ITI – 4 Months

Embedded Systems Track

Graduation Project

Adaptive Cruise Control System Connected to Infotainment System

By:

Alaa Ashraf Fawzy

Esraa Ali El-Qassas

Abdel Rahman Kamal

Mohamed Ashraf Abdel-Haleem

Under Supervision Of:

Youssef Nofal

Nour Hassan

Acknowledgment

We owe our gratitude to Eng. Youssef Nofal Head of Embedded System Track in ITI for his support and encouragement and for always pushing us to do better and to reach our own very limits.

We sincerely thank our instructors for their help and support throughout the last four month.

And we also thank the Information Technology Institute for allowing us to peruse our dreams of becoming a highly skilled engineers and giving us this chance to learn more about our field and show our work to the world.

Abstract

The proposed project aims to design an Adaptive Cruise Control System that's connected to Infotainment System based on ADAS system feature. The system uses sensors to detect the distance of other vehicles on the road, and then adjusts the speed of the vehicle accordingly. The system can also be used to maintain a safe following distance between vehicles, allowing for smoother driving and reducing the risk of collisions. Additionally, this system can is also used in conjunction with another driver assistance technology which is automatic emergency braking. This abstract outlines the benefits of adaptive cruise control systems and how they can help improve driver safety and comfort.

Table Of Contents

Table Of Figures

Chapter 1 - Introduction

1.1 General Overview

Adaptive cruise control (ACC) is a system designed to help vehicles maintain a safe following distance and stay within the speed limit. This system adjusts a car's speed automatically so drivers don't have to.

Since safety is valuable and human nature tends to be more comfortable and make life easier besides that technology is evolving so fast, Adaptive Cruise Control Systems have, in recent years, clearly become among the most actively discussed and researched topics. By all definitions, these systems, belong to the ADAS system.

Adaptive Cruise Control Systems will improve traffic safety, increase fuel efficiency. ACC research has been active for the last 20 years, with the recent Urban Challenge building tremendously on past successes. While the Urban Challenge was successful, there is much work to do as this technology becomes increasingly commercialized.

1.2 Objectives

Our aim in this project is to design and implement a low-cost Adaptive Cruise Control System that's implemented using the means of embedded system and infotainment system. Once the car starts ignition, the user accelerates the car manually and then he/she has two modes to choose from.

First mode is Normal Cruise Control (NCC), in this mode the car maintains its speed and the user can't accelerate the car above the maintained speed but can deaccelerate the car's speed, also if the front sensor detected an obstacle suddenly too close, the car stops immediately and Cruise Control is off just like emergency braking system.

The second mode is Adaptive Cruise Control (ACC), in this mode also the car maintains its speed and the user can't accelerate the car above the maintained speed but can deaccelerate the car's speed, but the difference is that the user selects the preferred distance range to keep between his/her car and other cars ahead, if a car ahead is slowing down or getting close the user's car also slows down so it keeps the selected distance between the two cars until it stops if the other car ahead got so close. Once the car in front starts moving again and the sensor not detecting obstacles ahead, the car starts accelerating again until it reaches the lately maintained speed on ACC mode, also if the front sensor detected an obstacle suddenly too close, the car stops immediately and Cruise Control is off just like emergency braking system.

Lastly the user can choose to turn off Cruise Control mode (CC OFF) and drive the car manually.

At the same time the distance and vehicle's speed displayed on the car's infotainment system which is also used to enables the user to choose between the Cruise Control modes and options and also have full control on these modes and options.

1.3 Motivation

The wide-scale adoption of adaptive cruise control systems (ACC) will lead to unprecedented economic, social and environmental change. For the public, the independence and freedom of personal travel will be available to almost everyone – youth, seniors and the physically, mentally and visually impaired. The expected reduction of road congestion would bring wide-ranging work and personal benefits. The gains from a drop in vehicle accidents and deaths are obvious.

In the near term, the arrival of the ACC pits the traditional evolutionary growth model of the legacy manufacturers against the riskier direct approach of nontraditional technology players.

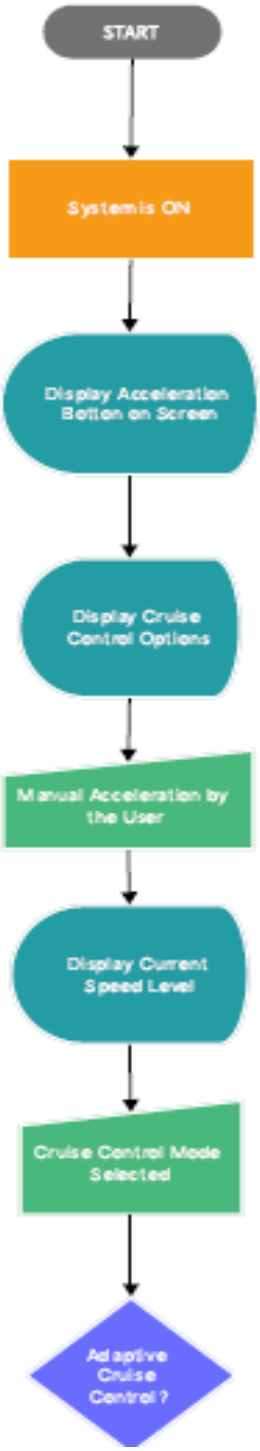
Autonomous Vehicles have many benefits to human being:

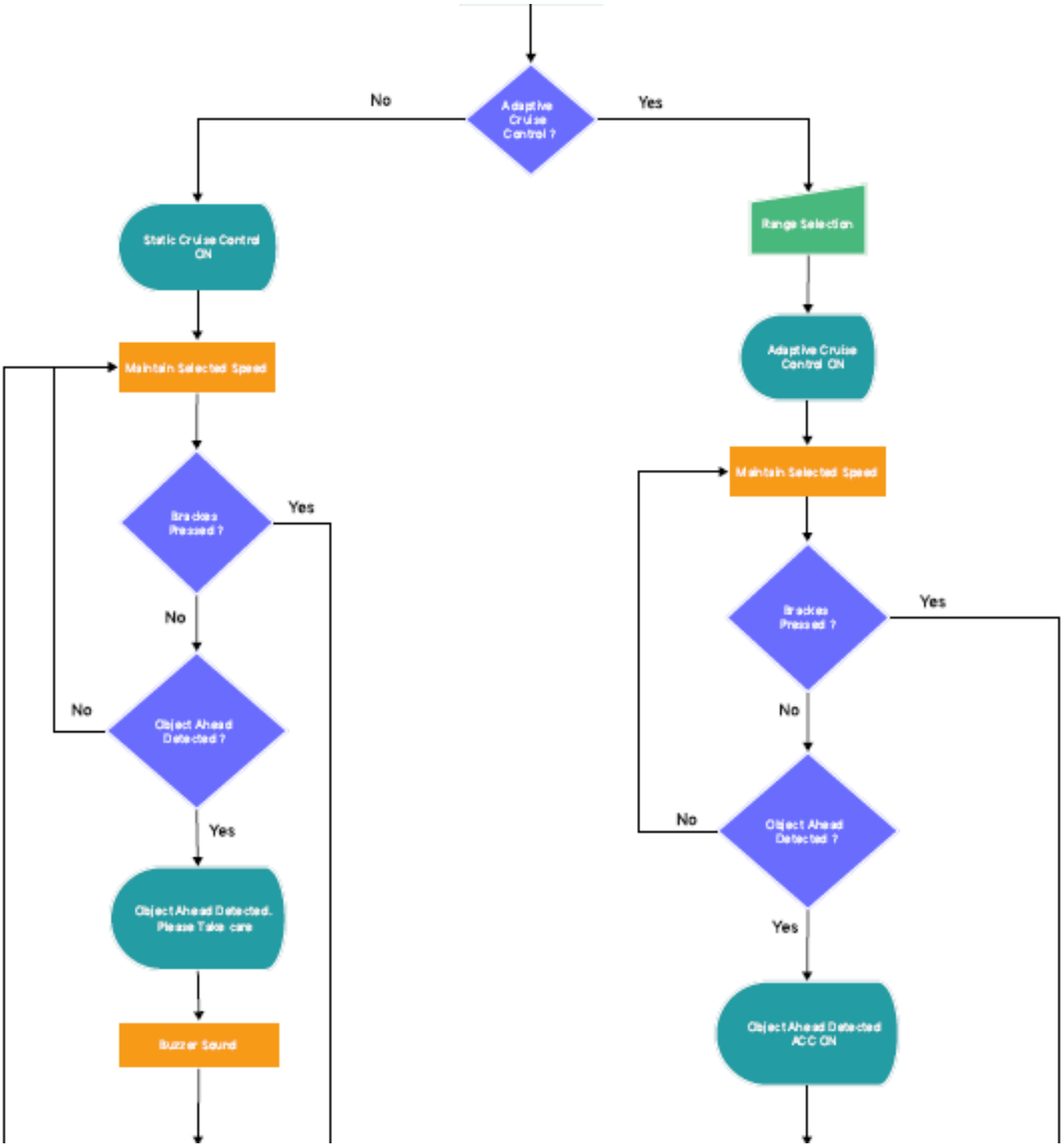
1. **Fewer Accidents:** The leading cause of most automobile accidents today is driver error. Alcohol, drugs, speeding, aggressive driving, over-compensation, inexperience, slow reaction time, inattentiveness, and ignoring road conditions are all contributing factors. Given some 40 percent of accidents can be traced to the abuse of drugs and or alcohol, self-driving cars would.

2. **Decreased Traffic Congestion**: One of the leading causes of traffic jams is selfish behavior among drivers. It has been shown when drivers space out and allow each other to move freely between lanes on the highway, traffic continues to flow smoothly, regardless of the number of cars on the road.

3. **Enhanced Human Productivity**: Currently, the time spent in our cars is largely given over to simply getting the car and us from place to place. Interestingly though, even doing nothing at all would serve to increase human productivity. Studies have shown taking short breaks increase overall productivity. Thus, doing nothing on your way to work could actually make you more productive when you get there.

1.4 System Flowchart




Adaptive Cruise Control ?
No
Yes
Static Cruise Control ON
Maintain Selected Speed
Brakes Pressed ?
Yes
No
Object Ahead Detected ?
No
Yes
Object Ahead Detected. Please Take care
Buzzer Sound
Range Selection
Adaptive Cruise Control ON
Maintain Selected Speed
Brakes Pressed ?
Yes
No
Object Ahead Detected ?
No
Yes
Object Ahead Detected ACC ON

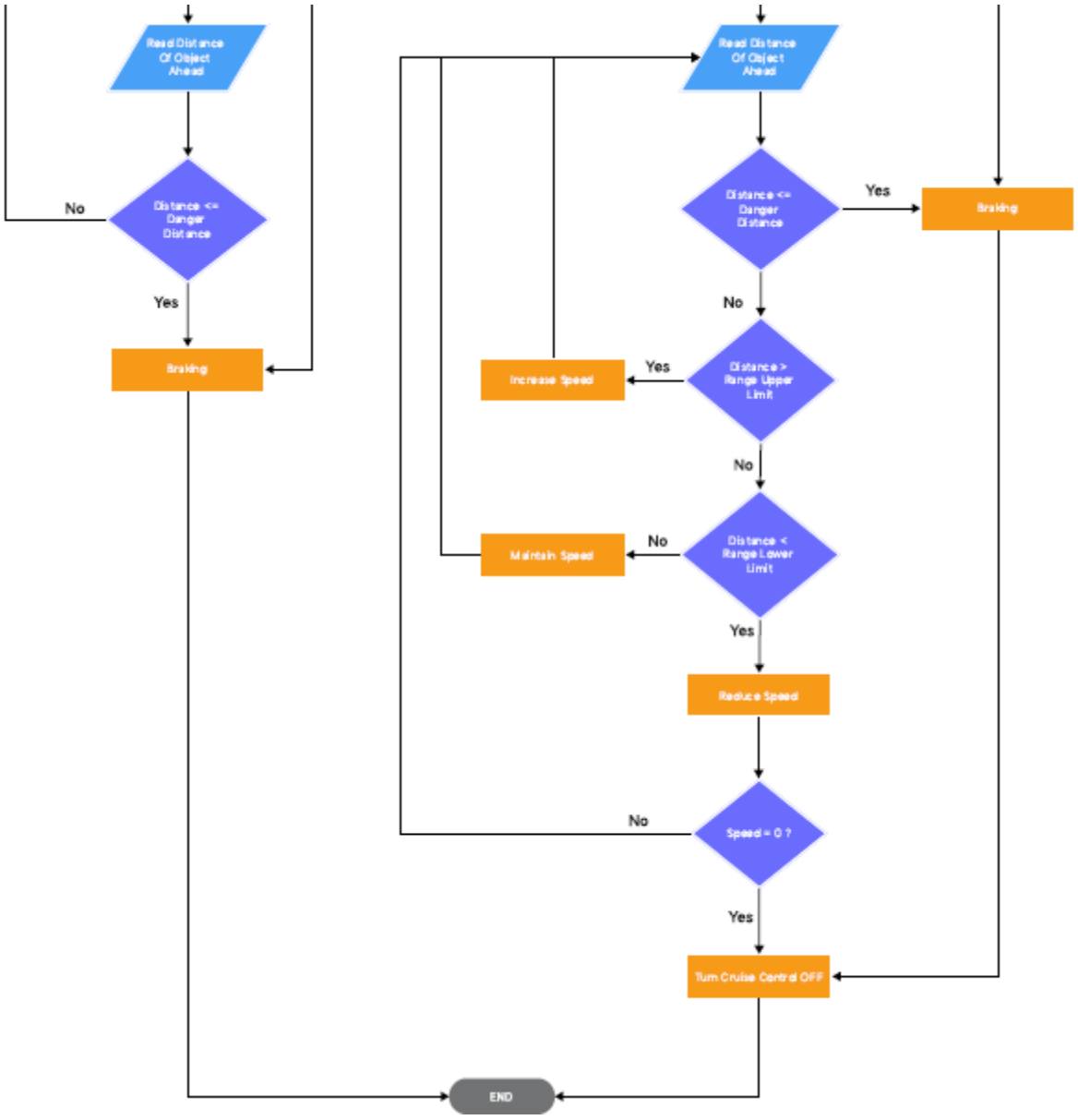


Figure 1. System Flowchart

1.5 Brief of Our Project

This project can be divided into 6 main phases:

1.5.1 System Start

When the vehicle starts ignition, all the system features are initialized and becomes ready to by applied on the system while running.

1.5.2 Movement

The user starts accelerating the car to move it forward and start driving and can deaccelerate it too.

1.5.3 Choosing between cruise control modes

The user starts choosing preferred cruise control mode between the two modes available through the infotainment system.

1.5.4 NCC mode

If the user chooses NCC mode the car maintains its speed and the user can't accelerate the car above the maintained speed but can deaccelerate the car's speed.

1.5.5 ACC mode

If the user chooses ACC mode the car maintains its speed and the user can't accelerate the car above the maintained speed but can deaccelerate the car's speed. The user also chooses preferred distance range between the three choices available, "close", "mid" and "far" range to keep between the car and other cars in front, this is maintained by the sensor in the car's preface. The default distance range kept between the car and obstacles ahead is "close" range when choosing ACC mode but the user can change it.

1.5.6 Sensor Detecting & Controlling

Whether the user chooses NCC or ACC mode, the front sensor starts detecting for any obstacles or cars ahead and if an obstacle appeared suddenly in front of the car (or sensor), the car stops immediately (Emergency Braking System) and Cruise Control mode is off.

In ACC mode after choosing preferred distance range, if a car ahead is slowing down or getting close the user's car also slows down so it keeps the selected distance between the two cars until it stops if the other car ahead got so close. Once the car in front starts moving again and the sensor not detecting obstacles ahead, the car starts accelerating again until it reaches the lately maintained speed on ACC mode.

1.5.7 Turning Off Cruise Control

If the user decided to turn off Cruise Control mode he/she can choose (CC OFF) mode in which the car is controlled manually completely until the user turns on Cruise Control mode again.

1.5.8 Infotainment System

The Infotainment System Displays speed of the car, distance of obstacles ahead if there's any and reactive GUI buttons to interact with the infotainment system which enables the user to not only fully control the cruise control system but also get info about car's speed and detected distance.

