva, Alexandru, Alex Gonzalez, and Chris Simmonds. Linux: Embedded Development. Packt Publishing Ltd, 2016.

[50] Hallinan, Christopher. “Reducing boot time in embedded linux systems.” LINUX journal 2009.188 (2009): 4.

[51] Wang, Ya-Jun. “Research and realization of the mechanism of embedded linux kernel semaphore.” 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE). Vol. 1. IEEE, 2010.

[52] Molloy, Derek. Exploring Raspberry Pi: interfacing to the real world with embedded Linux. John Wiley & Sons, 2016.

[53] Simmonds, Chris. Mastering embedded Linux programming. Packt Publishing Ltd, 2015.

[54] Salvador, Otavio, and Daiane Angolini. Embedded Linux Development with Yocto Project. Packt Publishing Ltd, 2014.

[55] Tripathy, B. K., and J. Anuradha, eds. Internet of things (IoT): technologies, applications, challenges and solutions. CRC press, 2017.

[56] Serpanos, Dimitrios, and Marilyn Wolf. Internet-of-things (IoT) systems: architectures, algorithms, methodologies. Springer, 2017.

[57] Vermesan, Ovidiu, and Peter Friess, eds. Building the hyperconnected society: Internet of things research and innovation value chains, ecosystems and markets. Vol. 43. River Publishers, 2015.

[58] Samie, Farzad, Lars Bauer, and Jörg Henkel. “IoT technologies for embedded computing: A survey.” 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS). IEEE, 2016.

[59] Kuo, Yaw-Wen, et al. “Design of a wireless sensor network-based IoT platform for wide area and heterogeneous applications.” IEEE Sensors Journal 18.12 (2018): 5187-5197.

[60] Weber, Rolf H., and Romana Weber. Internet of things. Vol. 12. Heidelberg: Springer, 2010.

[61] Lucero, Sam. “IoT platforms: enabling the Internet of Things.” White paper (2016).

[62] Light, Roger A. “Mosquitto: server and client implementation of the MQTT protocol.” Journal of Open-Source Software 2.13 (2017).

[63] Soni, Dipa, and Ashwin Makwana. “A survey on mqtt: a protocol of internet of things (iot).” International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017). Vol. 20. 2017.

[64] Lutz, Mark. Learning python: Powerful object-oriented programming. "O'Reilly Media, Inc.", 2013.

الملخص العربى

تحدث حوادث السيارات أو ما يُعرَف أيضاً بالحوادث المروريّة نتيجه حدوث تصادم بين مركبه ومركبة أخرى، أو حدوث تصادُمٍ بين مركبةٍ ما وعناصر الطريق المُحيطة بها، كالحيوانات،المارّة ، والأجسام الثابتة، ولحوادث المرور أضرار بالغة.

السبب الرئيسي وراء الحوادث هو الخطأ البشري. وتم توضيح بعض السلوكيات البشرية الشائعة التي تؤدي إلى وقوع الحواث.

وجد أن ٩٩% من حوادث الطريق يرجع سببها إلى الخطأ البشري، وهو حقيقة مزعجة أن معظم السائقين يواصلون ارتكاب نفس الأخطاء على الطريق دون أن ينتبهوا إلى عادات القيادة السيئة لديهم .

تحدث العديد من الحوادث بسبب الإنشغال عن الطريق، فنقوم باستخدام الانظمة المساعدة المتقدمه لتجنب الاصطدام والحفاظ علي المركبه والسيطره عليها واعطاء بعض الاشارات التحذريه، وفي هذا المشروع يتم استخدام ثلاث تطبيقات رئيسية
وهم:

1-الكبح التلقائي في حالات الطوارئ
هي ميزه أمان مصممه لاكتشاف الاصطدام الوشيك مع مركبه أخري أو عائق أخر فيقوم بتنبيه السائق أو تطبيق نظام الفرامل التلقائي عند عدم تنبه السائق.
2- نظام تثبيت السرعه التكيفي
هي تقنيه تعمل علي ضبط السرعه تلقائيا للحفاظ علي مسافه آمنه مع المركبه في الأمام عن طريق استخدام أجهزه استشعارلمراقبه البيئه المروريه.
3-نظام ركن السيارات الذاتي
هي عباره عن آليه تمكن السياره من الوصول الي مكان ركنها دون ادخال السائق عن طريق تحديد مكان مناسب بواسطه أجهزه استشعار.