1.6 State Machine

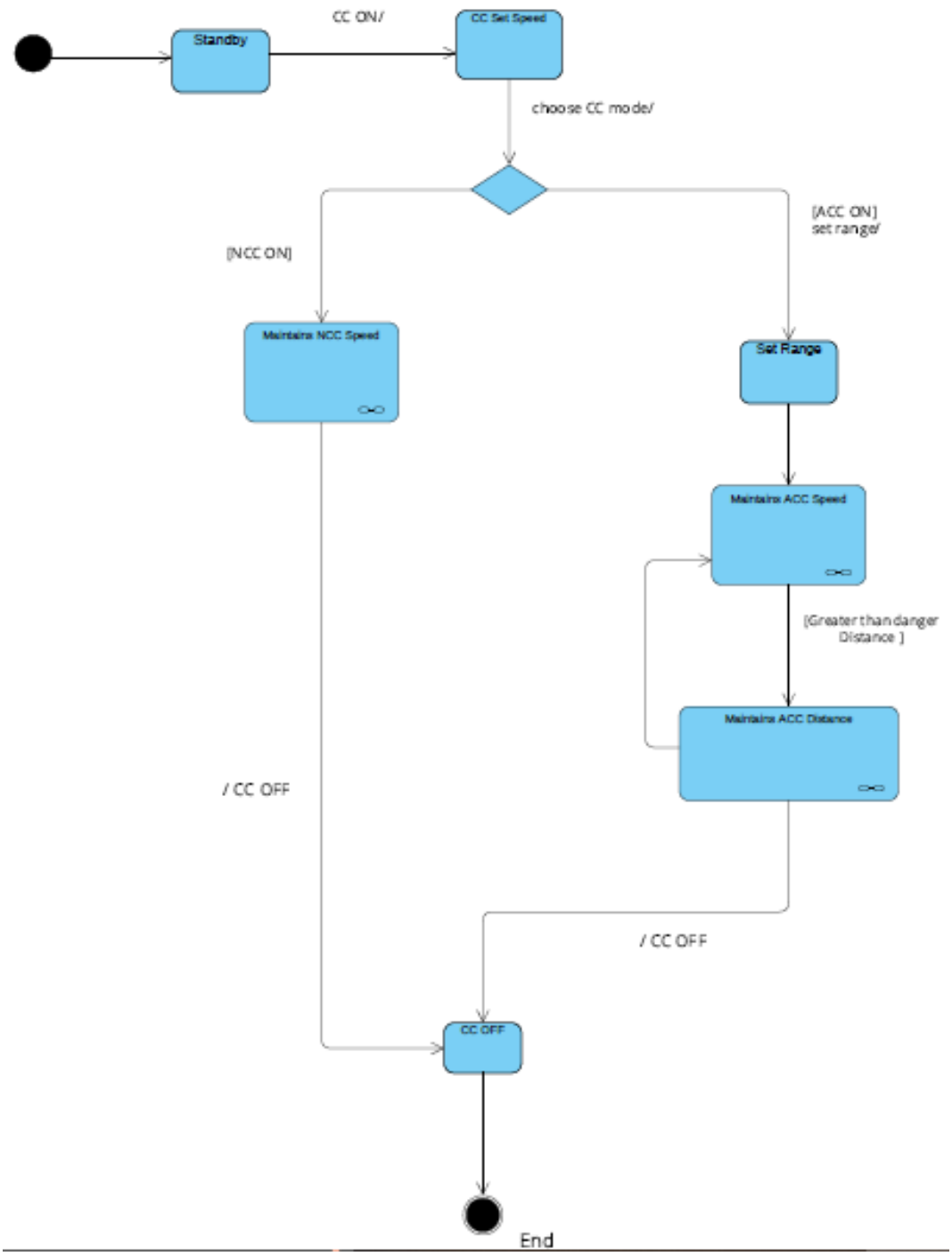
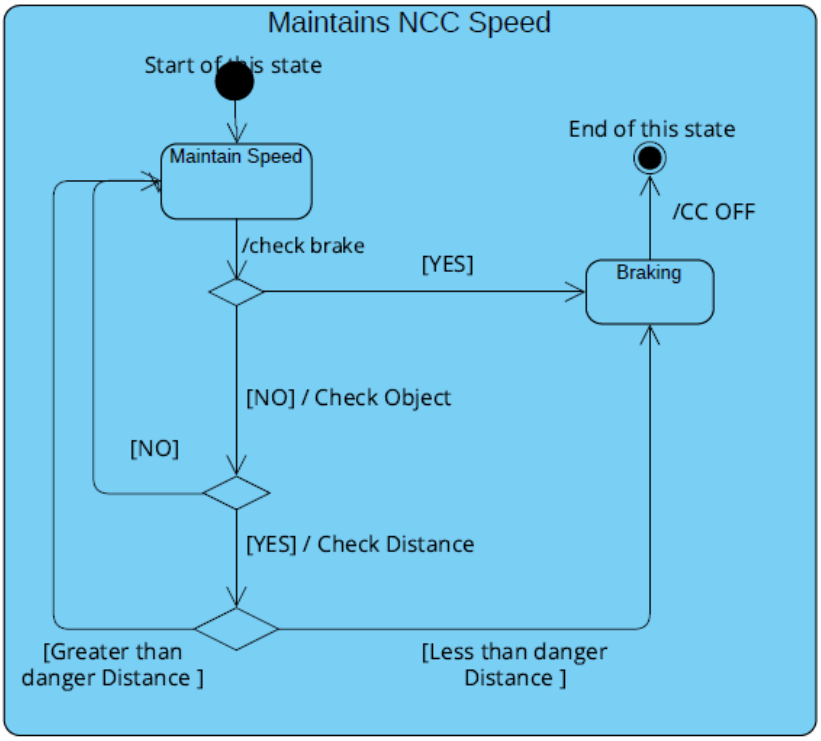
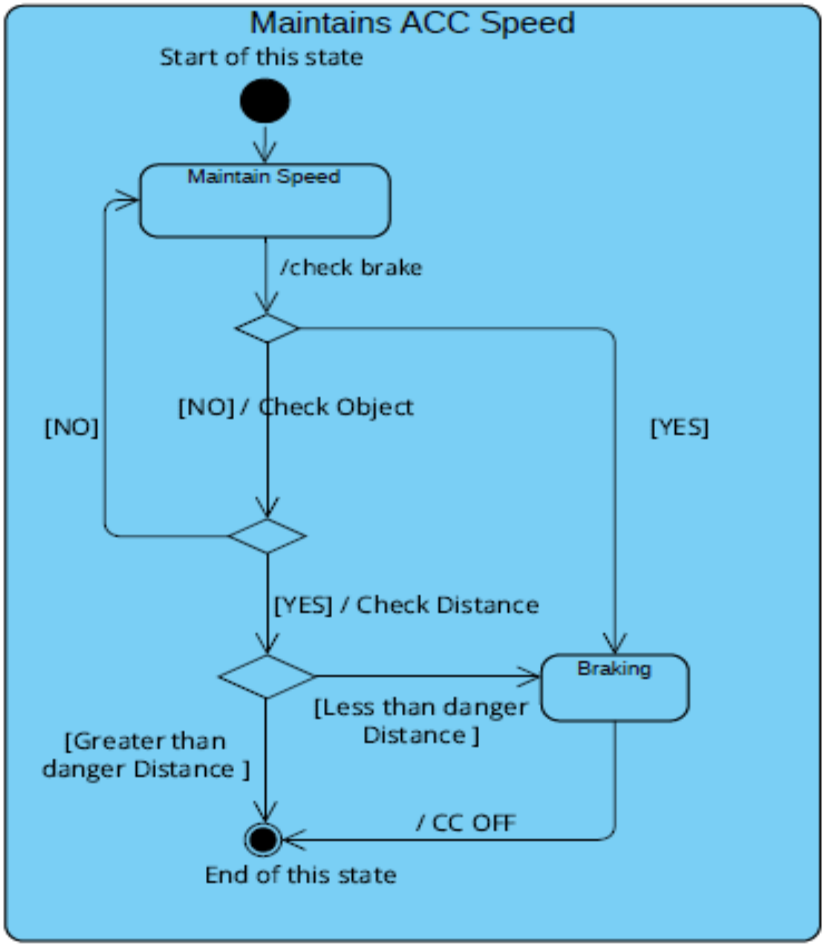
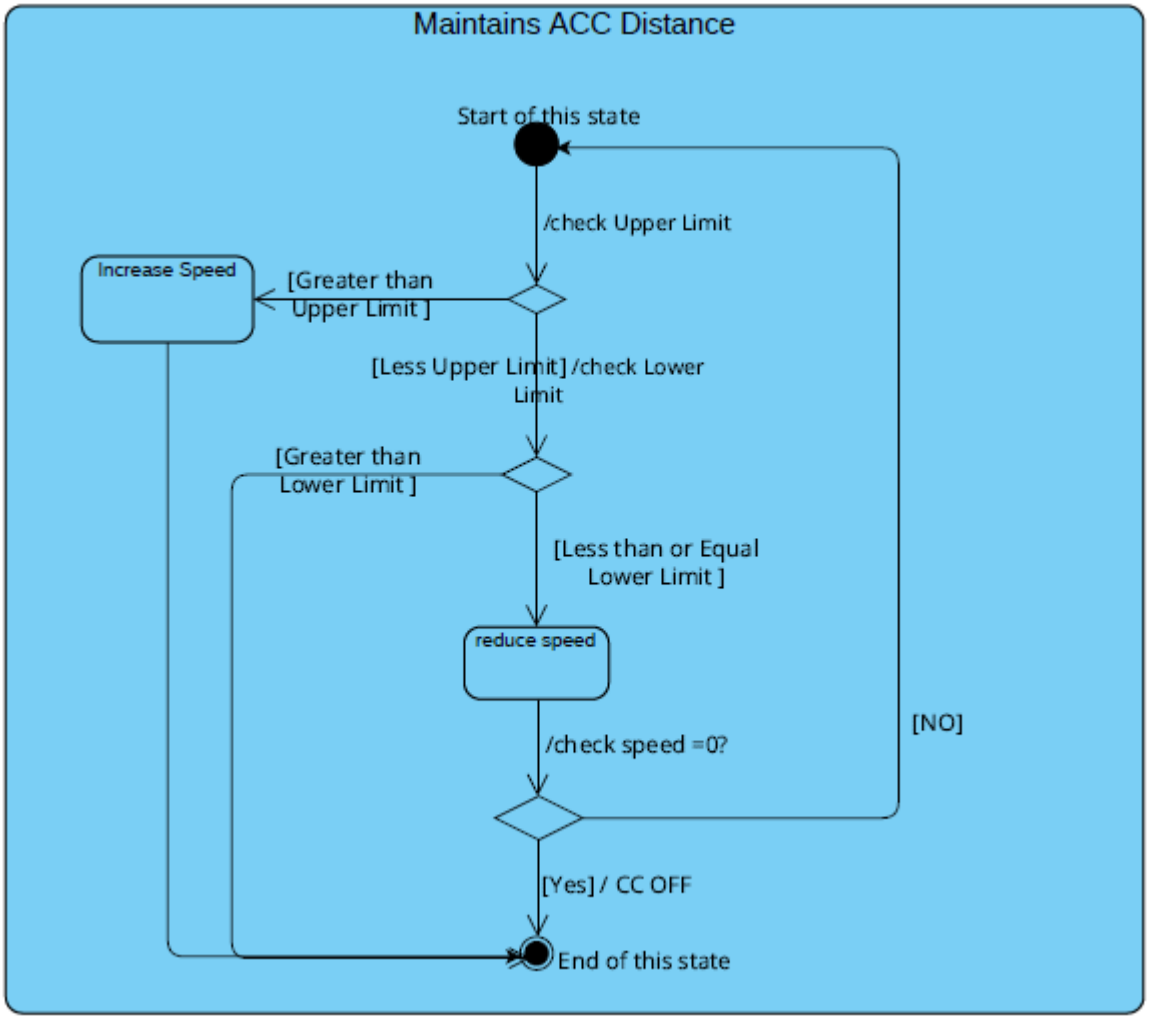


Figure 2. System Block Diagram


Maintains NCC Speed
Start of this state
Maintain Speed
End of this state
/CC OFF
/check brake
[YES]
Braking
[NO] / Check Object
[NO]
[YES] / Check Distance
[Greater than danger Distance]
[Less than danger Distance]


Maintains ACC Speed
Start of this state
Maintain Speed
/check brake
[NO]
[NO] / Check Object
[YES]
[YES] / Check Distance
Braking
[Less than danger Distance]
[Greater than danger Distance]
/ CC OFF
End of this state


Maintains ACC Distance
Start of this state
/check Upper Limit
Increase Speed
[Greater than
Upper Limit]
[Less Upper Limit] /check Lower
Limit
[Greater than
Lower Limit]
[Less than or Equal
Lower Limit]
reduce speed
/check speed =0?
[NO]
[Yes] / CC OFF
End of this state

1.8 Used Components

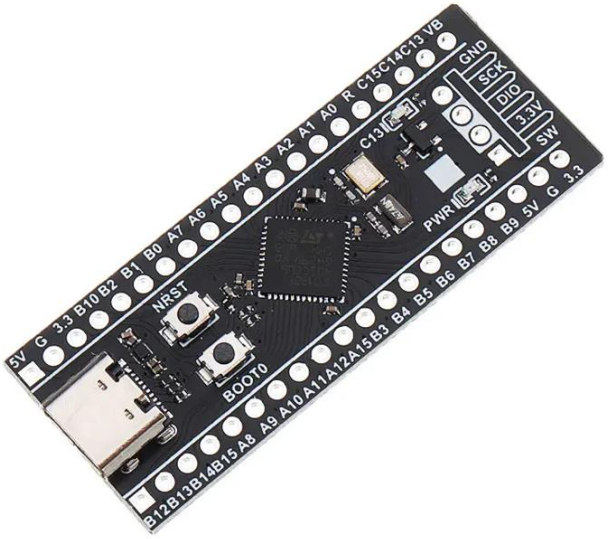


Figure 3. STM32F401CC Microcontroller

Figure 4. Raspberry Pi 3 B+

Figure 6. DC Motor

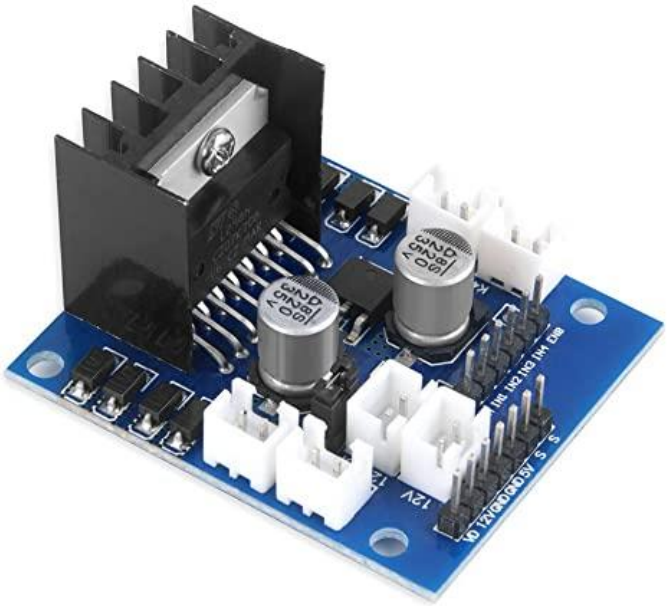


Figure 7. Motor Driver

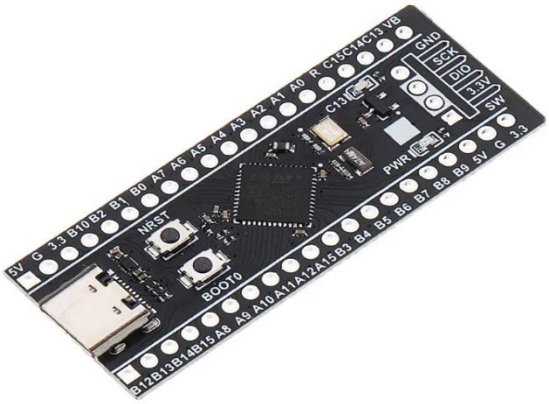
Figure 8. Ultrasonic Sensor

Figure 9. LCD Touch Screen

Chapter 2 - STM32F401CC Microcontroller

2.1 Introduction

The Cortex-M4 processor is a high performance 32-bit processor designed for the microcontroller market. Based on the high-performance Arm® Cortex® -M4 32-bit RISC core operating at a frequency of up to 84 MHz



2.2 Key Features

Consider some general features of STM32F401CC microcontroller is:

- Up to 256 Kbytes of Flash memory
- 512 bytes of OTP memory
- Up to 64 Kbytes of SRAM
- Debug mode: Serial wire debug (SWD) & JTAG
- Clock, reset and supply management
- General-purpose DMA
- Up to 11 timers: up to six 16-bit, two 32-bit timers up to 84 MHz
- Up to 81 I/O ports with interrupt capability
- Up to 11 communication interfaces
- Up to 3 USARTs
- CRC calculation unit
- 96-bit unique ID
- RTC: subsecond accuracy, hardware calendar
- Dynamic efficiency line with BAM (batch acquisition mode).

2.3 STM32 Microcontroller Pin Diagram

For explaining the STM32F401CC Microcontroller Pin diagram, consider a 40-pin Dual Inline Package (DIP) of microcontroller integrated circuit is:

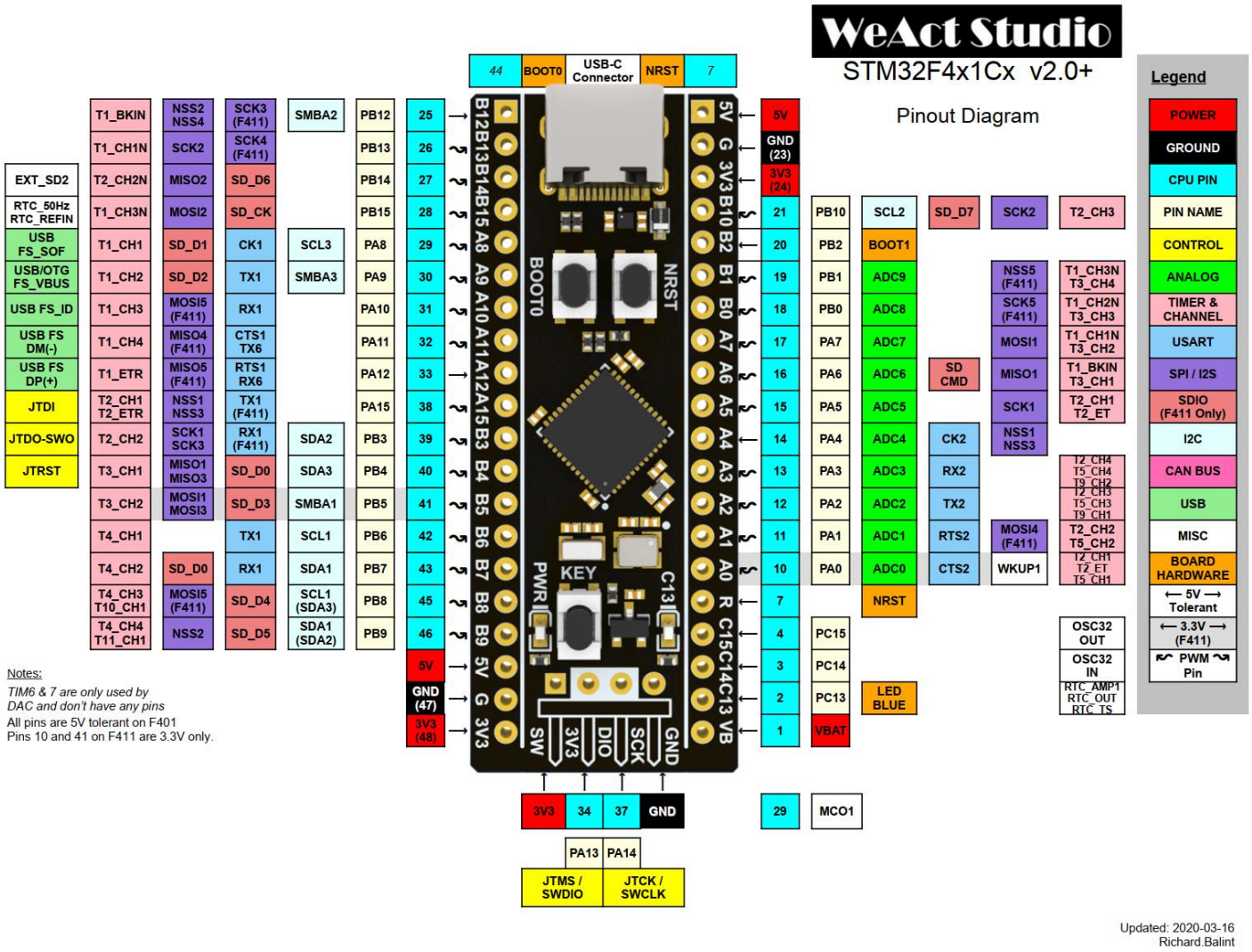


Figure 5. STM32 Pinouts Diagram

Chapter 3 - Raspberry Pi 3

3.1 Hardware Description

3.1.1 Design

The Raspberry Pi is a credit card-sized single-board computer. it is a full single-board computer, not just a single-chip microcontroller. The first-generation Pi was developed by the Raspberry Pi Foundation in Cambridge, England, in 2011. It uses a Broadcom system-on-chip CPU with USB and Ethernet communication ports, HDMI monitor output, special camera interface, and in the current versions even Wi-Fi and Bluetooth . So if you connect monitor, keyboard and mouse, you will have a fully functional PC. There are several different operating systems available for the Pi, most of them dialects of Linux, such as the Raspberry Pi OS (originally called Raspbian). All software, operating system and user programs reside on an external SD card (on the first-generation Pi this was a full-size SD card, now micro-SD cards are used). This solution has pros and cons. The advantages are that within seconds, the complete system software can be swapped out by simply replacing one SD card with another and different system versions can be kept on separate SD cards. The disadvantages, however, are that an additional component has to be purchased and that SD card data can easily get corrupted so that the Pi no longer boots up.

There are currently five Raspberry Pi models in market i.e. the Model B+, the Model A+, the Model B, the Model A, and the Compute Module (currently only available as part of the Compute Module development kit). All models use the same SoC (System on Chip - combined CPU & GPU), the BCM2835, but other hardware features differ. The A and B use the same PCB, whilst the B+ and A+ are a new design but of very similar form factor .The Compute Module is an entirely different form factor and cannot be used standalone.

They have variant per model: An entry-level version with 1GB of RAM, designed to hit the increasingly-challenging and shrinking-through-inflation $35 target price point; a more expensive 2GB version, designed as to the go-to model; and a 4GB variant for power users.

In this project, we have used the model B 4GB RAM for GPU and CPU computations. It comprises of other models with two USB ports and a 10/100 Ethernet controller.

The Raspberry Pi 4 B (4GB) has been the most popular version since launch, largely thanks to a relatively small price difference between it and the now-retired 1GB which gets you four times the RAM. It was also the highest capacity version prior to the release of the new 8GB model, and as a result the go-to choice for power users. Having 4GB of RAM offers you a little more breathing room. You won’t see too much of a difference for basic desktop use - 2GB is plenty there - but if you find yourself multitasking the extra RAM will definitely come in handy. There are other reasons to go 4GB, too: The additional 2GB of RAM can serve as file cache, speeding up commonly accessed data which would otherwise have to come via USB or the relatively slow microSD card. It allows you to dedicate more RAM to the GPU, which shares its memory with the CPU - though you won’t find that making any difference to the maximum supported display resolutions. For most users, 4GB will be plenty - and if you picked one up before the launch of the new 8GB model.

3.1.2 Performance

While operating at 700 MHz by default, the first-generation Raspberry Pi provided a real-world performance roughly equivalent to 0.041 GFLOPS. On the CPU level the performance is similar to a 300 MHz Pentium II of 1997–99. The GPU provides 1 Gpixel/s or 1.5 Gtexel/s of graphics processing or 24 GFLOPS of general purpose computing performance. The graphical capabilities of the Raspberry Pi are roughly equivalent to the performance of the Xbox of 2001. The Raspberry Pi 4, with a quad-core Cortex-A72 processor.

Overclocking:

Most Raspberry Pi systems-on-chip could be overclocked to 800 MHz, in extreme cases, even to 1500 MHz (discarding all safety features and over-voltage limitations). In Raspberry Pi OS the overclocking options on boot can be made by a software command running "sudo raspi-config" without voiding the warranty.

The speed of the new Raspberry Pi 4 is a step up from earlier models. For the first time, we've built a complete desktop experience. Whether you're editing documents, browsing the web with a bunch of tabs open, juggling spreadsheets or drafting a presentation, you'll find the experience smooth and very recognizable — but on a smaller, more energy-efficient and much more cost-effective machine.

Fast networking: Raspberry Pi 4 comes with Gigabit Ethernet, along with onboard wireless networking and Bluetooth.

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

USB 3: Your new Raspberry Pi 4 has upgraded USB capacity: along with two USB 2 ports you'll find two USB 3 ports, which can transfer data up to ten times faster.

- **Features offered in Raspberry Pi Model B:**

  - Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
  - 1GB RAM
  - BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
  - 100 Base Ethernet
  - 40-pin extended GPIO
  - 4 USB 2 ports
  - 4 Pole stereo output and composite video port
  - Full size HDMI
  - CSI camera port for connecting a Raspberry Pi camera
  - DSI display port for connecting a Raspberry Pi touchscreen display
  - Micro SD port for loading your operating system and storing data
  - Upgraded switched Micro USB power source up to 2.5A

3.1.3 Connectors

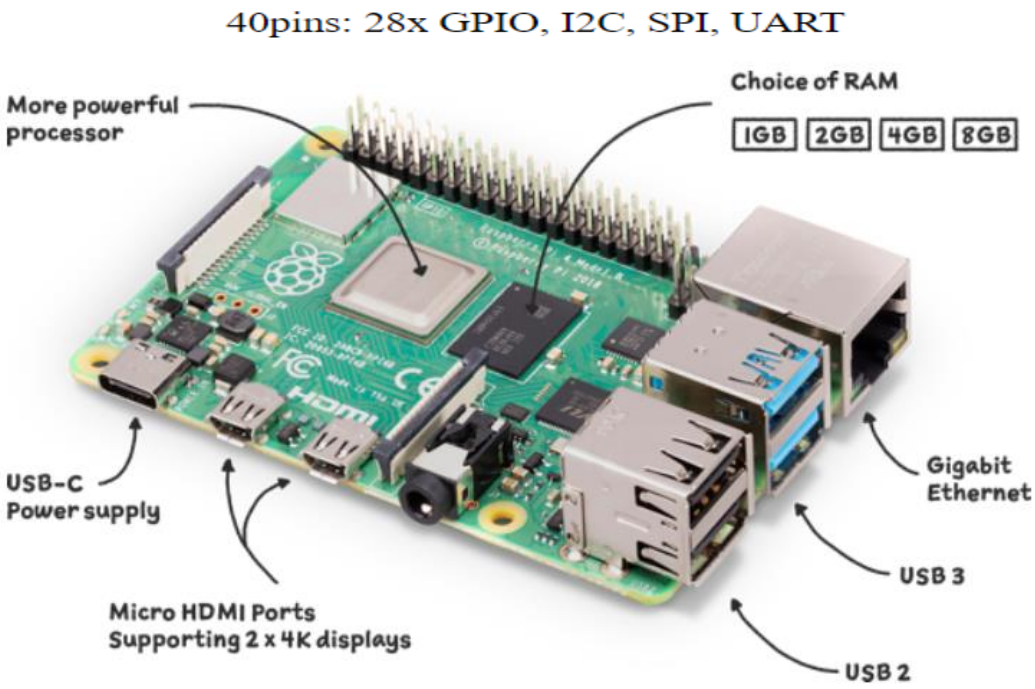


Figure 6. Location of connectors and main IC's

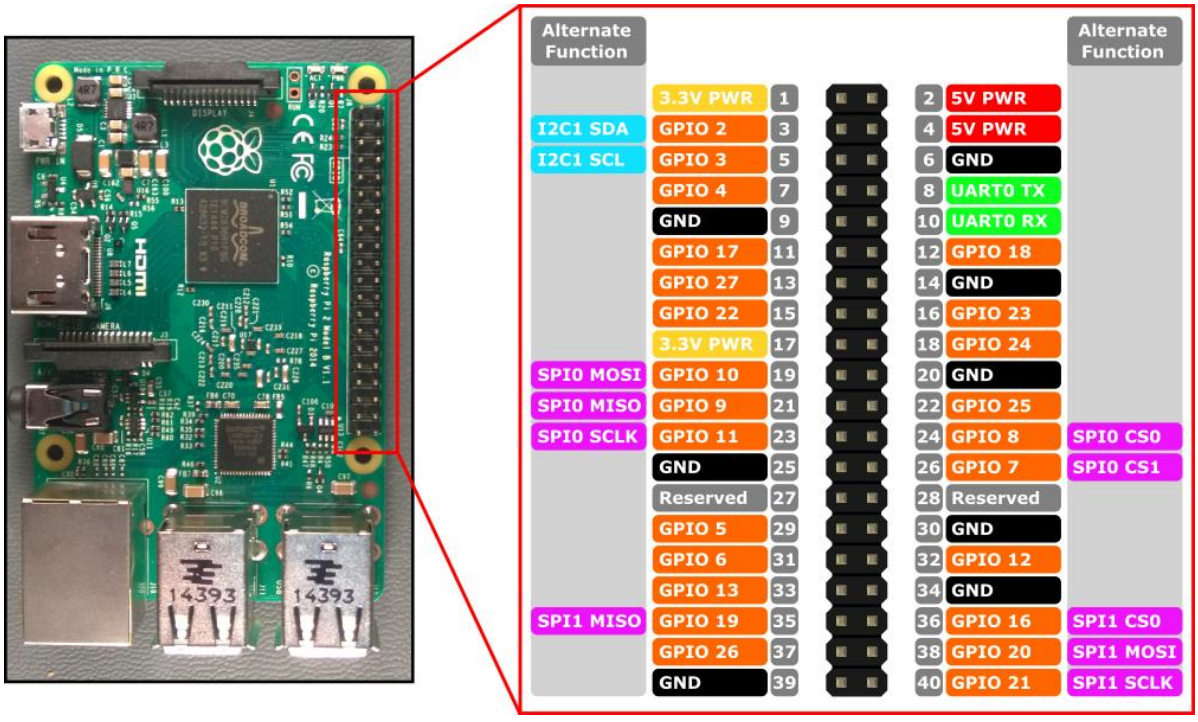


Figure 19. Raspberry pi 3 pinout

- **J8 header and general-purpose input-output (GPIO):**

Raspberry Pi have 40-pin pinout called *J8 header*. The J8 header is commonly referred to as GPIO connector as a whole even though only a subset of the pins are GPIO pins.

- **Power and Ground Pins:**

The power and ground pins are used to power external circuitry. have two 5V pins and two 3.3V pins.

3.1.4 Accessories

1- Cable USB

Figure 20. Raspberry pi 3 pinout

2- SanDisk 8 GB Flash Memory Card

Figure 20. SanDisk 8 GB Flash Memory Card

3- HDMI cable

4- LCD touch screen

We have used it to display the Graphical User Interface (GUI).

Software Description

3.2.1 Software Description Raspbian OS

Of all the operating systems Arch, Risc OS, Plan 9 or Raspbian available for Raspberry Pi, Raspbian comes out on top as being the most user-friendly, best-looking, has the best range of default software's and optimized for the Raspberry Pi hardware. Raspbian is a free operating system based on Debian (LINUX), which is available for free from the Raspberry Pi website.

3.2.2 Python

Python is a widely used general-purpose, high-level programming language . Its syntax allows the programmers to express concepts in fewer lines of code when compared with other languages like C, C++or java.

3.3.3 RPi.GPIO Python Library

The RPi.GPIO Python library allows you to easily configure and read-write the input/output pins on the Pi's GPIO header within a Python script. This package is not shipped along with Raspbian.

3.3.4 Raspberry Pi Operating System and Setup

We will only be using the Raspberry Pi OS (Raspbian), which is sufficient for most applications. It should be noted that although this is not a real-time operating system, it is sufficient for most robotics applications. Even complex robotics tasks rarely require real-time operation, as long as all sensor data can be marked with real-time clock readings. Raspbian is the Raspberry Pi's most popular operating system, a spin-off of the Linux distribution Debian that works well on the Raspberry Pi's hardware. Raspbian is a competent and versatile operating system that gives your Raspberry Pi all the comforts of a PC: a command line, a browser, and tons of other programs. You can use a Raspberry Pi running Raspbian as a cheap and effective home computer, or you can use it as a springboard and turn your Raspberry Pi into any of countless other functional devices, from wireless access points to retro gaming machines. Here's how to install Raspbian on the Raspberry Pi.

Installing Raspbian on the Raspberry Pi is pretty straightforward. We'll be downloading Raspbian and writing the disc image to a microSD card, then booting the Raspberry Pi to that microSD card. For this project, you'll need a microSD card (go with at least 16 GB).

Step 1: Download Raspbian

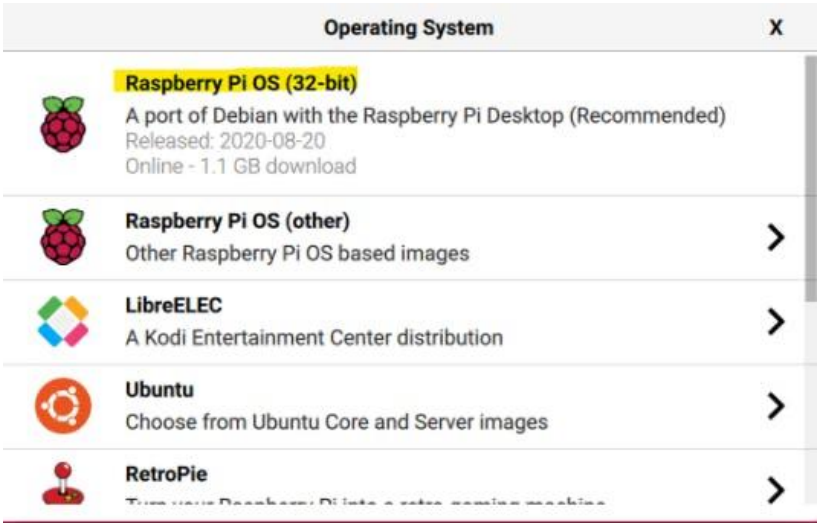
and download the Raspbian disc image.

1. Write the OS into your SD Card

- Connect your microSD card to your computer. (You might need to use a SD Card Adapter).

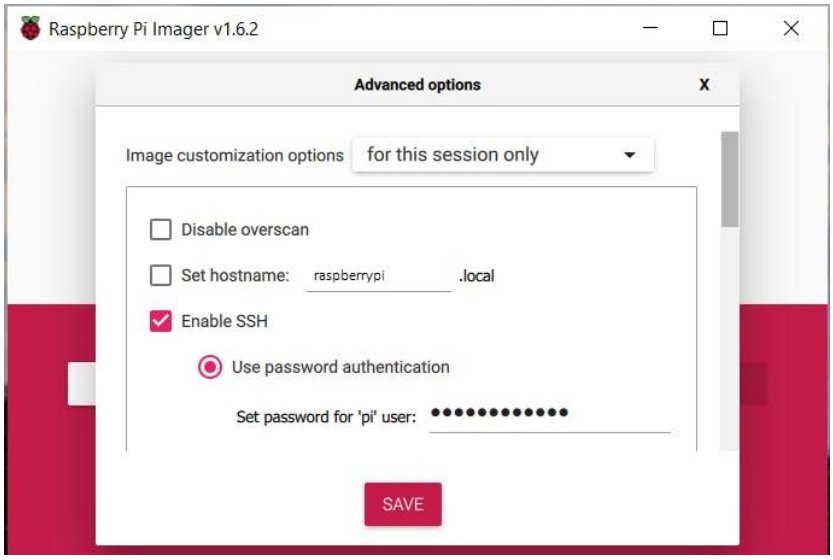
- Open RPi installer, and Click 'Choose OS'.:
- Click on 'Raspberry Pi OS (32-bit)'/The one on the top.
- Click 'Choose Storage' and select your SD Card reader.

p.s. **DO NOT** click "Write" yet.



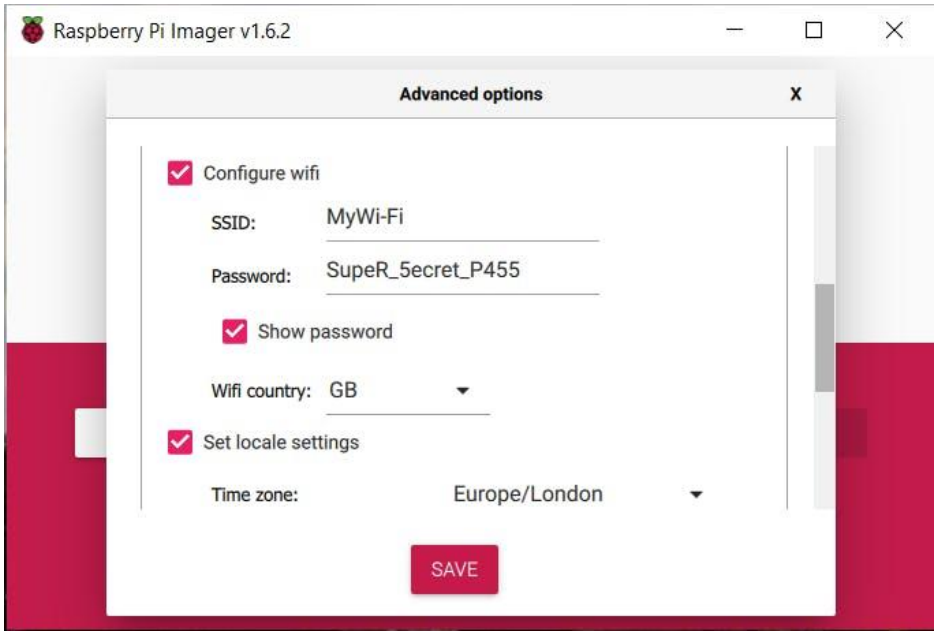
2. SSH and Wi-Fi Enabling

- Press **Ctrl + Shift + X** to bring up the Advanced options menu.
- Tick **Enable SSH** tick boxes in the menu.
- Give your RPi a password for SSH.



- Also, tick **Configure Wifi** and type your router's **SSID (Name)** and **Password** into the provided spaces.
- Change your **Wifi Country** if it needs adjustments.

- Change **Locale Settings** if desired (It can be done later).



- Click **SAVE** followed by clicking **WRITE**.
- Wait till the OS is written and Verified.

p.s. **DO NOT** plug the microSD into the RPi yet.

3. Install VNC Viewer on your computer.

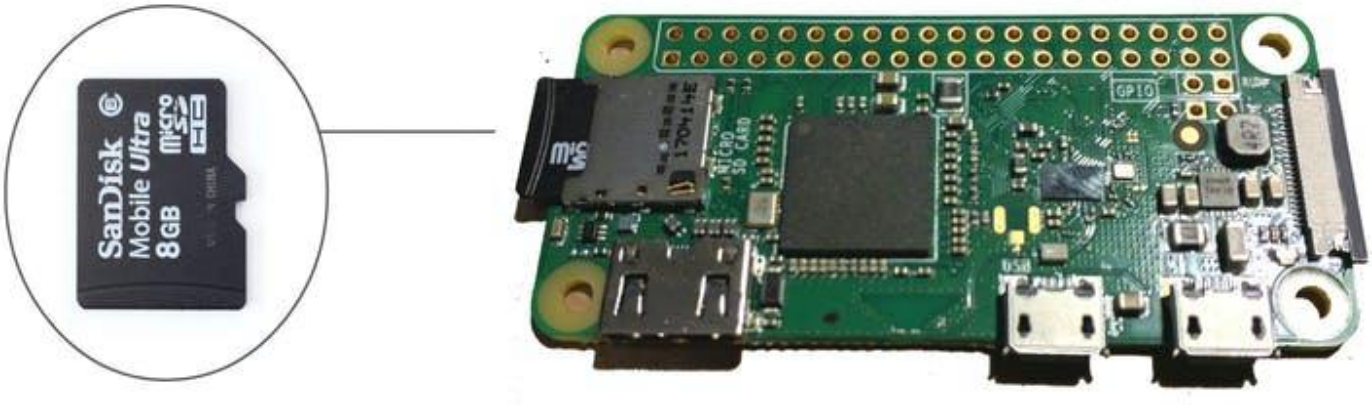
Go to the following link, download and install VNC Viewer on your computer:

https://www.realvnc.com/en/connect/download/viewer/

p.s. Make sure you install 'VNC Viewer', and **NOT** 'VNC Server'

4. Insert your microSD card into your RPi

After verifying the OS, go ahead and get your microSD card from your SD card Adapter, and plug it into your RPi.



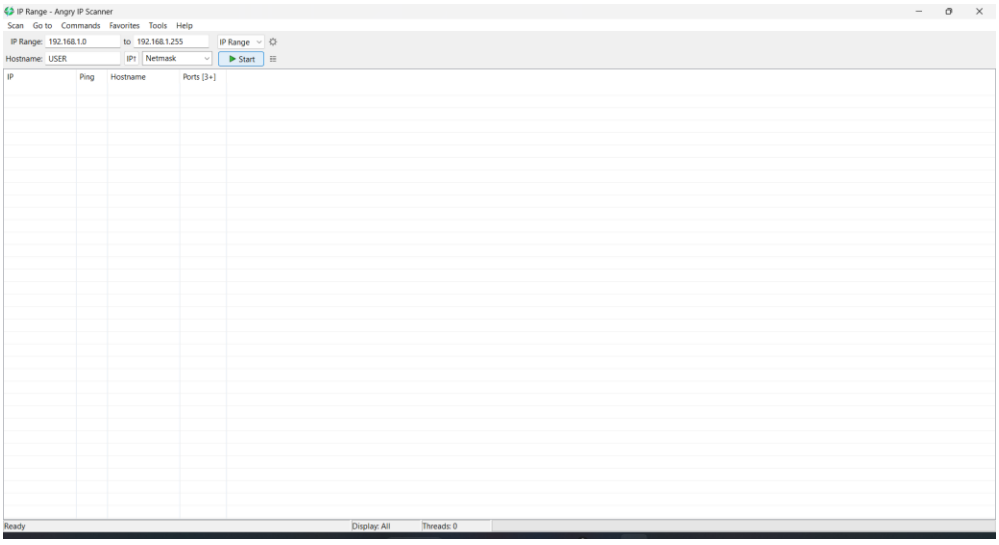
Then plug in your RPi into it's power source.

4. Find your RPi's IP Address

Go to the following link, download and install Angry IP scanner on your computer:

https://www.realvnc.com/en/connect/download/viewer/

after installing, open it and press “start”, you will find your raspberry pi IP address among the Ips.



There is another method to get your raspberry pi IP address, open the web page account of your router and you will find the IP addresses of all the devices connected to your router.