جامعة الأزهر
كلية الهندسة
قسم الإلكترونيات و الإتصالات

مشروع تخرج بكالوريوس
(الإلكترونيات و الإتصالات)

أنظمة متقدمة لمساعدة السائق لتفادي الاصطدامات

تحت إشراف:
د. محمد يسن إبراهيم عفيفي

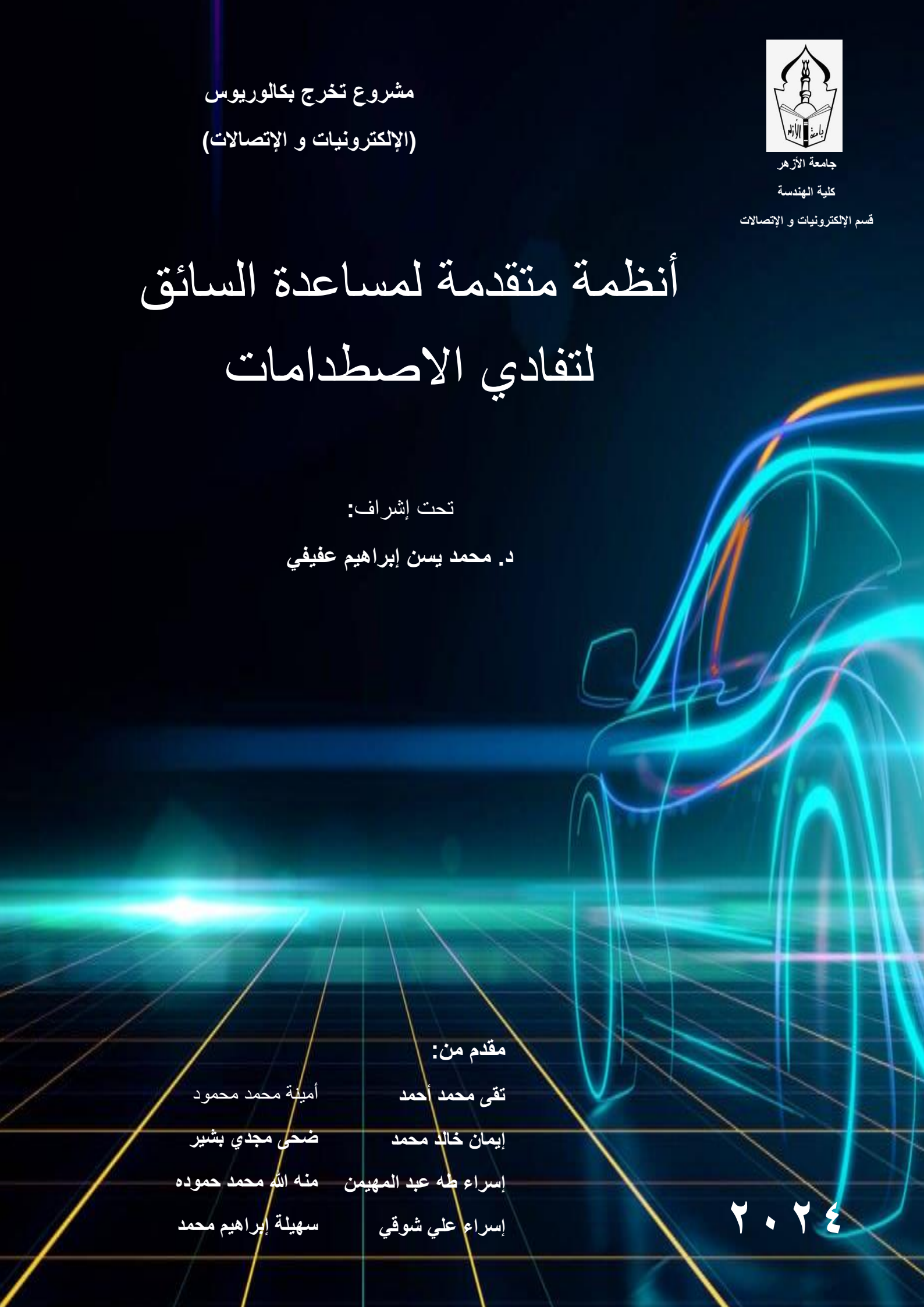
مقدم من:
تقى محمد أحمد
أمينة محمد محمود
إيمان خالد محمد
ضحى مجدي بشير
إسراء طه عبد المهيمن
منه الله محمد حموده
إسراء علي شوقي
سهيلة إبراهيم محمد

٢٠٢٤

جامعة الأزهر
كلية الهندسة
قسم الإلكترونيات و الإتصالات

مشروع تخرج بكالوريوس
(الإلكترونيات و الإتصالات)

أنظمة متقدمة لمساعدة السائق لتفادي الاصطدامات



تحت إشراف:

د. محمد يسن إبراهيم عفيفي

مقدم من:

تقى محمد أحمد أمينة محمد محمود

إيمان خالد محمد **ضحى مجدي بشير**

إسراء طه عبد المهيمن **منه الله محمد حموده**

إسراء علي شوقي **سهيلة إبراهيم محمد**

٢٠٢٤