7. Get into your Pi

Type on **your** terminal:

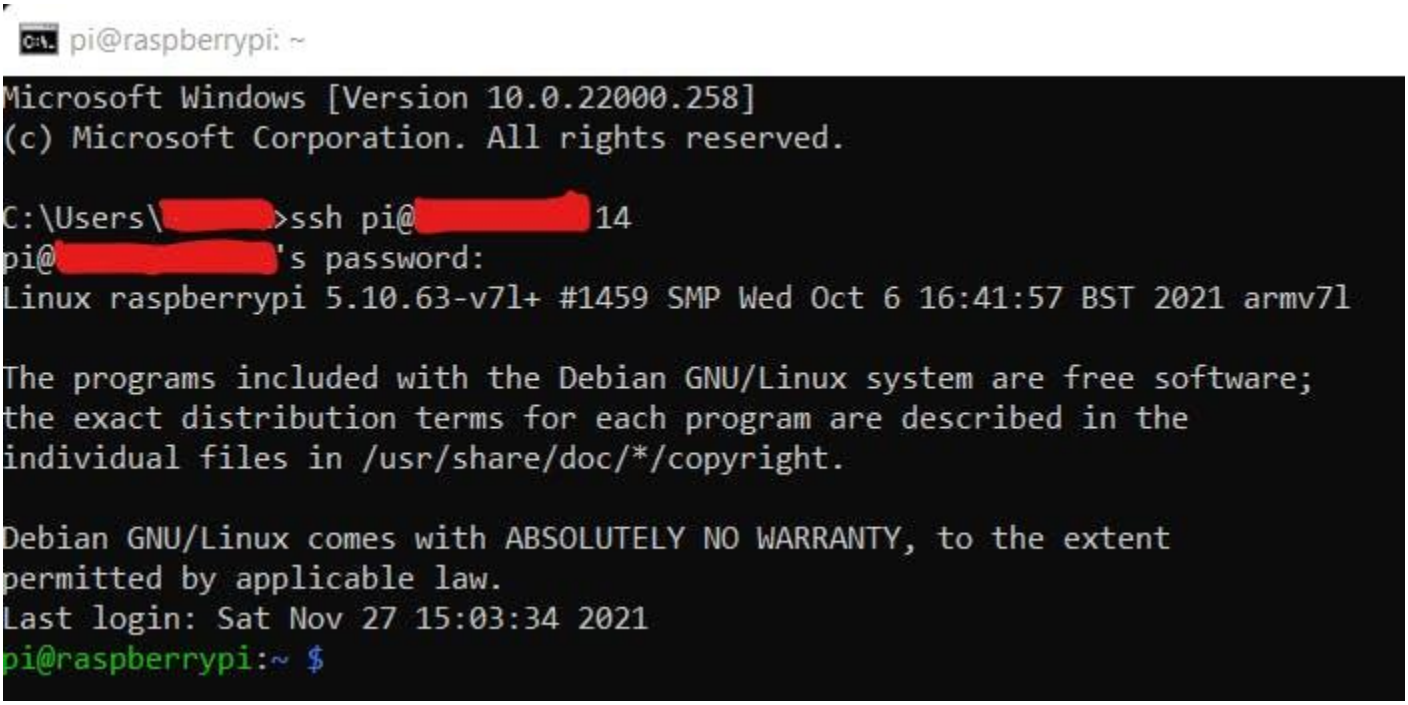
```
ssh pi@'IP'
```

Use the RPi's IP that you got.

Then you will find a prompt asking for your password.

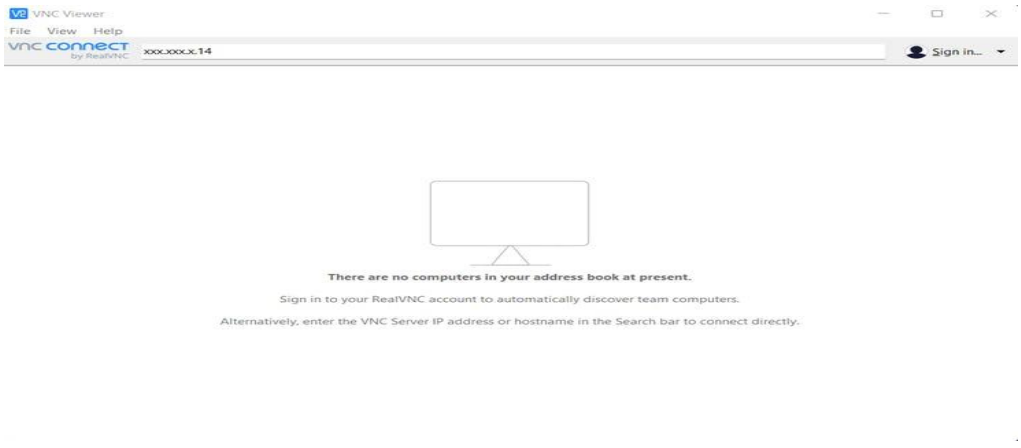
Type the password that you entered in the RPi Imager for SSH, to get into your Pi.

Hit **Enter** and:



8. View desktop of RPi (Finally!)

Launch VNC Viewer on your computer, type your RPi's IP:



-Press Enter.

-Type in **pi** for the username.

-Type in your password.

You should now see your Desktop:

Step2: Install package management tool pip

- If it isn't already installed, install the package management tool, pip by running the following command**:**
  sudo apt-get install python3-pip

```
pi@raspberrypi:~ $ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-pip is already the newest version (18.1-5+rpt1).
The following packages were automatically installed and are no longer required:
  python3-pyperclip python3-thonny rpi.gpio-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
```

Step 3: Install PyQT5 library in your computer

Open your computer's terminal window, then write this command:

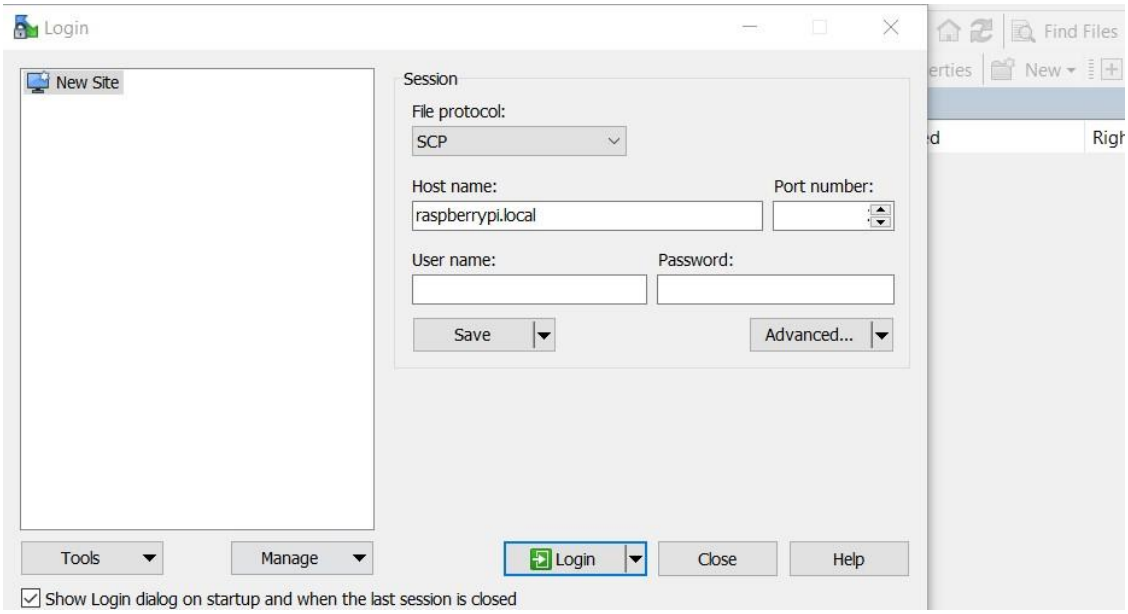
```
pip install pyqt5
```

you can then use any text Editor -like Notepad++ - and then transfer them to your raspberry pi.

Step 4: Download and Install WinSCP program

To transfer the files and folders from and to your PC and raspberry pi, install WinSCP program from the official site here: https://winscp.net/eng/download.php

When you install the program, open it and write your raspberry pi host name, raspberry pi username and password. Then now you are ready to transfer files to your pi from the PC.



Step 5: Download and Install QT Designer

From the official website https://build-system.fman.io/qt-designer-download

Download the QT designer to design your graphical user interface (GUI).

Our GUI Design will be like this:

In our GUI we have used, the speed meter widget, labels, LCD to display the current speed in digital way, push Buttons, LineEdit to display the mode, state, and current distance.

Chapter 4 - DC Motors

4.1 Overview

In this chapter, we describe the motion of the automotive car using DC motors and L298N Drivers.

We interface ATmega32 with 12v DC motors and L298N motor drivers to control the movement of our autonomous car with controllable speed and two possible directions of motion, i.e. forward and backward.

4.2 DC Motors Fundamentals

DC motors consist of rotor (or armature), commutator, brushes, rotating shaft, bearings, and stator with permanent magnet, figure 19 illustrates the structure of DC motors. The principle of operation with a simple two-pole dc motor: The torque is produced by the fact that like field poles attract and unlike poles repel.

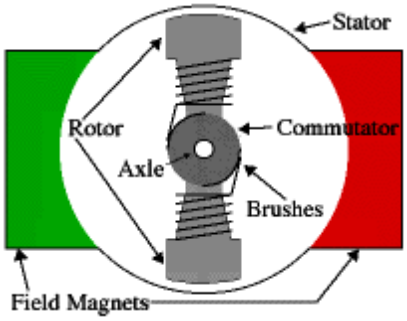


Figure 7. Structure Of a DC Motor

4.3 Working Principle

DC Motor converts DC power into mechanical power. Field poles are supplied by DC excitation current, which produces a DC magnetic field. Armature winding (conductor) is supplied by DC current through the brushes and commutator. So, based on the principle of Lorentz Law that when a current carrying conductor is placed in a magnetic field, the conductor experiences a mechanical force. All the conductors placed on the periphery of a DC motor are subjected to these forces. These forces cause the armature to rotate in the clockwise direction. Therefore, the armature of a DC motor rotates in the direction of the torque developed by the motor.

4.4 Interfacing with STM32

Since the output of the microcontroller is 5 volts maximum with very small current whereas, a decent DC motor needs at least 5v or 12v, the microcontroller cannot drive the DC motors directly. So, in order to amplify this voltage, a motor driver IC is used. In our project we use the L298N Dual H-Bridge driver which can drive motors up to 36v and can provide a drive current of 2A. Figure 20 demonstrates the block diagram of interfacing the DC motor with ATmega32.

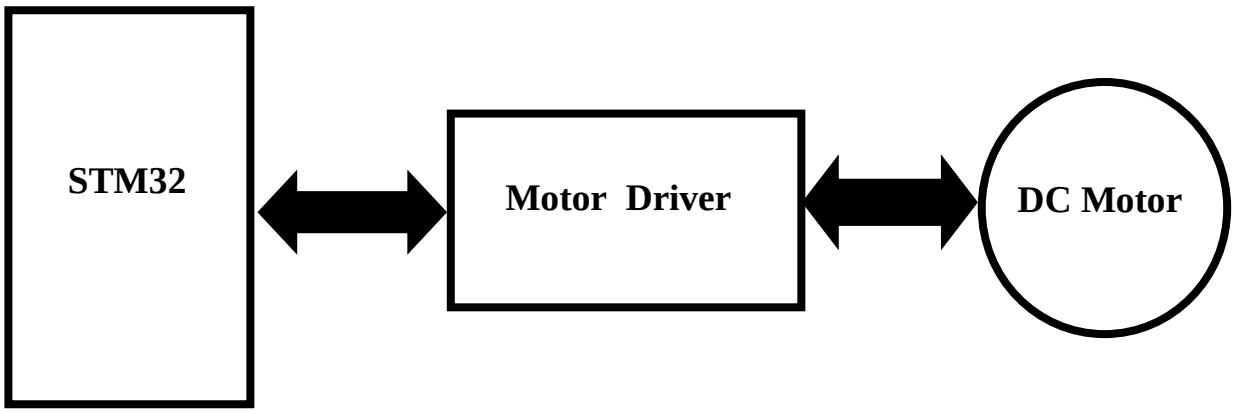


Figure 8.Block Diagram of DC Motor Interfacing

4.4.1 H-Bridge Circuit

The H-Bridge circuit shown in figure 13, is used to control the rotation by simply switching the direction of the current flow through the motor. It consists of four switching elements, transistors or MOSFETs, with the motor at the center forming an H-like configuration. By activating two switches while deactivating the other two, we are able to change the direction of the current flow. Therefore, rotating the motor in clockwise or anticlockwise direction.

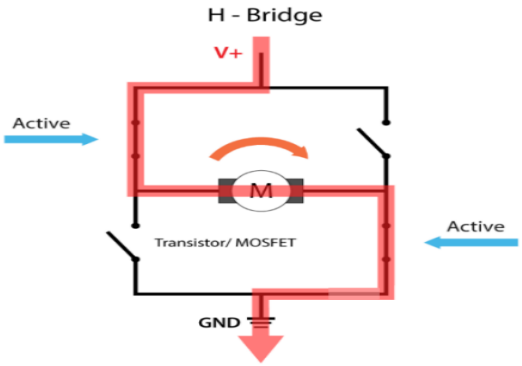


Figure 9. H Bridge Circuit

4.4.2 L298N Driver

L298N-Pinout:

L298N is a dual H-Bridge driver which allows speed and direction control of two DC motors at the same time. It can drive DC motors with voltages between 4.5 υ to 36 υ, with a current up to 2 A. the pin out of l298N is demonstrated in figure 22. It has three screw terminal blocks, two for the DC motors 1 and 2, where the other one is for the GND pin, the VCC for motor, and a 5V pin which can either be an input or output.

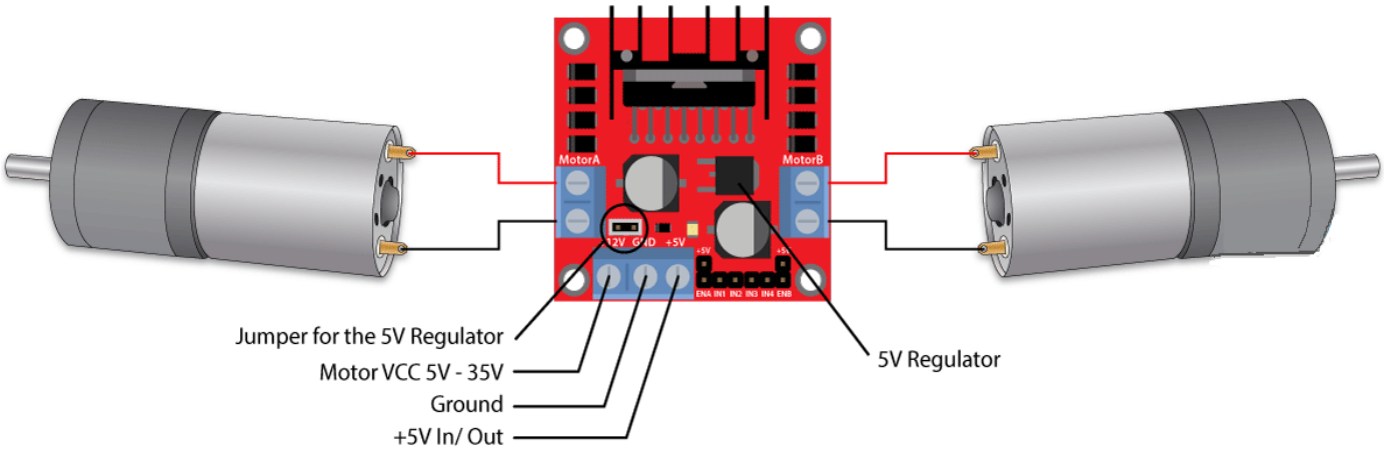


Figure 10. L298N-Pinout

Speed Controller Pins:

The Enable A and Enable B pins are used for enabling and controlling the speed of the two motors. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled.

Rotational Controller Pins:

The Input 1 and Input 2 pins are used for controlling the rotation direction of the motor A, and the inputs 3 and 4 for the motor B. Using these pins we actually control the switches of the H-Bridge inside the L298N IC. If input 1 is LOW and input 2 is HIGH the motor will move forward, and vice versa, if input 1 is HIGH and input 2 is LOW the motor will move backwards. In case both inputs are the same, i.e. either LOW or HIGH, the motor will stop. The effect on motor 1 in demonstrated in table 1. Same applies for the inputs 3 and 4 and the motor 2.

Table 1. The Effect of l298N Input Pins on Motor 1.

IN1	IN2	Motor 1
High	Low	Forward/Anticlockwise
Low	High	Forward/Clockwise
Low	Low	Stop
High	High	Stop

Table 2. The Effect of l298N Input Pins on Motor 2.

IN3	IN4	Motor 2
High	Low	Forward/Anticlockwise
Low	High	Forward/Clockwise
Low	Low	Stop
High	High	Stop

4.4.3 Microcontroller's Connections

We use two L298N drivers to control the four motors. Each side has two motors that act as a single motor. IN1 and IN2 of L298N are connected to PA1 and PA2 pins of the microcontroller which control the right side motor. Where IN3 and IN4 that control the left side motor are connected to PA3 and PA4 respectively. There is PWM signal pin on PA0 to control the speed of DC Motors.

4.4.4 Basic Movement

In table 1, we demonstrated the effects on motor 1. Same goes for motor 2. As shown in table 2, we explain how these effects will attain the actual movement of the car. We apply a break mechanism before the car stops which reverses the direction then stops the motors. This technique saves the motor drivers from burning.

Table 3. The Car Movement.

Right Motor Motion	Left Motor Motion	Car Movement
Forward/Clockwise	Forward/Clockwise	Forward
Forward/Anticlockwise	Forward/Anticlockwise	Backward
Stop	Stop	Stop

Chapter 5 - Ultrasonic Sensor

5.1 Overview

Ultrasonic sensing is one of the best ways to sense proximity and detect levels with high reliability.

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves.

An ultrasonic sensor uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity.

High-frequency sound waves reflect from boundaries to produce distinct echo patterns.

In this project we used HC-SR04 ultrasonic to detect width and depth of parking area.

5.2 Features

Here's a list of some of the HC-SR04 ultrasonic sensor features and specs—for more information, you should consult the sensor's datasheet:

- Power Supply :+5V DC
- Quiescent Current : <2mA
- Working Current: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1″ – 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS TTL pulse
- Echo Output Signal: TTL pulse proportional to the distance range
- Dimension: 45mm x 20mm x 15mm

5.3 Why use an Ultrasonic Sensor

Ultrasound is reliable in any lighting environment and can be used inside or outside. Ultrasonic sensors can handle collision avoidance for a robot, and being moved often, as long as it isn't too fast.

Ultrasonics are so widely used, they can be reliably implemented in grain bin sensing applications, water level sensing, drone applications and sensing cars at your local drive-thru restaurant or bank.

Ultrasonic rangefinders are commonly used as devices to detect a collision.

Ultrasonic Sensors are best used in the non-contact detection of:

- Presence
- Level
- Position
- Distance

Non-contact sensors are also referred to as **proximity sensors.**

Ultrasonics are Independent of:

- Light
- Smoke
- Dust
- Color
- Material (except for soft surfaces, i.e. wool, because the surface absorbs the ultrasonic sound wave and doesn't reflect sound.)

Long range detection of targets with varied surface properties.

Ultrasonic sensors are superior to infrared sensors because they aren't affected by smoke or black materials, however, soft materials which don't reflect the sonar (ultrasonic) waves very well may cause issues. It's not a perfect system, but it's good and reliable.

5.3 Working Principle

Ultrasonic sensors work by sending out a sound wave at a frequency above the range of human hearing. The transducer of the sensor acts as a microphone to receive and send the ultrasonic sound. Our ultrasonic sensors, like many others, use a single transducer to send a pulse and to receive the echo. The sensor determines the distance to a target by measuring time lapses between the sending and receiving of the ultrasonic pulse.

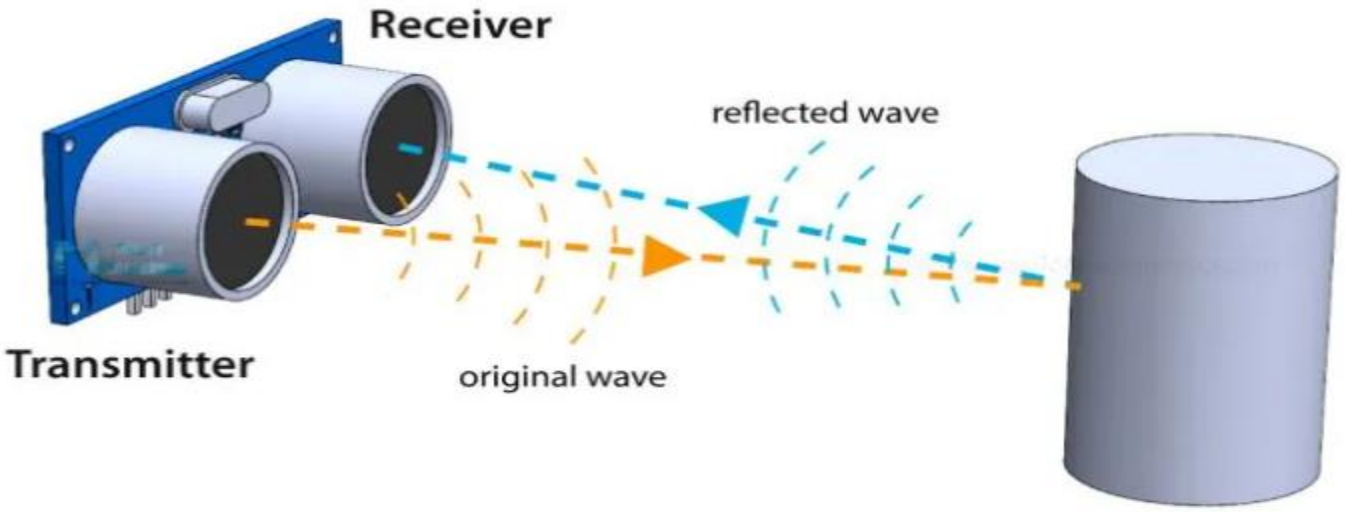


Figure 11. Ultrasonic Sensor working prinicple

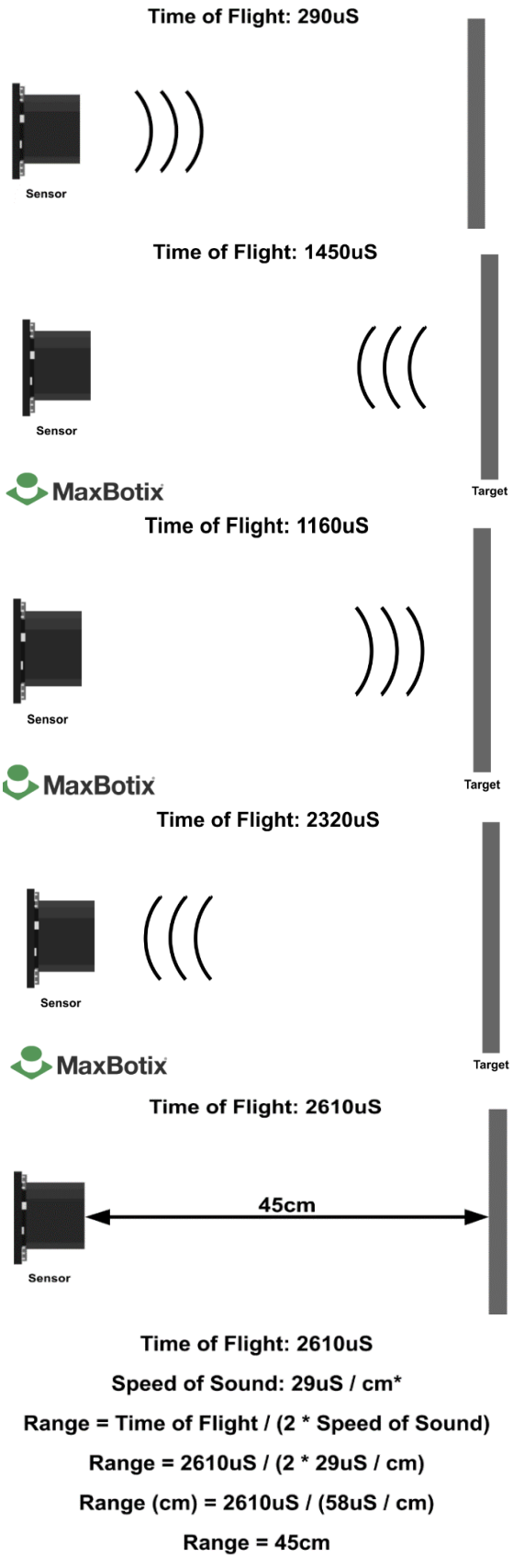


Figure 12. Ultrasonic Distance measurement

The working principle of this module is simple. It sends an ultrasonic pulse out at 40kHz which travels through the air and if there is an obstacle or object, it will bounce back to the sensor. By calculating the travel time and the speed of sound, the distance can be calculated.

Ultrasonic sensors are a great solution for the detection of clear objects. For liquid level measurement, applications that use infrared sensors, for instance, struggle with this particular use case because of target translucence.

For presence detection, ultrasonic sensors detect objects regardless of the color, surface, or material (unless the material is very soft like wool, as it would absorb sound.)

To detect transparent and other items where optical technologies may fail, ultrasonic sensors are a reliable choice.

5.4 How are Ultrasonic Sensors used?

Ultrasonic distance, level, and proximity sensors are commonly used with microcontroller platforms like Raspberry Pi, AVR, ARM, PIC, Arduino, Beagle Board, and more.

Ultrasonic sensors transmit sound waves toward a target and will determine its distance by measuring the time it took for the reflected waves to return to the receiver.

This sensor is an electronic device that will measure the distance of a target by transmitting ultrasonic sound waves, and then will convert the reflected sound into an electrical signal.

Our sensor is often used as proximity sensors.

Ultrasonic sensors are also used in obstacle avoidance systems, as well as in manufacturing.

Our Short Range sensors offers the opportunity for closer range detection where we may need a sensor that ranges objects as close to 2cm. These are also built with very low power requirements in mind, as well as environments where noise rejection is necessary.

It is possible that small grains or particles will deflect the ultrasonic pulses away from the sensor due to surface variation or angle.

In these environments, the target may absorb sound or deflect sound energy away from the sensor due to the angle of repose, surface variation, or both.

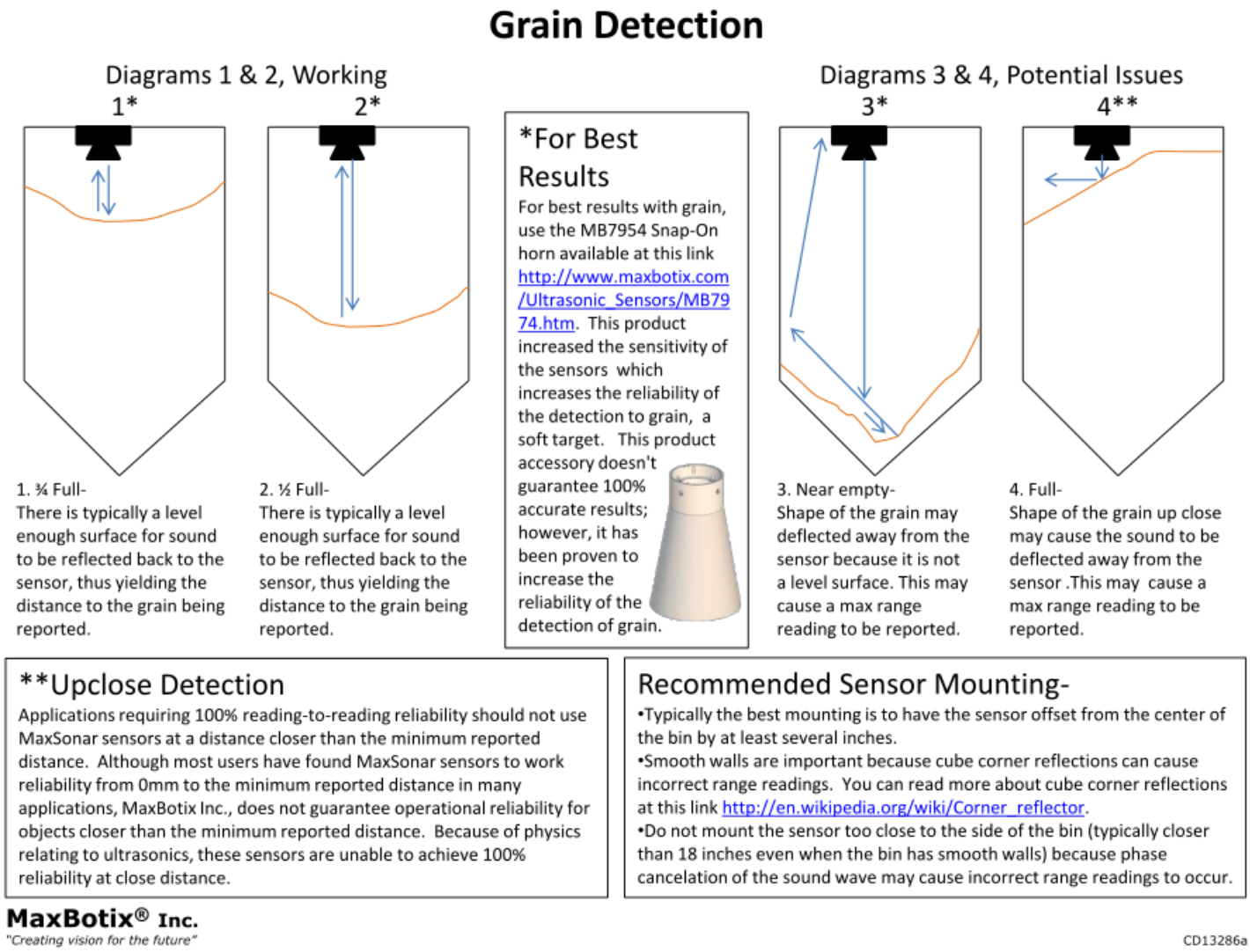


Figure 13. Gain Detection Usimg Ultrasonic Sensor

5.5 Ultrasonic pinout

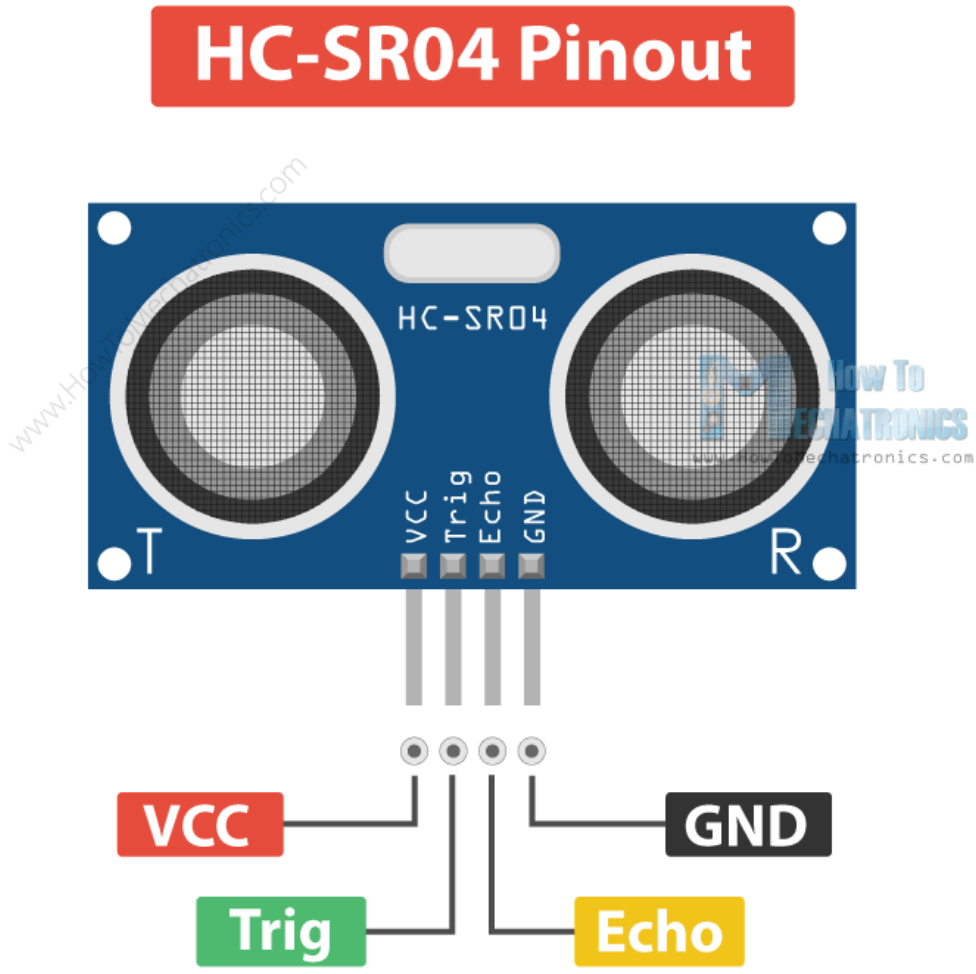


Figure 14. Ultrasonic Sensor Pinout

The sensor has 4 pins. **VCC** and **GND** go to **5V** and **GND** pins on the **STM32**, the **Triger** goes to the pin **PA5** and **Echo** goes to **PA6**. Using the **Trig** pin we send the ultrasound wave from the transmitter, and with the **Echo** pin we listen for the reflected signal.

Chapter 6 - Software

6.1 Overview

In this chapter we'll talk about the software peripherals used in this project.

This chapter is divided into these peripherals which are:

- Non-Vectored Interrupts (NVIC)
- Timer 3 - ICU mode
- Timer 2 - PWM mode

6.2 Non-Vectored Interrupts (NVIC)

In this project we used External interrupts, External interrupts are triggered by both Timer 3 and USART 1 Peripherals. To use the features of the two interrupts; we have to enable general interrupt using NVIC peripheral. But first let's a brief intro to interrupts.

- **Interrupt service routine (ISR)**

For every interrupt, there must be a program associated with it. When an interrupt occurs this program is executed to perform certain service for the interrupt. This program is commonly referred to as an *interrupt service routine* (ISR) or an *interrupt handler*. When an interrupt occurs, the CPU runs the interrupt service routine. Now, the question is how the ISR gets executed?

As shown in Figure 9-1, in the Arm CPU there are pins that are associated with hardware interrupts. They are input signals into the CPU. When the signals are triggered, CPU pushes the PC register onto the stack and loads the PC register with the address of the interrupt service routine. This causes the ISR to get executed.

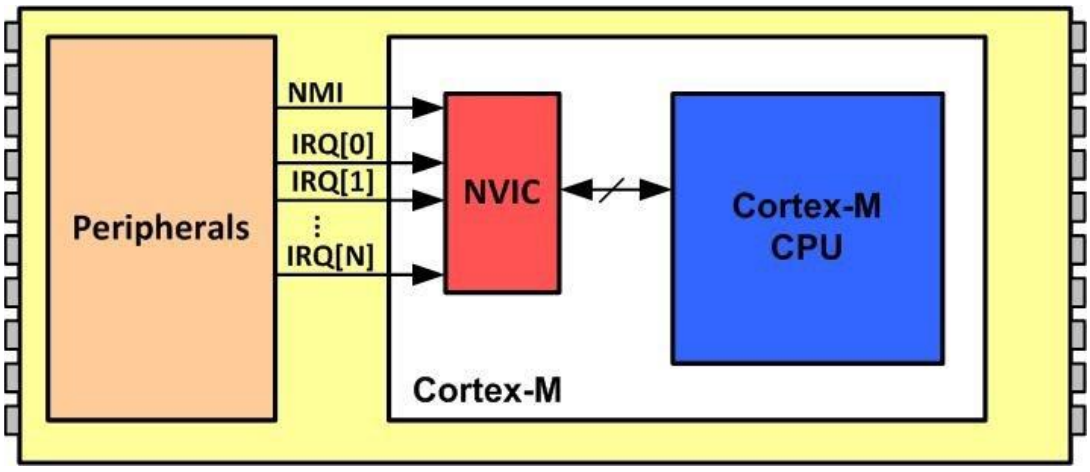


Figure 9-1: NVIC in Arm Cortex-M

- **Interrupt Vector Table**

Since there is a program (ISR) associated with every interrupt and this program resides in memory (RAM or ROM), there must be a look-up table to hold the addresses of these ISRs. This look-up table is called *interrupt vector table*. In the Arm Cortex-M4 by default, the lowest 1024 bytes (256 × 4 = 1024) of memory space are set aside for the interrupt vector table and must not be used for any other function. Table 9-1 provides a list of interrupts and their designated functions as defined by Arm Cortex-M4 products. Of the 47 interrupts, some are used for software interrupts and some are for hardware IRQ interrupts. We used Timer 3 and USART 1 Interrupts.

Position	Priority	Type of priority	Acronym	Description	Address
15	22	settable	DMA1_Stream4	DMA1 Stream4 global interrupt	0x0000 007C
16	23	settable	DMA1_Stream5	DMA1 Stream5 global interrupt	0x0000 0080
17	24	settable	DMA1_Stream6	DMA1 Stream6 global interrupt	0x0000 0084
18	25	settable	ADC	ADC1 global interrupts	0x0000 0088
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C
24	31	settable	TIM1_BRK_TIM9	TIM1 Break interrupt and TIM9 global interrupt	0x0000 00A0
25	32	settable	TIM1_UP_TIM10	TIM1 Update interrupt and TIM10 global interrupt	0x0000 00A4
26	33	settable	TIM1_TRG_COM_TIM11	TIM1 Trigger and Commutation interrupts and TIM11 global interrupt	0x0000 00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000 00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000 00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000 00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000 00B8
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000 00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000 00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000 00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000 00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000 00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000 00D0
37	44	settable	USART1	USART1 global interrupt	0x0000 00D4
38	45	settable	USART2	USART2 global interrupt	0x0000 00D8
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000 00E0
41	48	settable	EXTI17 / RTC_Alarm	EXTI Line 17 interrupt / RTC Alarms (A and B) through EXTI line interrupt	0x0000 00E4
42	49	settable	EXTI18 / OTG_FS_WKUP	EXTI Line 18 interrupt / USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000 00E8
47	54	settable	DMA1_Stream7	DMA1 Stream7 global interrupt	0x0000 00FC
49	56	settable	SDIO	SDIO global interrupt	0x0000 0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000 0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000 010C

Table 9-1: Interrupt Vector table for STM32F401xB/CSTM32F401xD/E

- **NVIC (nested vector interrupt controller) In Arm Cortex-M**

The Cortex-M has an on-chip interrupt controller called NVIC (Nested Vector Interrupt Controller). See Figure 6-2. This allows some degree of standardization among the Arm Cortex-M (M0, M0+, M1, M3, M4, and M7) family members. The classical Arm chips and Cortex-A and Cortex-R series do not have this NVIC interrupt controller, therefore Arm manufacturers implementation of the interrupt handling varies. This chapter focuses on the interrupts for Arm Cortex-M series. It must be noted that there are substantial differences between the Arm Cortex-M series and classical Arm versions as far as interrupt handling are concerned. The study of classical Arm and Arm Cortex A and R series interrupts are left to the reader since they are used for high-performance systems using complex OS and real-time system.

- **IRQ Peripheral interrupts**

An ISR can be launched as a result of an event at the peripheral devices such as timer timeout or analog-to-digital converter (ADC) conversion complete. The largest number of the interrupts in the Arm Cortex-M belongs to this category. Notice from Table 6-1 that Arm Cortex-M NVIC has set aside the first 15 interrupts (INT1 to INT15) for internal use and is not available to the chip designer. The Reset, NMI, Hard Fault, and so on are part of this group of exceptions. The rest of the interrupts are used for peripherals. INT16 to INT47 (IRQ0 to IRQ31) are used by the chip manufacturer to be assigned to various peripherals such as timers, ADC, Serial COM, external hardware interrupts, and so on. There is no standard in assigning the INT16 to INT47 to the peripherals. Different manufacturers assign different interrupts to different peripherals. Not all the interrupts are assigned. You need to examine the datasheet for your Arm Cortex-M chip. Each peripheral module has a group of special function registers that must be used to configure the functions of the peripheral module. For a given peripheral interrupt to take effect, the interrupt for that peripheral must be enabled. The special function registers for that device provide the way to enable the interrupts.

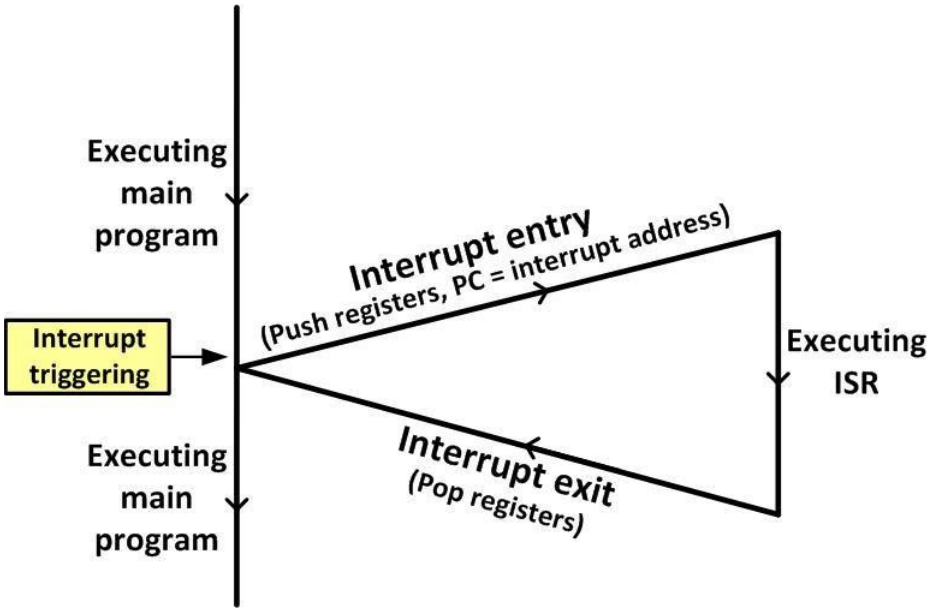


Figure 9-2: Main Program gets interrupted

6.3 Timer 3 – ICU mode

6.3.1 Overview

In addition to SysTick timers, STM32Fxx Arm comes with a large number of timers. The timers are called TIMx where x=0, 1, 2, 3, 4, and so on. They fall into three categories: a) Advanced Control Timer, b) General Purpose Timer, and c) Basic Timer. Advanced Control Timer is mostly used for PWM and DC motor control. The TIMx are used for a wide range of generic timer/counter applications. While most timers in STM Arm are 16-bit, there are one or two 32-bit timers too. The 32-bit timers are in the general purpose timers group and usually designated as TIM2 and TIM5. The 32-bit Timers discussed first in this section. Other aspects of timers will be discussed in later part of this section.

TIMx has a counter similar to SysTick timer with additional features including:

• To slow down the speed of timer counting, it has a prescaler.

• It can be used to generate an output of programmable frequency and pulse width modulation.

• It can capture the frequency and pulse width of an input signal.

• It can be used as event counter.

• On a counter overflow, compare match, or capture, it can generate

interrupts or events.

In Input capture mode, the Capture/Compare Registers (TIMx_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCXIF flag (TIMx_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx_SR register) is set. CCxIF can be cleared by software by writing it to 0 or by reading the captured data stored in the TIMx_CCRx register. CCxOF is cleared when written to 0.

When an input capture occurs:

- The TIMx_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

Now we will talk about the used registers to use ICU mode in Timer 3

9.3.2 TIM3 control register 1 (TIM3_CR1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 9-3: TIM3_CR1 pin configuration

- **Bit 0 – CEN: Counter Enable**

The (CEN) bit is set to 1 to enable counter of Timer 3 and make the timer start, if set to 0 then the Timer 3's counter is disabled and timer stopped.

6.3.3 TIM3 DMA/Interrupt enable register (TIM3_DIER)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Figure 9-4: TIM3_DIER pin configuration

- **Bit 1 – CC1E: Capture/Compare Interrupt Enable**

The CC1E is set to 1 to enable timer interrupt triggering on capture compare on channel 1 since Timer 3 has four channels, if set to 0 the Timer interrupt triggering is disabled.

6.3.4 TIM3 capture/compare mode register 1 (TIM3_CCMR1)

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. Take care that the same bit can have a different meaning for the input stage and for the output stage.

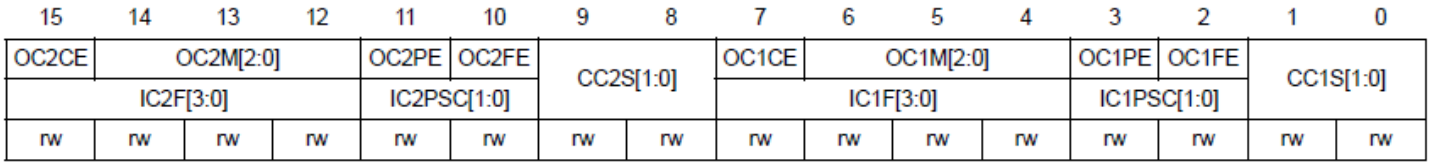


Figure 9-5: TIM3_CCMR1 pin configuration

In Input Capture Mode:

- **Bits 1:0 CC1S: Capture/Compare 1 selection**

By setting these two bits to 0 1 respectively, channel 1 in Timer 3 is configured as input, IC1 is mapped on TI1.

6.3.5 TIM3 capture/compare enable register (TIM3_CCER)

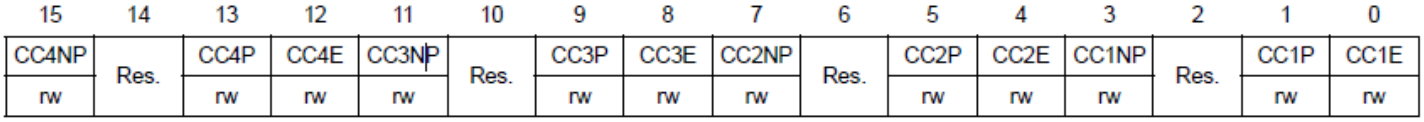


Figure 9-6: TIM3_CCER pin configuration

- **Bit 0 CC1E: Capture/Compare 1 output enable**

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not. If set to 1 the Capture is enabled else if set to 0 then Capture is disabled.

- **Bits 1:3 CC1P/CC1NP: Capture/Compare 1 output Polarity**

CC1NP/CC1P bits select TI1FP1 and TI2FP1 polarity for trigger or capture operations. When both bits are set to 1 this means that capture happens at both edges (Rising/Falling edge).

6.3.6 TIM3 counter (TIM3_CNT)

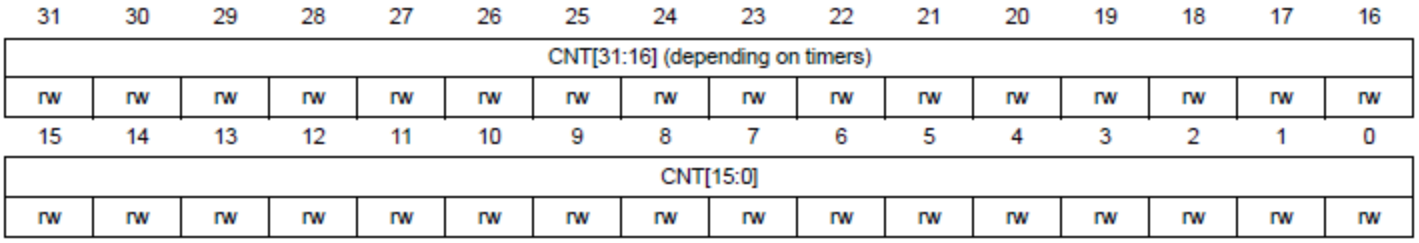


Figure 9-7: TIM3_CNT register

TIM3_CNT register is used to set the initial value that the timer/counter starts counting from, also used to clear the counter.

In our system it's used to clear the counter after capturing the falling edge.

6.3.7 TIM3 Prescaler (TIM3_PSC)

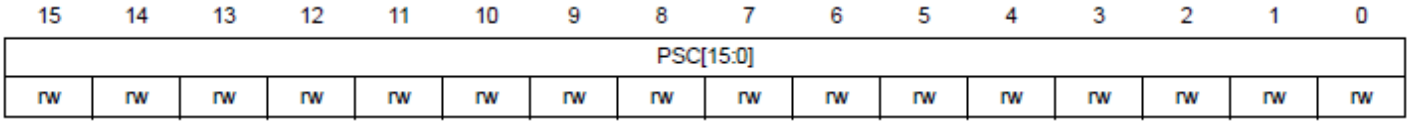


Figure 9-8: TIM3_PSC register

- **Bits 15:0 PSC [15:0]: Prescaler value**

The counter clock frequency CK_CNT is equal to FCK_PSC / (PSC[15:0] + 1). PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIM3_EGR register or through trigger controller when configured in “reset mode”).

6.3.8 TIM3 Auto-Reload Register (TIM3_ARR)

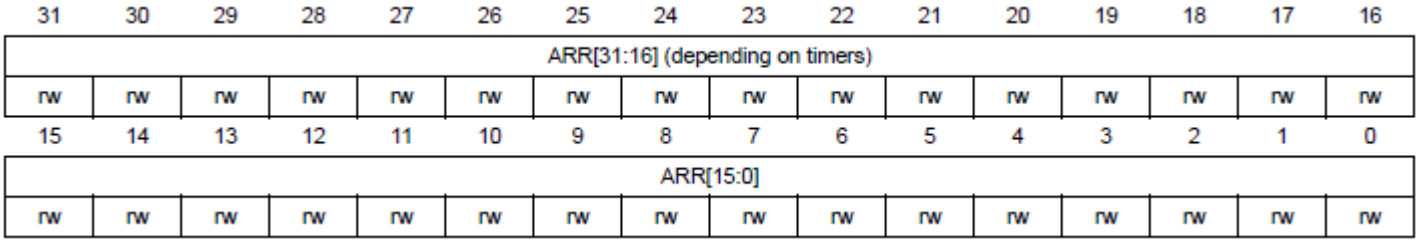


Figure 9-9: TIM3_ARR register

- **Bits 15:0 ARR [15:0]: Auto-reload value**

ARR is the value to be loaded in the actual auto-reload register. In our system ARR value is maximum 0xFFFF (65535).

6.3.9 TIM3 capture/compare register 1 (TIM3_CCR1)

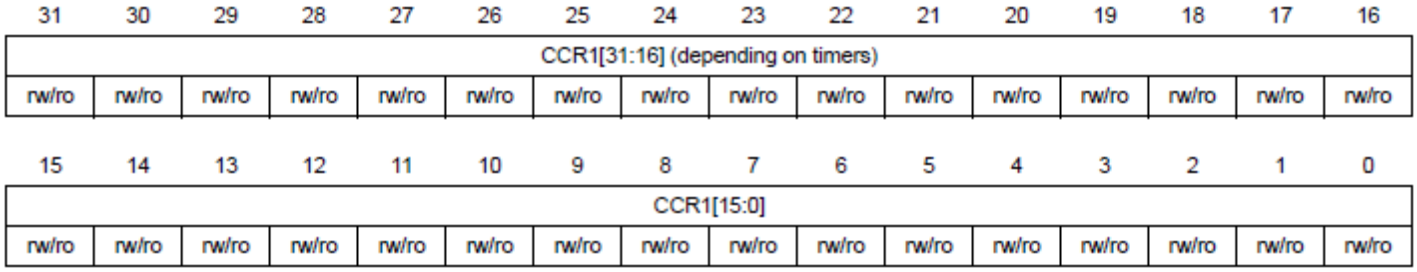


Figure 9-10: TIM3_CCR1 register

If channel CC1 is configured as input:

CCR1 is the counter value transferred by the last input capture 1 event (IC1). The

TIM3_CCR1 register is read-only and cannot be programmed.

6.4 Timer 2

6.4.1 Overview

Timer 2 is one of the general-purpose timer. The general-purpose timers consist of a 16-bit or 32-bit auto-reload counter driven by a programmable prescaler. They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare and PWM). Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

6.4.2 Timer 2 Main Features

General-purpose TIMx timer features include:

- 16-bit (TIM3 and TIM4) or 32-bit (TIM2 and TIM5) up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65536.
- Up to 4 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge- and Center-aligned modes)
  - One-pulse mode output
- Synchronization circuit to control timer with external signals and to interconnect several timers together
- Interrupt/DMA generation based on several events

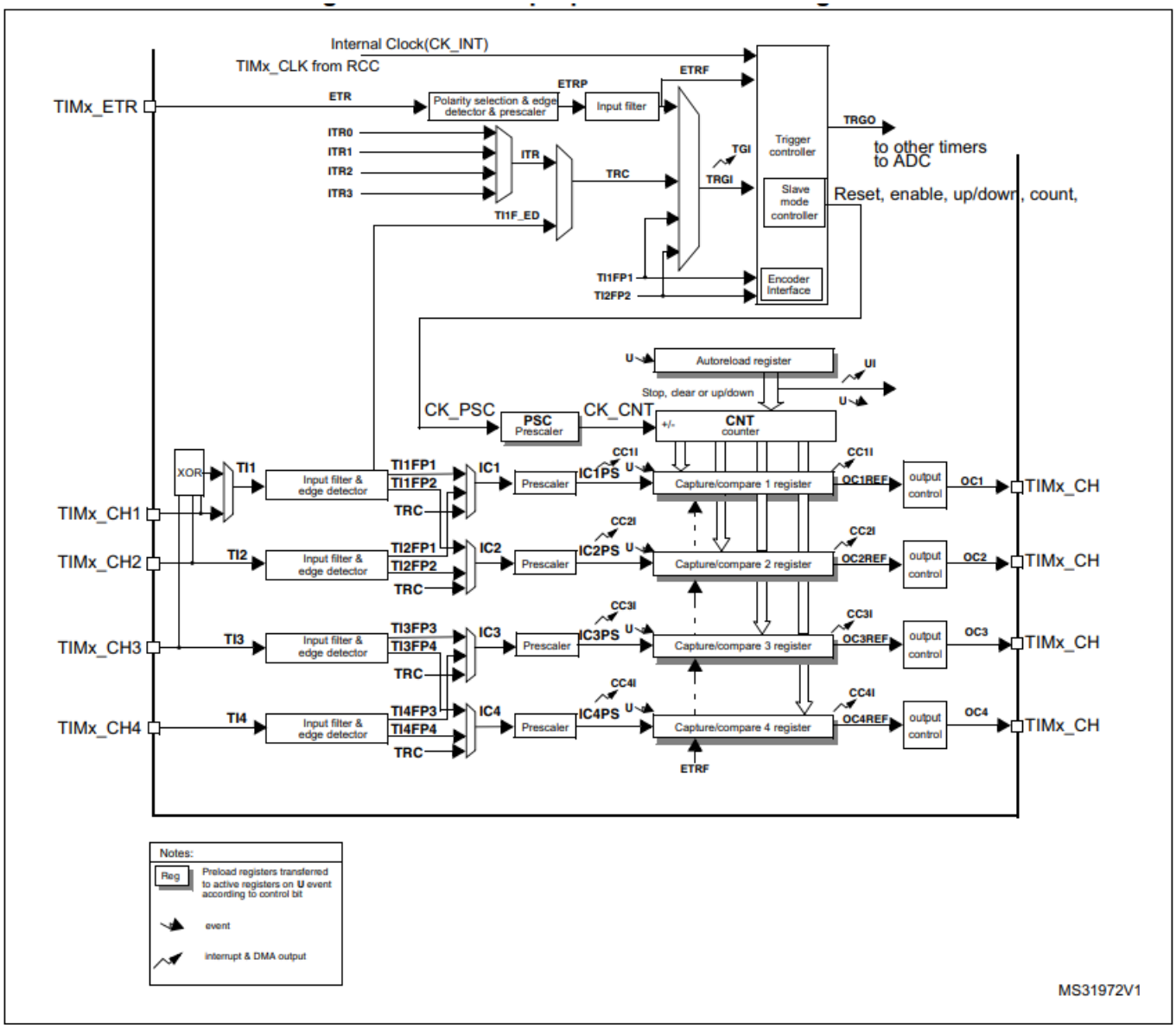


Figure 15: General-purpose Timers Block Diagram

6.4.3 Time-Base Unit

The main block of the programmable timer is a 16-bit/32-bit counter with its related auto reload register. The counter can count up. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC)
- Auto-Reload Register (TIMx_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling)

Note that the actual counter enable signal CNT_EN is set 1 clock cycle after CEN.

6.4.4 Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit/32-bit register (in the TIMx_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

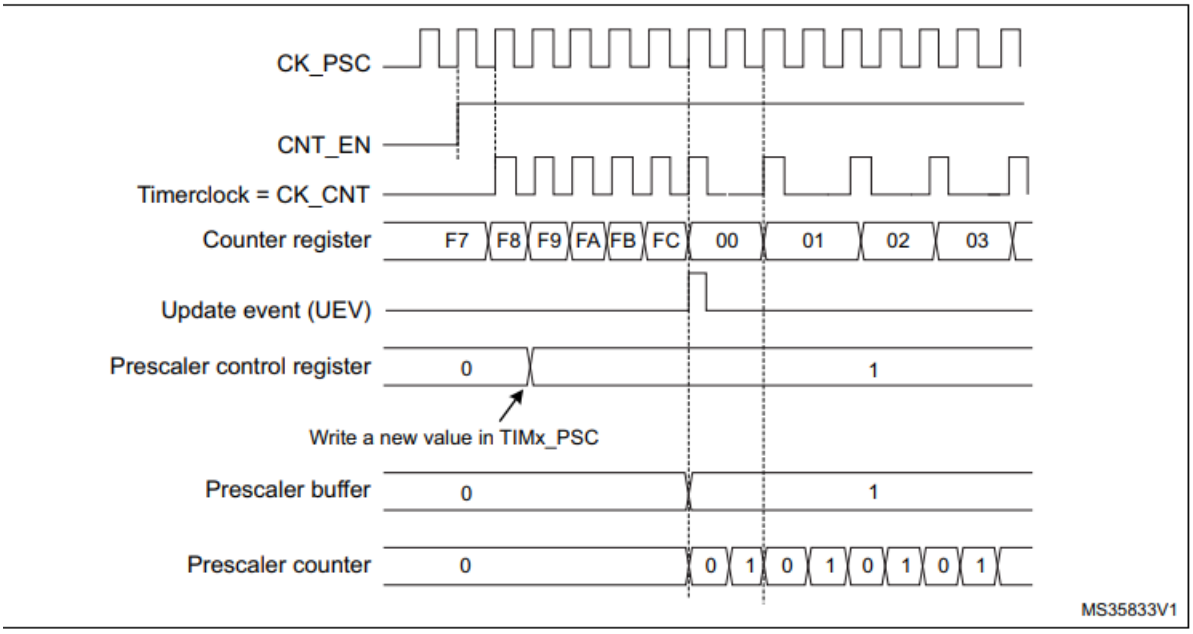


Figure 16: Prescaler Example

6.4.5 Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

An Update event can be generated at each counter overflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller).

The UEV event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36

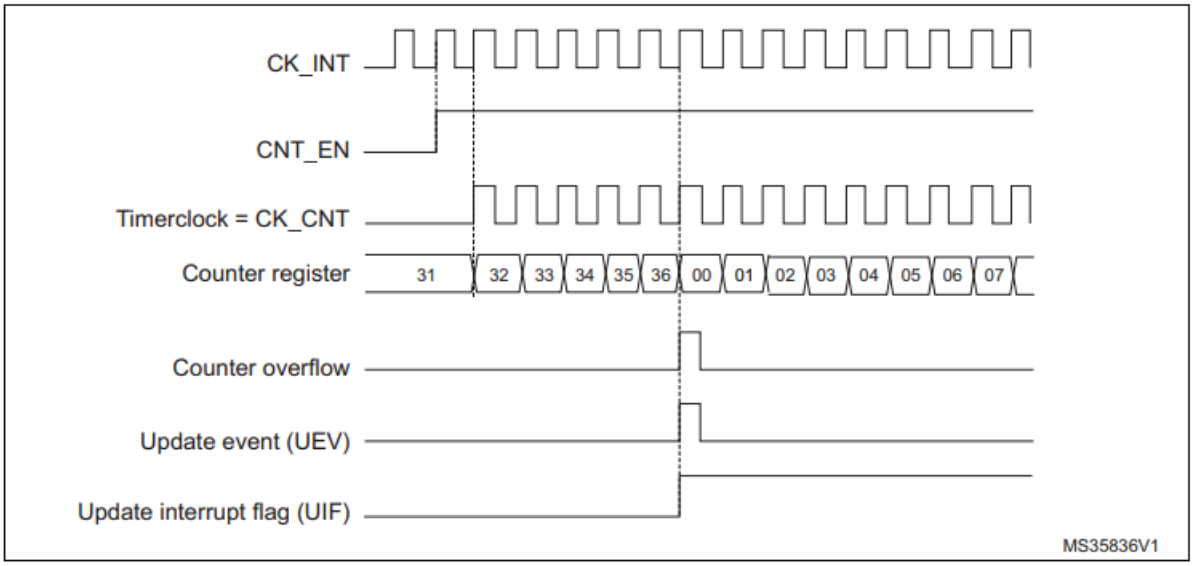


Figure 17: Counter timing diagram, internal clock divided by 1

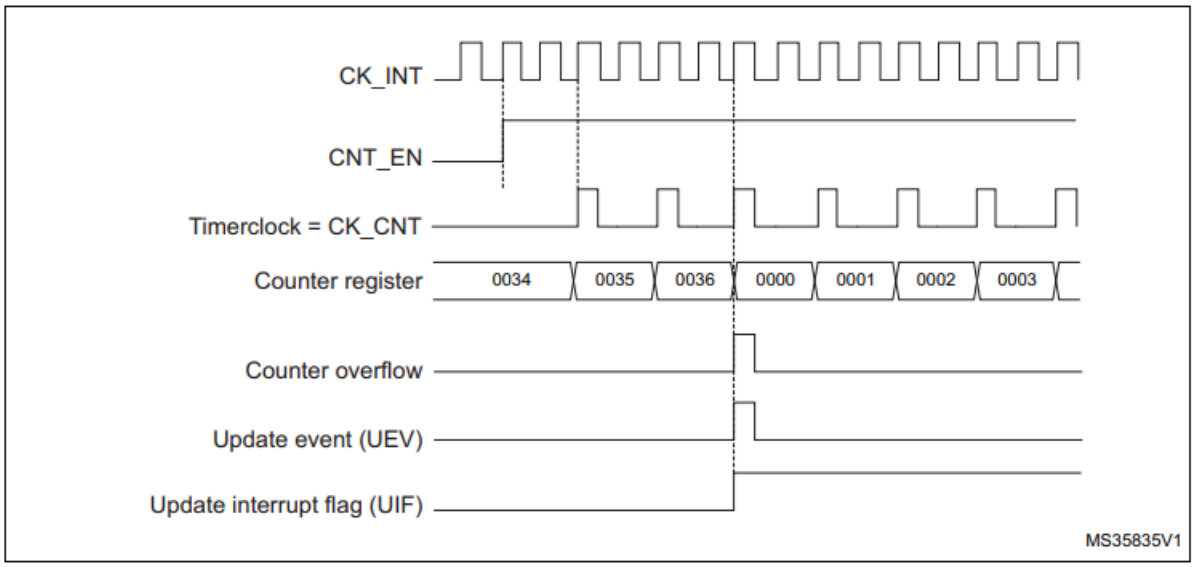


Figure 18: . Counter timing diagram, internal clock divided by 2

6.4.6 Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

An Update event can be generate at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller)

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx_SR register) is set (depending on the URS bit):

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

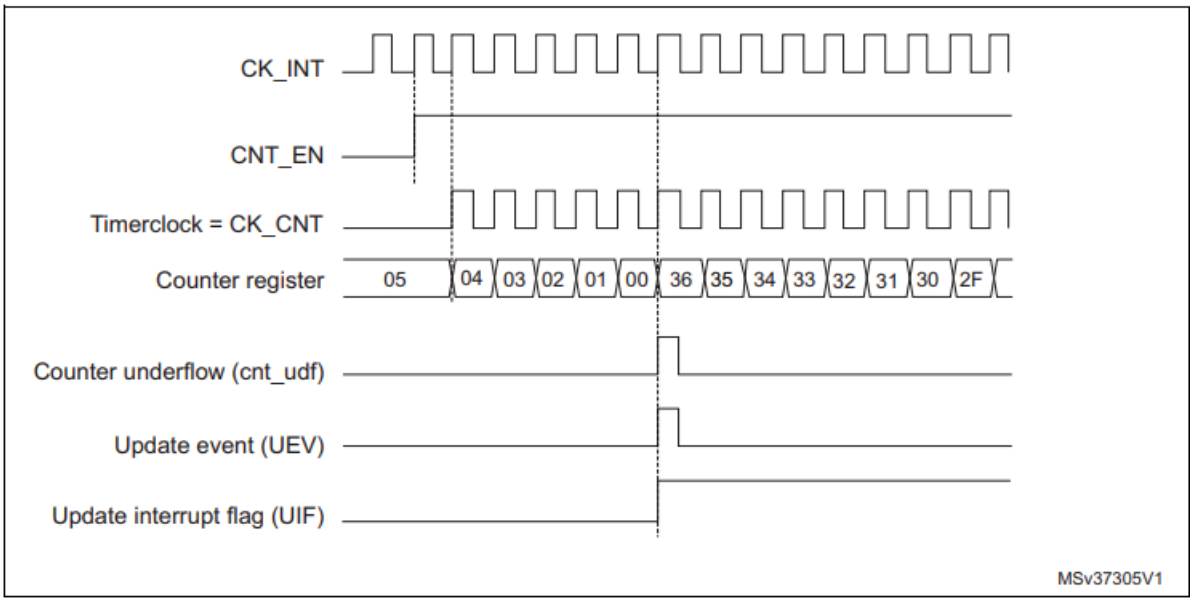


Figure 19: Counter timing diagram, internal clock divided by 1

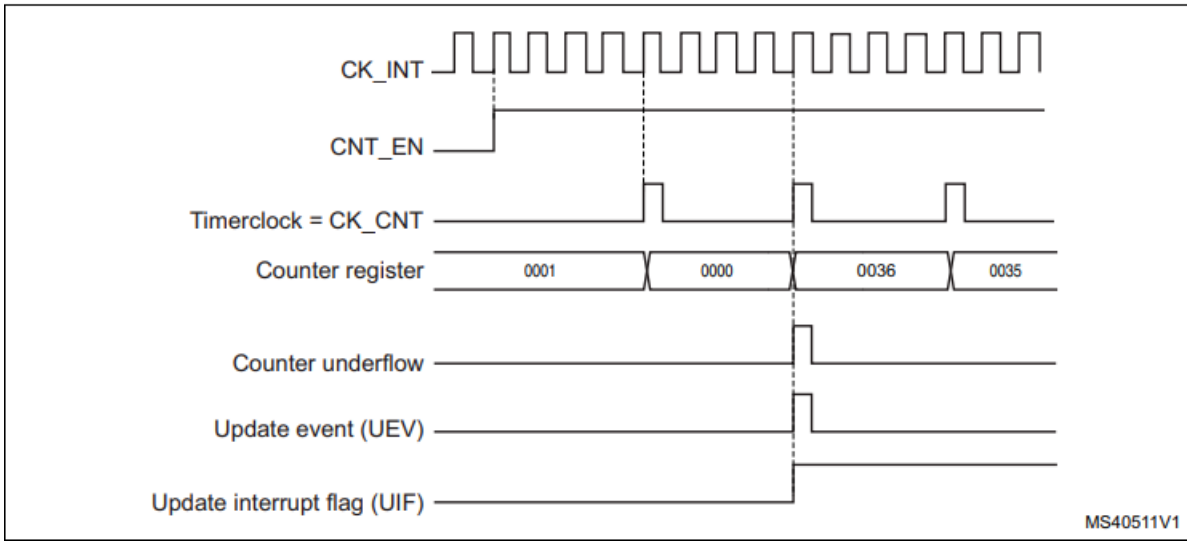


Figure 20: Counter timing diagram, internal clock divided by 4

6.5 Pulse Width Modulation

6.5.1 PWM Signal

Digital signals are signals that can be represented by a 0 or 1. Analog signals on the other hand have a greater range of possible values than just a 0 or 1. Both of these signals are used in the electronics around us but they are handled very differently. If we need to take an analog input, we can get the real-time analog data from a sensor, and then using an analog-to-digital converter (ADC), convert it to digital data for a microcontroller. But what if we need to control an analog device from our microcontroller? Some microcontrollers have an onboard digital-to-analog converter (DAC) to output a true analog signal in order to control analog devices and we can even use an external DAC. But a DAC is relatively expensive to produce in terms of cost and it also takes up a lot of silicon area. To overcome these issues and to easily achieve the functionality of a DAC in a much more cost-efficient way, we can use the technique of PWM.

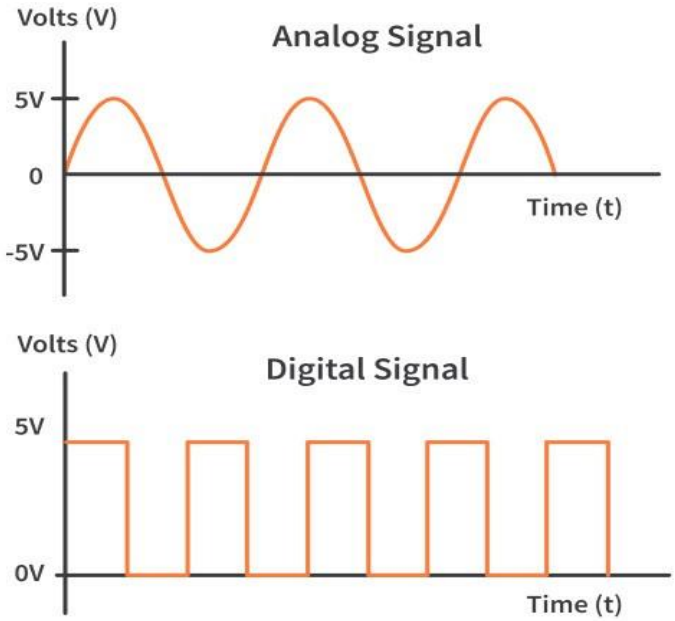


Figure 21: Analog vs Digital Signal

PWM or Pulse Width Modulation is a technique used to control analog devices, using a digital signal. This technique can be used to output an analog-like signal from a digital device, like a microcontroller. We can control motors, lights, actuators, and more using the generated PWM signal. An important thing to note here is that PWM is not a true analog signal. The digital signal is modified in a way to fake an analog signal.

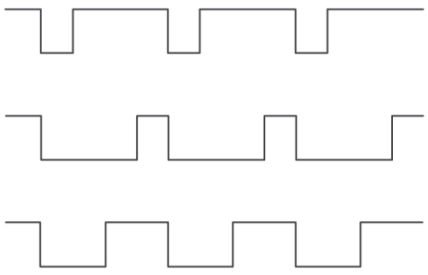


Figure 22: PWM Signal Waveforms

With electrical components, you can control the current delivered to them, even if the only control you have is switching the power supply on and off. You can rapidly switch the power supply on and off in a pattern to control the current delivered to the device. Keeping it “on” for a duration more than the “off” duration will raise the average power level and doing it the opposite way will reduce the average power level. We can generate a varied range of speeds (analog) for our motors and actuators even though in reality we are just switching between the two possible states of being on and off (digital). This is exactly what a PWM signal is.

Let’s use this analogy as an example. Imagine you have a ceiling fan in your home but without any speed regulator. So you can either turn it on and it’ll gradually achieve max speed, or you can turn it off. Now, what if I ask you to run the fan at 50% of its max speed. Is it possible without a speed regulator? Give this question a thought. The answer is yes, it is possible. While we don’t recommend actually doing this, you can achieve it by playing a little with the switch of the ceiling fan. We know that the fan will not instantly achieve the max speed the moment we turn it on and neither will it immediately come to a halt as we turn it off. Turn on the switch, wait until you see that the fan has achieved 50% of the speed, and turn off the switch. Switch it on again as it starts to slow down. Using this delay to our advantage, we can switch it on or off, making it faster or slower, to get the speed we want. An important thing to note here is that continuous switching of the fan will make it draw a large amount of current, so again, we discourage you from doing this in real life.

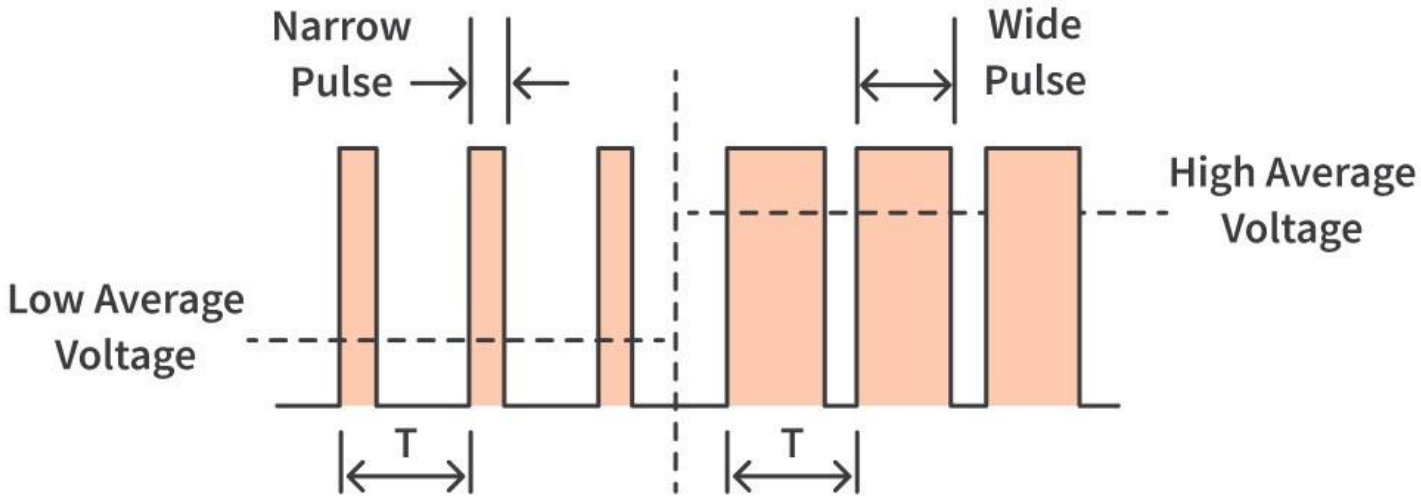


Figure 23: Pulse and time period of a PWM signal

The switching on and off is the pulse. The duration for which the pulse is held at a high state is the pulse width. T represents the total time taken to complete a cycle. Modulation refers to the modification of the original signal to get our desired signal. So we are modifying a signal or pulse such that it is “on” for the duration as per our need, hence, Pulse-Width-Modulation.

6.5.2 Duty Cycle

In signals, we define logic high as “on-time”. To represent the duration of “on-time”, we use the concept of duty cycle. In simple terms, the duty cycle describes the percentage of time a digital signal is “on” over an interval or period. It is represented in percentage (%).

Let’s take a signal with a maximum voltage of 10V. If our signal takes one second to complete a cycle and the signal is on for 0.5 seconds and off for the other 0.5 seconds, this is called a 50% duty cycle and we will get 5V as the average voltage output. If the signal is on for 0.75 seconds and off for the other 0.25 seconds, it will be a 75% duty cycle and the output will be 7.5V. The below image represents signals having different duty cycles and the average voltage generated by them:

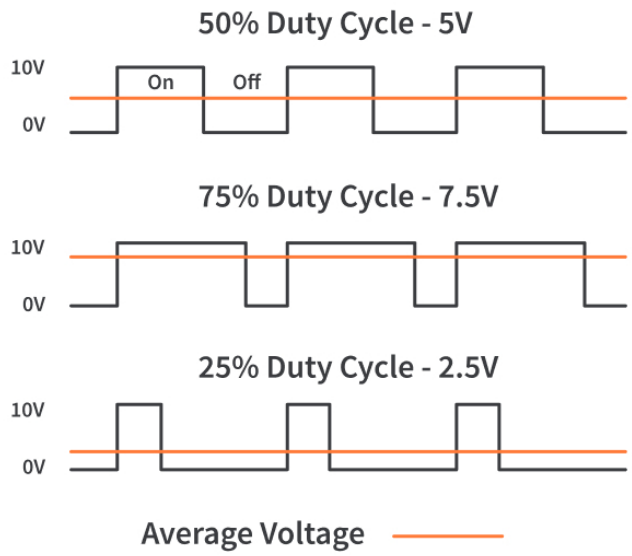


Figure 24: Different duty cycles and their average voltage

Duty cycles make PWM what they actually are. We get different PWM signals by varying duty cycles of the signal. The duty cycle can be calculated as per the below formula:

$$D = \frac{T_{on}}{Period} * 100$$

where:

D = Duty Cycle in Percentage

Ton = Duration of the signal being in the “on” state

Period = Total time taken to complete one cycle (Ton + Toff)

After we’ve calculated the duty cycle, we can calculate the average voltage of the signal using the following formula:

$$V_{avg} = \frac{D}{100} * V_{max}$$

Where: Vavg = Average voltage of the signal

D = Duty Cycle in Percentage

Vmax = Max voltage of the signal

6.5.3 Timer 2 - PWM mode

Pulse width modulation mode allows generating a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register. The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or ‘111 (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. The user must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, the user has to initialize all the registers by setting the UG bit in the TIMx_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. OCx output is enabled by the CCxE bit in the TIMx_CCER register. Refer to the TIMx_CCERx register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx≤TIMx_CNT or TIMx_CNT≤TIMx_CCRx (depending on the direction of the counter). However, to comply with the ETRF (OCREF can be cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison changes.
- When the output compare mode (OCxM bits in TIMx_CCMRx register) switches from the “frozen” configuration (no comparison, OCxM=‘000) to one of the PWM modes (OCxM=‘110 or ‘111).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

6.5.4 Timer 2 – PWM Configuration

The OCxM bits can be set as mode 1 (110) or mode 2 (111) for PWM. The difference between these two options is the way the output starts as active or inactive state. For the PWM, the frequency is these two options is the way the output starts as active or inactive state. For the PWM, the frequency is bit mode or 32-bit timers) register.

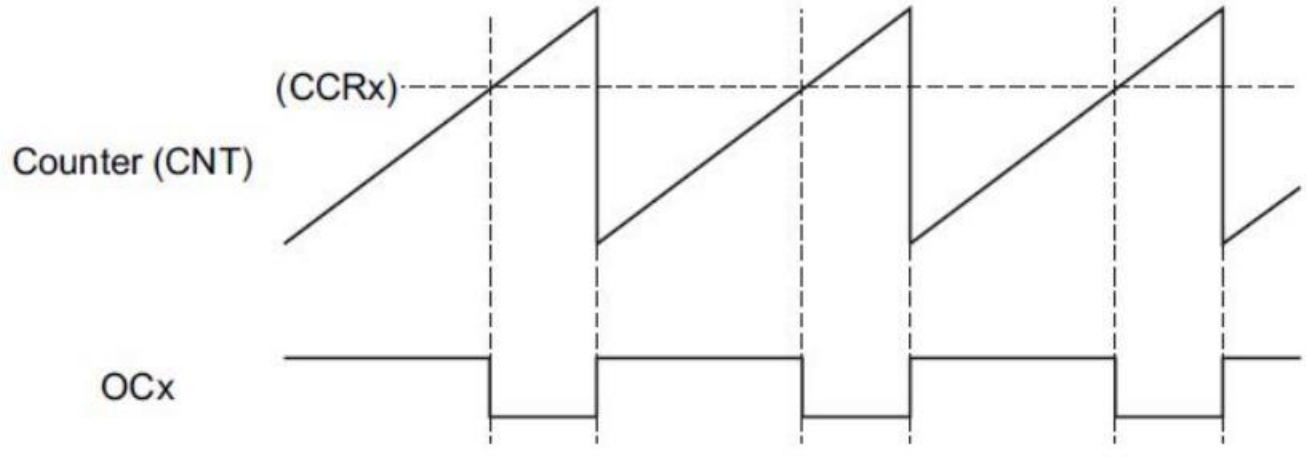


Figure 25: PWM Mode

1- Enable Timer 2 Clock from RCC Peripheral
2- Set the Prescaler at Timer 2 Prescaler Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 26: TIMx_PSC

Bits 15:0 PSC[15:0]: Prescaler value The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1). PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

3- Set the Auto Reload Preload Value at Timer 2 ARR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 27: TIMx_ARR

Bits 15:0 ARR[15:0]: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

4- Set The Capture Compare Mode Register to PWM Mode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 28: TIMx_CCMR1

Bits 6:4 OC1M: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1)

011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.

100: Force inactive level - OC1REF is forced low

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF=‘0) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=1).

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

5- Enable PWM Channel 1 Through Capture Compare Enable Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw

Figure 29: TIMx_CCER

Bit 0 CC1E: Capture/Compare 1 output enable

1: On - OC1 signal is output on the corresponding output pin.

6- Set the Value of the Capture Compare Register 1 to determine the duty cycle

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR1[31:16] (depending on timers)															
rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro	rw/ro

Figure 30: CCR1

7- Enable Timer 2 though the Control Register 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 0 CEN: Counter enable

0: Counter disabled

1: Counter enabled

Chapter 7 - Communication

7.1 General Overview

An embedded system often requires a means of communication with the external world for a number of possible reasons. It could be to transfer data to another device, sending and receiving commands, or simply for debugging purposes. One of the most common interfaces used in embedded systems is the universal asynchronous receiver/transmitter (UART). We use the UART protocol in our project for several reasons, debugging, communicating with each of ESP8266, GPS module, GSM module, and Bluetooth module.

7.2 Duplex Communication System

7.2.1 Simplex

In which one device transmits and the others can only “listen”, ex. Radio.

7.2.2 Half-Duplex

In which both parties can communicate with each other, but not simultaneously, the communication is one direction at a time, ex. Walkie-talkie.

7.2.3 Full-Duplex

In which both parties can communicate with each other simultaneously, ex. Telephone.

Figure 56 below gives a clearer understanding of the 3 communication system types

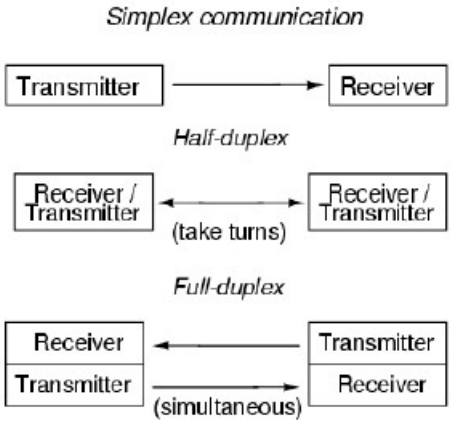


Figure 31. 3 System Communication types

7.3 UART

The Universal Asynchronous Receiver/Transmitter (UART) is a two wire hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end [9]. It offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format [10].

The UART performs serial-to-parallel conversions on data received from a peripheral device and parallel-to-serial conversion on data received from the CPU. The CPU can read the UART status at any time. The UART includes control capability and a processor interrupt system that can be tailored to minimize software management of the communications link [10].

7.3.1 UART Parameters

UART peripherals typically have several configurable parameters required to support different standards. There are five parameters that must be configured correctly to establish a basic serial connection

- **Baud Rate :**

Baud rate is the number of symbols or modulations per second. It indicates how many times the lines can change state (high or low) per second. Since each symbol represents one bit, the bit rate equals the baud rate. We use a baud rate of 9600 to connect ATmega32 and Raspberry pi 4.

Since the UART interface does not use a clock signal to synchronize the transmitter and receiver devices, it transmits data asynchronously. Instead of a clock signal, the transmitter generates a bit stream based on its clock signal while the receiver uses its internal clock signal to sample the incoming data. The point of synchronization is achieved by having the same baud rate on both devices. Failure to do so affects the timing of sending and receiving data that can cause discrepancies during data handling. The allowable difference of baud rate is up to 10% before the timing of bits gets too far off.

- **Start Bit**

The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start data transfer, the transmitting UART pulls the transmission line from high-to-low for one clock cycle. When the receiving UART detects the high-to-low voltage transition, it begins reading the bits in the data frame at the baud rate frequency. Location of the start bit in the packet is shown in figure 57.

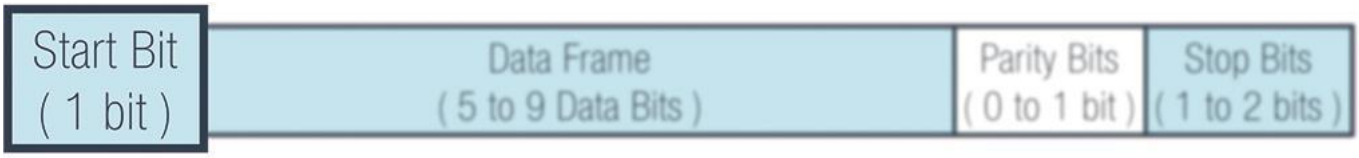


Figure 32. Start Bit

- **Data Frame**

The data frame shown in figure 18 contains the actual data being transferred. It can be five (5) bits up to eight (8) bits long if a parity bit is used. If no parity bit is used, the data frame can be nine (9) bits long. In most cases, the data is sent with the least significant bit first.

Figure 33. Data Frame

- **Parity**

Parity describes the evenness or oddness of a number. The parity bit demonstrated in figure 19, is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers.

After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number.

When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd, or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

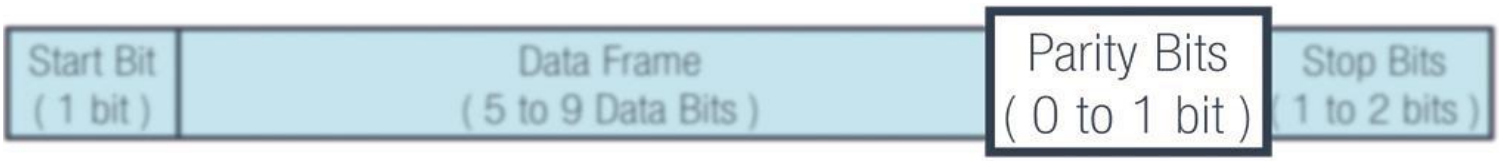


Figure 34. Parity bits

- **Stop Bits**

To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for one (1) to two (2) bit(s) duration. Location of the stop bit in the data packet is shown in figure 60

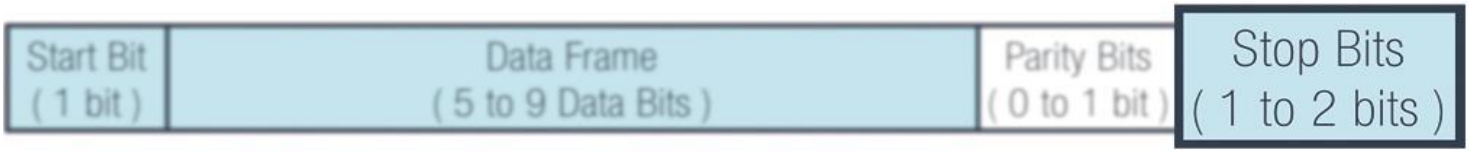


Figure 35. Stop Bits

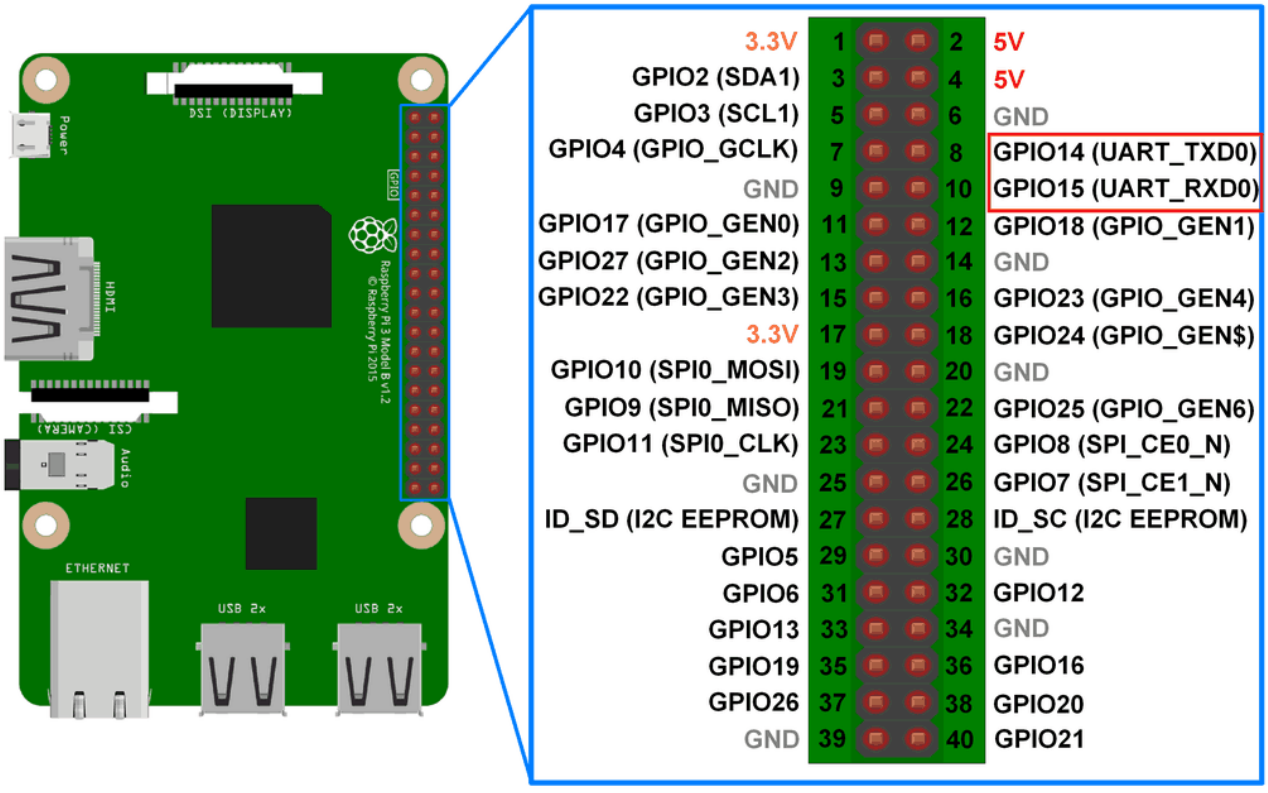
7.3.2 UART through Raspberry Pi 3

UART is multi master communication protocol.in this project this protocol is quite easy to use and very convenient for communicating between several boards: Raspberry Pi 3 to STm32f401.

For using UART you need 3 pins:

- GND that you'll connect to the global GND of your circuit.
- RX for Reception. You'll connect this pin to the TX pin of the other component.
- TX for Transmission. You'll connect this pin to the RX of the other component.

If the component you're communicating with is not already powered, you'll also have to use a power pin (3.3V or 5V) to power on that component.



7.3.2 UART through STm32f401

7.4 USART1

USART1 is a vendor peripheral, asynchronous communications.

The main features are:

- Full duplex
- Configurable oversampling method by 16 or by 8 to give flexibility between speed and clock tolerance.
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits

7.4.1 Registers

• A status register (USART_SR)
• Data Register (USART_DR)
• A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
• A Guardtime Register (USART_GTPR) in case of Smartcard mode.

Chapter 8 - Testing

8.1 Overview

In this chapter, we show the testing technique we used to test our project in order to ensure that it satisfies the target requirements. Briefly, testing every sub-system individually to ensure that it is working well under all conditions and performs its desired function. After that, sub-systems are combined and tested together until such time that the entire system can enter final testing to achieve requirements.

8.2 Embedded Testing

Embedded testing is the process of verification and validation of both software and hardware. It ensures the defect free whole system including software and hardware. It is basically performed on hardware in order to find the defects. It also ensures that system meets the end user's requirements.

8.3 Why we test?

In general, embedded developers test for four main reasons:

- To find bugs in software:
- To reduce risk to both users and the company
- To reduce development and maintenance costs: The earlier a bug is found, the less expensive it is to fix. Refer to figure 61.

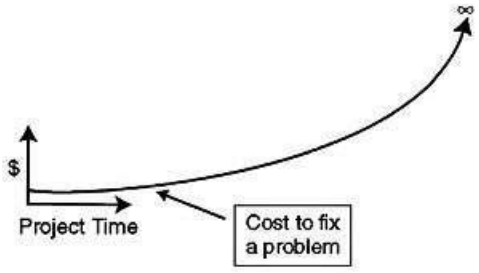


Figure 36. The cost to fix a problem

- **To improve performance:** Testing maximizes the performance of the system. Finding and eliminating dead code and inefficient code can help ensure that the software uses the full potential of the hardware and thus avoids the dreaded "hardware re-spin.

8.4 Testing in Project

To test the successfulness of this project, we followed the following four testing levels as in figure 62 below:

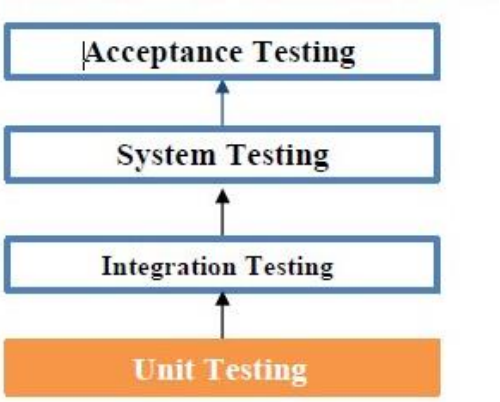


Figure 37. Testing Levels

8.4.1 Unit/Module Testing

It is the initial level of software testing our project where units/components are tested individually. The goal of this step is to ensure that each unit of the software code acts as we expected. It is done by isolating a section of code and verifying its correctness. Unit tests help to fix bugs early in the development cycle and save costs. Table 20 shows the test cases scenario used in this level.

Table 4. Test Cases Scenario

Test Case Scenario	**Status (Pass/Fail)**
DC Motors	
Increase Car Speed Manually From GUI	Pass
Decrease Car Speed Manually From GUI	Pass
Stop Motors Manually From GUI	Pass
Changing Motors Speed by Changing Duty Cycle	Pass
UARTS Driver	
Transmitting/Receiving a String (Polling//Interrupt)	Pass
Ultrasonic sensor	
Measure distance in CM	Pass
Raspberry Pi	
Sending a String to STM32	Pass
Receiving Distance & Speed from STM32	Pass

8.4.2 Integration Testing

It comes after the unit testing, where individual units are combined and tested as a group to expose faults in the interaction between integrated units. Table 21 shows the test cases we used for Integration Testing.

Table 5. Test Cases for Integration Testing

Cruise Control System	
Turning Normal Cruise Control ON From GUI	Pass
Turning Normal Cruise Control OFF From GUI	Pass
Stops The Ability to Increase Speed at NCC Mode	Pass
Stop the Car Automatically When an Object Enter the Danger Region at NCC Mode	Pass
Turning Adaptive Cruise Control ON From GUI	Pass
Turning Adaptive Cruise Control OFF From GUI	Pass
Selecting Ranges at ACC Mode From GUI	Pass
Increase / Decrease Car Speed According to the Distance Between it And the Object Ahead at All Available Ranges at ACC Mode	Pass
Stop the Car Automatically When an Object Enter the Danger Region at ACC Mode	Pass
Changing the Selected Range From the GUI	Pass
Changing CCMode at Runtime	Pass

8.4.3 System Testing

This is the last step of testing, where the whole system is tested. Our test case scenario is to test that when the car starts ignition, The ultrasonic sensor starts measuring distance while the car moves forward upon the signal the comes from the raspberry pi.

The car will move according to the character received from raspberry to STM32

A: Increase Car Speed

D: Decrease Car Speed

S: Stop Motors

O: Turn OFF Cruise Control

P: Turn ON Adaptive Cruise Control

N: Turn ON Normal Cruise Control

F: Set the Range to FAR

M: Set the Range to MEDUIM

C: Set the Range to CLOSE

8.4.4 Acceptance Testing

It is an end-user testing to determine whether or not the software system has met the requirement specifications. Since the project discussion committee is our client/end-user, we leave it up to them to evaluate our work.

Raspberry Pi Appendices

Appendix A

```python
import cv2
import numpy as np
import utlis

curveList = []
avgVal=10

def getLaneCurve(img,display=2):

    imgCopy = img.copy()
    imgResult = img.copy()
    #### STEP 1
    imgThres = utlis.thresholding(img)

    #### STEP 2
    hT, wT, c = img.shape
    points = utlis.valTrackbars()
    imgWarp = utlis.warpImg(imgThres,points,wT,hT)
    imgWarpPoints = utlis.drawPoints(imgCopy,points)

    #### STEP 3
    middlePoint,imgHist = utlis.getHistogram(imgWarp,display=True,minPer=0.5,region=4)
    curveAveragePoint, imgHist = utlis.getHistogram(imgWarp, display=True, minPer=0.9)
    curveRaw = curveAveragePoint - middlePoint

    #### SETP 4
    curveList.append(curveRaw)
    if len(curveList)>avgVal:
        curveList.pop(0)
    curve = int(sum(curveList)/len(curveList))

    #### STEP 5
    if display != 0:
        imgInvWarp = utlis.warpImg(imgWarp, points, wT, hT, inv=True)
        imgInvWarp = cv2.cvtColor(imgInvWarp, cv2.COLOR_GRAY2BGR)
        imgInvWarp[0:hT // 3, 0:wT] = 0, 0, 0
        imgLaneColor = np.zeros_like(img)
        imgLaneColor[:] = 0, 255, 0
        imgLaneColor = cv2.bitwise_and(imgInvWarp, imgLaneColor)
        imgResult = cv2.addWeighted(imgResult, 1, imgLaneColor, 1, 0)
        midY = 450
        cv2.putText(imgResult, str(curve), (wT // 2 - 80, 85), cv2.FONT_HERSHEY_COMPLEX, 2, (255, 0, 255), 3)
        cv2.line(imgResult, (wT // 2, midY), (wT // 2 + (curve * 3), midY), (255, 0, 255), 5)
        cv2.line(imgResult, ((wT // 2 + (curve * 3)), midY - 25), (wT // 2 + (curve * 3), midY + 25), (0, 255, 0), 5)
        for x in range(-30, 30):
            w = wT // 20
            cv2.line(imgResult, (w * x + int(curve // 50), midY - 10),
                     (w * x + int(curve // 50), midY + 10), (0, 0, 255), 2)
        #fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);
        #cv2.putText(imgResult, 'FPS ' + str(int(fps)), (20, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (230, 50, 50), 3);
```

```
    if display == 2:
        imgStacked = utlis.stackImages(0.7, ([img, imgWarpPoints, imgWarp],
                                             [imgHist, imgLaneColor, imgResult]))
        cv2.imshow('ImageStack', imgStacked)
    elif display == 1:
        cv2.imshow('Resutlt', imgResult)

    #### NORMALIZATION
    curve = curve/100
    if curve>1: curve ==1
    if curve<-1:curve == -1

    return curve


if __name__ == '__main__':
    cap = cv2.VideoCapture('vid1.mp4')
    intialTrackBarVals = [102, 80, 20, 214 ]
    utlis.initializeTrackbars(intialTrackBarVals)
    frameCounter = 0
    while True:
        frameCounter += 1
        if cap.get(cv2.CAP_PROP_FRAME_COUNT) == frameCounter:
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            frameCounter = 0

        success, img = cap.read()
        img = cv2.resize(img,(480,240))
        curve = getLaneCurve(img,display=2)
        print(curve)
        #cv2.imshow('Vid',img)
        cv2.waitKey(1)
```

Appendix B

Utilities

Appendix C

Raspberry Pi UART code

ARM Appendices

Scan QR code for Drive link

Conclusion

In this paper, a mid-cost vehicle tracking the lanes and auto-parking is presented. This project is an automotive efficient merge between Embedded Systems and Computer Vision with the help of ATmega32 & Raspberry Pi 4 the vehicle can self-drive on the road safely and smoothly, it can also stop immediately if an obstacle detected in front of it by the IR sensor, besides that it turns left or right with the road's direction and lastly it detects the parking area detected using the Ultrasonic Sensor and measures if it's suitable to park, once it's suitable; it starts self-parking, once it's parked successfully; it can self-unpark when we push the button and gets back on the road again.

References

1. https://ieeexplore.ieee.org/abstract/document/8574003
2. https://www.autobytel.com/car-buying-guides/features/2021-jeep-grand-cherokee-l-vs-2021-ford-explorer-134348/
3. https://reports.weforum.org/digital-transformation/the-driverless-car-revolution/
4. https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm
5. https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/
6. https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
7. https://circuitdigest.com/article/servo-motor-working-and-basics
8. https://robu.in/ir-sensor-working/#:~:text=IR%20sensor%20is%20an%20electronic,some%20form%20of%20thermal%20radiation
9. https://www.elprocus.com/infrared-ir-sensor-circuit-and-working/
10. https://www.watelectronics.com/ir-sensor/
11. https://www.microchip.com/wwwproducts/productds/ATmega32
12. https://www.javatpoint.com/atmega32-avr-microcontroller
13. https://www.javatpoint.com/what-is-raspberry-pi
14. https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf