**SOLAR POWERED AUTONOMOUS MULTIPURPOSE AGRICULTURAL ROBOT USING ESP32**

A

MAJOR PROJECT REPORT

**Submitted in the partial Fulfillment of the requirements for the award of the**

**Degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**Submitted By**

1. **G. Sireesha 17R21A04D8**
2. **B. Ruchitha Reddy 17R21A04G5**
3. **S. Nikitha 17R21A04H3**
4. **P. Shivani 18R25A0431**

**UNDER THE GUIDANCE OF**

**Dr. B. Sridhar**

**Professor**

**** MLR Institute of Technology

(Autonomous)

**(Affiliated to JNTUH, Hyderabad)**

**Dundigal, Hyderabad-500043**

**2017-2021**

**** MLR Institute of Technology

(Autonomous)

**(Affiliated to JNTUH, Hyderabad)**

**Dundigal, Hyderabad-500043**



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

***CERTIFICATE***

This is to certify that the project *entitled* ***“Solar powered autonomous multipurpose agricultural robot using”*** *is the bonafied work done by, G.Sireesha, B.Ruchitha, S.Nikitha and P.Shivani bearing roll numbers 17R21A04D8,17R21A04G5,17R21A04H3 and 18R25A0431 respectively* in partial fulfillment of the requirement for the award of the degree of B.Tech in Electronics and Communication Engineering, during the academic year 2020-2021.

**Internal Guide Head of the Department**

**External Examiner**

We express our profound thanks to the management of **MLR Institute of Technology**, Dundigal, Hyderabad, for completing this project.

We take immense pleasure in expressing my sincere thanks to **Dr.K.Srinivasa Rao**, Principal, MLR Institute of Technology, for his kind support and encouragement.

We are very much grateful to **Dr S.V.S Prasad**, Professor & Head of the Department, MLR Institute of Technology, for permitting me to do the project and encouraged me with his valuable suggestions.

We are very much grateful to **Dr. B. Sridhar, Professor** for his unflinching cooperation through the project.

We would like to express our sincere thanks to the teaching and non teaching faculty members of ECE Dept., MLR Institute of Technology, who extended their help to us in making our project work successful.

Project associates:

G.Sireesha (17R21A04D8)

B.Ruchitha (17R21A04G5)

S.Nikitha (17R21A04H3)

P.Shivani (18R25A0431)

**ABSTRACT**

Agriculture has always been an important occupation in India. Almost 70% of the population depends on agriculture as their occupation. The present methods of seed sowing, pesticide spraying and grass cutting are expensive and inconvenient to handle. So the agricultural system in India should be encouraged by developing a system which will reduce the man power and time.

Main aim of our project is to reduce man power by developing a robot which does the activities like seeding, ploughing and watering. In addition to this we are using soil sensor to check the moisture in soil. Solar panels are used for power backup. We are developing an android application using embedded C and connect to ESP32 by which we can give the commands to the robot.

It is mainly used in agriculture sectors and local farming. The benefits of robot are reduced human

intervention and efficient resources utilization. Instructions are passed to the system which ensures no direct contact with human and thus safety of operator is ensured. The robot is solar powered hence it is renewable energy source. By using this advanced work, farmer can save more time and also reduce lot of labour cost.

**TABLE OF CONTENTS**

**ABSTRACT viii to ---**

**LIST OF FIGURES --- to ---**

**LIST OF TABLES (if any) --- to ---**

**LIST OF ABBREVIATIONS --- to ---**

**NOTATION --- to ---**

**CHAPTER 1 Introduction Page No.**

1.1 Introduction 1

1.2 Project Overview

**CHAPTER 2 BACKGROUND INFORMATION Page No.**

2.1 Embedded Systems 15

2.1.1 History 15

2.1.2 Tools 16

2.1.4 Real time issues 16

2.2 Need For Embedded Systems 17

2.2.1 Debugging 18

2.2.2 Reliability 19

2.3 Explanation of embedded systems 20

2.3.1 Software Architecture 21

2.3.2 Stand Alone embedded system 22

2.3.3 Real-time embedded system 23

2.3.4 Network communication embedded system 24

2.3.5 Different types of processing units 25

2.4 Applications of embedded systems 26

2.4.1 Consumer Applications 27

2.4.2 Office automation 28

2.4.3 Industrial automation 29

2.4.4 Computer Networking 30

2.4.5 Telecommunications 31

**CHAPTER 3 Project Description Page No.**

3.1 Block diagram 19

3.2 Advantages and Disadvantages

Hardware components 19

3.3 Applications

**CHAPTER 4 Hardware and Software Description Page No.**

4.1 Hardware Description

4.1.1 Introduction

4.2 Solar Panel 20

4.3 Rechargeable battery 21

4.4 ESP 32 development board 21

4.4.1 Features

4.4.2 QFN packed chip and module

4.5 Soil Moisture Sensor 22

4.5.1 Working Principle of Moisture Sensor

4.5.2 Moisture Sensor Module Features & Specifications

4.6 L293D motor driver

4.6.1 L293D Pin description

4.6.2 Features

4.6.3 Where to use L293D IC

4.6.4 How to use a L293D Motor Driver IC

4.6.5 Applications

4.7 DC Motor

4.8 Software description

4.8.1 Programming

4.8.2 How to install ESP32 board in Arduino IDE

4.8.3 Testing the Installation

4.8.4 Trouble Shooting

4.9 Programming in embedded C

4.9.1 What is C language?

4.9.2 What is an Embedded C Programming?

4.9.3 Embedded System Programming

4.9.4 Steps to build an Embedded C programming

4.9.4.1 Comments

4.9.4.2 Directives of Processor

4.9.4.3 Global Variables

4.9.4.4 Core Function / Main Function

4.9.4.5 Declaration of Variable

4.9.4.6 The logic of the Program

4.9.5 Main Factors of Embedded C Program

**CHAPTER 5 Results and Discussions**

5.1 Result

**CHAPTER 6**

**6.1 Conclusion**

**6.2 Future Scope**

**REFERENCES:**

**LIST OF FIGURES**

**S.NO FIG.NO: DESCRIPTION PAGE NO:**

1 2.1 A modern example of embedded system 16

2 2.2 Network communication embedded system 23

3 2.3 Automatic coffee makes equipment 24

4 2.4 Fax machine 24

5 2.5 Printing machine 24

6 2.6 Vehicle 25

7 2.7 Computer networking 25

8 2.8 Cell Phone 26

9 2.9 Web Camera 26

10 4.1 Solar Panels 31

11 4.2 Rechargeable Battery 32

12 4.3 Charging of a secondary cell or battery 33

13 4.4 ESP-WROOM-32 module with ESP32-

D0WDQ6 Chip 34

14 4.5 ESP32 function block diagram 35

15 4.6 Moisture Sensor 43

16 4.7 Motor Driver IC L293D 46

17 4.8 Using of L293D Motor Driver 48

18 4.9 DC Motor 50

19 5.1 Image of Robot 69

**LIST OF TABLES**

**S. No. TABLE No. DESCRIPTION PAGE NO:**

1 4.4.1 QFN Packaged chip and module 36

2 4.4.2 Printed circuit boards 37

3 4.4.3 Development and other boards 40

4 4.4.4 Moisture Sensor Module Pin out

Configuration 44

5 4.6.1 L293D PIN Description 47

CHAPTER 1: INTRODUCTION

1.1 Introduction:

We all knew that Agriculture is the basic source for human beings and 70% of our country depends on this to survive. We must understand the value of farming and the way it’s done. Many methods have been exposed in the development of Agriculture and today our project helps in a technical method by using a robot. This robot is fast and the operation is easy, it also reduces the time and labour work. Human involvement is very less and the process will be done in a clear way.

In existing system, it doesn’t use soil moisture sensor and the robot doesn’t work efficiently. So, we want to design a robot which works efficiently without any man power.

Basically, this robot works on embedded systems, that is controlled by microcontrollers or processors in which we dump a program for its operation. Today lot of industries are using this system for process control. This controls many devices nowadays and being so advance and common use. To save costs, embedded systems are frequently used and it has cheap microcontrollers.

1.2 Project overview:

This is the internet of things ( IOT ) based project, where we particularly uses micro controller ( ESP32 ), Solar panels, DC motors, Soil moisture sensor, Motor drivers(L293D) , Rechargeable battery, Pump motors, Ultrasonic sensor with robot chassis to built these robotic car setup. Solar panels take sunlight as source to produce electric power using photovoltaic effect. This produced electric power is stored in the rechargeable battery. This rechargeable battery is connected to ESP32. Soil moisture sensor senses the moisture in the soil and gives the signal to the ESP32. Based upon the signals given by the sensor it gives commands to Motor drivers. Motor drivers are used to give required voltage to the DC Motors as micro controller gives only 5volts which is not sufficient to the DC Motors. In this project we used 4 DC Motors in which two of them are used for the motion of robot and other two DC Motors are used for ploughing and seeding. As Robot works in the field there may be the chances of getting obstacles on its way. So, we used Ultrasonic sensor to sense obstacles and gives the signal to ESP32, which gives commands to DC motor drivers to stop.

Mobile application is used to command the robot from anywhere, where the mobile is connected to the wifi of ESP32

CHAPTER 2: EMBEDDED SYSTEMS

2.1 Embedded Systems:

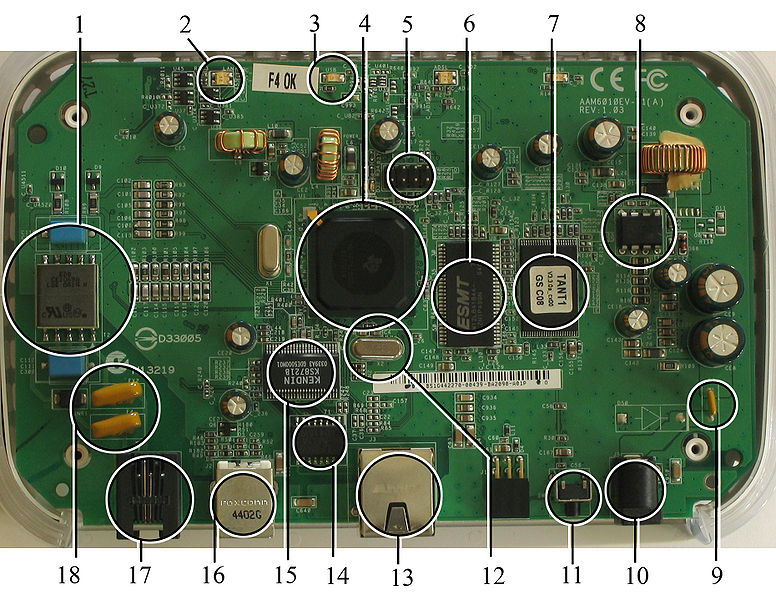
An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Embedded systems are controlled by one or more main processing cores that are typically either microcontrollers or digital signal processors (DSP). The key characteristic, however, is being dedicated to handle a particular task, which may require very powerful processors. For example, air traffic control systems may usefully be viewed as embedded, even though they involve mainframe computers and dedicated regional and national networks between airports and radar sites. (Each radar probably includes one or more embedded systems of its own.)

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not a strictly definable term, as most systems have some element of extensibility or programmability. For example, handheld computers share some elements with embedded systems such as the operating systems and microprocessors which power them, but they allow different applications to be loaded and peripherals to be connected. Moreover, even systems which don't expose programmability as a primary feature generally need to support software updates. On a continuum from "general purpose" to "embedded", large application systems will have subcomponents at most points even if the system as a whole is "designed to perform one or a few dedicated functions", and is thus appropriate to call "embedded". A modern example of embedded system is shown in fig: 2.1.



**Fig 2.1:A modern example of embedded system**

Labelled parts include microprocessor (4), RAM (6), flash memory (7).Embedded systems programming is not like normal PC programming. In many ways, programming for an embedded system is like programming PC 15 years ago. The hardware for the system is usually chosen to make the device as cheap as possible. Spending an extra dollar a unit in order to make things easier to program can cost millions. Hiring a programmer for an extra month is cheap in comparison. This means the programmer must make do with slow processors and low memory, while at the same time battling a need for efficiency not seen in most PC applications. Below is a list of issues specific to the embedded field.

2.1.1 History:

In the earliest years of computers in the 1930–40s, computers were sometimes dedicated to a single task, but were far too large and expensive for most kinds of tasks performed by embedded computers of today. Over time however, the concept of controllers evolved from traditional [electromechanical](http://en.wikipedia.org/wiki/Electromechanical) sequencers, via solid state devices, to the use of computer technology.

One of the first recognizably modern embedded systems was the [Apollo Guidance Computer](http://en.wikipedia.org/wiki/Apollo_Guidance_Computer), developed by [Charles Stark Draper](http://en.wikipedia.org/wiki/Charles_Stark_Draper) at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project as it employed the then newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the [Minuteman missile](http://en.wikipedia.org/wiki/Minuteman_(missile)), released in 1961. It was built from [transistor](http://en.wikipedia.org/wiki/Transistor) [logic](http://en.wikipedia.org/wiki/Digital_circuit) and had a [hard disk](http://en.wikipedia.org/wiki/Hard_disk) for main memory. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits.

2.1.2 Tools:

Embedded development makes up a small fraction of total programming. There's also a large number of embedded architectures, unlike the PC world where 1 instruction set rules, and the UNIX world where there's only 3 or 4 major ones. This means that the tools are more expensive. It also means that they're lowering featured, and less developed. On a major embedded project, at some point you will almost always find a compiler bug of some sort.

Debugging tools are another issue. Since you can't always run general programs on your embedded processor, you can't always run a debugger on it. This makes fixing your program difficult. Special hardware such as JTAG ports can overcome this issue in part. However, if you stop on a breakpoint when your system is controlling real world hardware (such as a motor), permanent equipment damage can occur. As a result, people doing embedded programming quickly become masters at using serial IO channels and error message style debugging.

2.1.3 Resources:

To save costs, embedded systems frequently have the cheapest processors that can do the job. This means your programs need to be written as efficiently as possible. When dealing with large data sets, issues like memory cache misses that never matter in PC programming can hurt you. Luckily, this won't happen too often- use reasonably efficient algorithms to start, and optimize only when necessary. Of course, normal profilers won't work well, due to the same reason debuggers don't work well.

Memory is also an issue. For the same cost savings reasons, embedded systems usually have the least memory they can get away with. That means their algorithms must be memory efficient (unlike in PC programs, you will frequently sacrifice processor time for memory, rather than the reverse). It also means you can't afford to leak memory. Embedded applications generally use deterministic memory techniques and avoid the default "new" and "malloc" functions, so that leaks can be found and eliminated more easily. Other resources programmers expect may not even exist. For example, most embedded processors do not have hardware FPUs (Floating-Point Processing Unit). These resources either need to be emulated in software, or avoided altogether.

2.1.4 Real Time Issues:

Embedded systems frequently control hardware, and must be able to respond to them in real time. Failure to do so could cause inaccuracy in measurements, or even damage hardware such as motors. This is made even more difficult by the lack of resources available. Almost all embedded systems need to be able to prioritize some tasks over others, and to be able to put off/skip low priority tasks such as UI in favor of high priority tasks like hardware control.

2.2 Need For Embedded Systems:

The uses of embedded systems are virtually limitless, because every day new products are introduced to the market that utilizes embedded computers in novel ways. In recent years, hardware such as microprocessors, microcontrollers, and FPGA chips have become much cheaper. So when implementing a new form of control, it's wiser to just buy the generic chip and write your own custom software for it. Producing a custom-made chip to handle a particular task or set of tasks costs far more time and money. Many embedded computers even come with extensive libraries, so that "writing your own software" becomes a very trivial task indeed. From an implementation viewpoint, there is a major difference between a computer and an embedded system. Embedded systems are often required to provide Real-Time response. The main elements that make embedded systems unique are its reliability and ease in debugging.

2.2.1 Debugging:

Embedded debugging may be performed at different levels, depending on the facilities available. From simplest to most sophisticate they can be roughly grouped into the following areas:

* Interactive resident debugging, using the simple shell provided by the embedded operating system (e.g. Forth and Basic)
* External debugging using logging or serial port output to trace operation using either a monitor in flash or using a debug server like the Remedy Debugger which even works for heterogeneous multi core systems.
* An in-circuit debugger (ICD), a hardware device that connects to the microprocessor via a JTAG or Nexus interface. This allows the operation of the microprocessor to be controlled externally, but is typically restricted to specific debugging capabilities in the processor.
* An in-circuit emulator replaces the microprocessor with a simulated equivalent, providing full control over all aspects of the microprocessor.
* A complete emulator provides a simulation of all aspects of the hardware, allowing all of it to be controlled and modified and allowing debugging on a normal PC.
* Unless restricted to external debugging, the programmer can typically load and run software through the tools, view the code running in the processor, and start or stop its operation. The view of the code may be as assembly code or source-code.

Because an embedded system is often composed of a wide variety of elements, the debugging strategy may vary. For instance, debugging a software(and microprocessor) centric embedded system is different from debugging an embedded system where most of the processing is performed by peripherals (DSP, FPGA, co-processor). An increasing number of embedded systems today use more than one single processor core. A common problem with multi-core development is the proper synchronization of software execution. In such a case, the embedded system design may wish to check the data traffic on the busses between the processor cores, which requires very low-level debugging, at signal/bus level, with a logic analyzer, for instance.

2.2.2 Reliability:

Embedded systems often reside in machines that are expected to run continuously for years without errors and in some cases recover by them if an error occurs. Therefore the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided.

Specific reliability issues may include:

* The system cannot safely be shut down for repair, or it is too inaccessible to repair. Examples include space systems, undersea cables, navigational beacons, bore-hole systems, and automobiles.
* The system must be kept running for safety reasons. "Limp modes" are less tolerable. Often backup s are selected by an operator. Examples include aircraft navigation, reactor control systems, safety-critical chemical factory controls, train signals, engines on single-engine aircraft.
* The system will lose large amounts of money when shut down: Telephone switches, factory controls, bridge and elevator controls, funds transfer and market making, automated sales and service.

A variety of techniques are used, sometimes in combination, to recover from errors—both software bugs such as memory leaks, and also soft errors in the hardware:

* Watchdog timer that resets the computer unless the software periodically notifies the watchdog
* Subsystems with redundant spares that can be switched over to
* software "limp modes" that provide partial function
* Designing with a Trusted Computing Base (TCB) architecture[6] ensures a highly secure & reliable system environment
* An Embedded Hypervisor is able to provide secure encapsulation for any subsystem component, so that a compromised software component cannot interfere with other subsystems, or privileged-level system software. This encapsulation keeps faults from propagating from one subsystem to another, improving reliability. This may also allow a subsystem to be automatically shut down and restarted on fault detection.
* Immunity Aware Programming

2.3 Explanation of Embedded Systems:

2.3.1 Software Architecture:

There are several different types of software architecture in common use.

* Simple Control Loop:

In this design, the software simply has a loop. The loop calls subroutines, each of which manages a part of the hardware or software.

* Interrupt Controlled System:

Some embedded systems are predominantly interrupt controlled. This means that tasks performed by the system are triggered by different kinds of events. An interrupt could be generated for example by a timer in a predefined frequency, or by a serial port controller receiving a byte. These kinds of systems are used if event handlers need low latency and the event handlers are short and simple.

Usually these kinds of systems run a simple task in a main loop also, but this task is not very sensitive to unexpected delays. Sometimes the interrupt handler will add longer tasks to a queue structure. Later, after the interrupt handler has finished, these tasks are executed by the main loop. This method brings the system close to a multitasking kernel with discrete processes.

* Cooperative Multitasking:

A non-preemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API. The programmer defines a series of tasks, and each task gets its own environment to “run” in. When a task is idle, it calls an idle routine, usually called “pause”, “wait”, “yield”, “nop” (stands for no operation), etc.The advantages and disadvantages are very similar to the control loop, except that adding new software is easier, by simply writing a new task, or adding to the queue-interpreter.

* Primitive Multitasking:

In this type of system, a low-level piece of code switches between tasks or threads based on a timer (connected to an interrupt). This is the level at which the system is generally considered to have an "operating system" kernel. Depending on how much functionality is required, it introduces more or less of the complexities of managing multiple tasks running conceptually in parallel.

As any code can potentially damage the data of another task (except in larger systems using an MMU) programs must be carefully designed and tested, and access to shared data must be controlled by some synchronization strategy, such as message queues, semaphores or a non-blocking synchronization scheme.

Because of these complexities, it is common for organizations to buy a real-time operating system, allowing the application programmers to concentrate on device functionality rather than operating system services, at least for large systems; smaller systems often cannot afford the overhead associated with a generic real time system, due to limitations regarding memory size, performance, and/or battery life.

* Micro kernels And Exokernels:

A microkernel is a logical step up from a real-time OS. The usual arrangement is that the operating system kernel allocates memory and switches the CPU to different threads of execution. User mode processes implement major functions such as file systems, network interfaces, etc.

In general, microkernels succeed when the task switching and intertask communication is fast, and fail when they are slow. Exokernels communicate efficiently by normal subroutine calls. The hardware and all the software in the system are available to, and extensible by application programmers. Based on performance, functionality, requirement the embedded systems are divided into three categories:

2.3.2 Stand Alone Embedded System:

These systems takes the input in the form of electrical signals from transducers or commands from human beings such as pressing of a button etc.., process them and produces desired output. This entire process of taking input, processing it and giving output is done in standalone mode. Such embedded systems comes under stand alone embedded systems

Eg: microwave oven, air conditioner etc..

2.3.3 Real-time embedded systems:

Embedded systems which are used to perform a specific task or operation in a specific time period those systems are called as real-time embedded systems. There are two types of real-time embedded systems.

* Hard Real-time embedded systems:

These embedded systems follow an absolute dead line time period i.e.., if the tasking is not done in a particular time period then there is a cause of damage to the entire equipment.

Eg: consider a system in which we have to open a valve within 30 milliseconds. If this valve is not opened in 30 ms this may cause damage to the entire equipment. So in such cases we use embedded systems for doing automatic operations.

* Soft Real Time embedded systems:

Eg: Consider a TV remote control system, if the remote control takes a few milliseconds delay it will not cause damage either to the TV or to the remote control. These systems which will not cause damage when they are not operated at considerable time period those systems comes under soft real-time embedded systems.

2.3.4 Network communication embedded systems:

A wide range network interfacing communication is provided by using embedded systems.

Eg:

* Consider a web camera that is connected to the computer with internet can be used to spread communication like sending pictures, images, videos etc.., to another computer with internet connection throughout anywhere in the world.
* Consider a web camera that is connected at the door lock.

Whenever a person comes near the door, it captures the image of a person and sends to the desktop of your computer which is connected to internet. This gives an alerting message with image on to the desktop of your computer, and then you can open the door lock just by clicking the mouse. Fig: 2.2 show the network communications in embedded systems.



**Fig 2.2: Network communication embedded systems**

2.3.5 Different types of processing units:

The central processing unit (c.p.u) can be any one of the following microprocessor, microcontroller, digital signal processing.

* Among these Microcontroller is of low cost processor and one of the main advantage of microcontrollers is, the components such as memory, serial communication interfaces, analog to digital converters etc.., all these are built on a single chip. The numbers of external components that are connected to it are very less according to the application.
* Microprocessors are more powerful than microcontrollers. They are used in major applications with a number of tasking requirements. But the microprocessor requires many external components like memory, serial communication, hard disk, input output ports etc.., so the power consumption is also very high when compared to microcontrollers.
* Digital signal processing is used mainly for the applications that particularly involved with processing of signals

2.4 APPLICATIONS OF EMBEDDED SYSTEMS:

2.4.1 Consumer applications:

At home we use a number of embedded systems which include microwave oven, remote control, vcd players, dvd players, camera etc….



**Fig2.3: Automatic coffee makes equipment**

2.4.2 Office automation:

We use systems like fax machine, modem, printer etc…



**Fig2.4: Fax machine Fig2.5: Printing machine**

2.4.3. Industrial automation:

Today a lot of industries are using embedded systems for process control. In industries we design the embedded systems to perform a specific operation like monitoring temperature, pressure, humidity ,voltage, current etc.., and basing on these monitored levels we do control other devices, we can send information to a centralized monitoring station.



**Fig2.6: Vehicle**

In critical industries where human presence is avoided there we can use vehicle**s** which are programmed to do a specific operation.

2.4.5 Computer networking:

Embedded systems are used as bridges routers etc..



**Fig2.7: Computer networking**

2.4.6 Tele communications:

Cell phones, web cameras etc.



**Fig2.8: Cell PhoneFig2.9: Web camera**

CHAPTER 3: PROJECT DESCRIPTION

3.1 Block Diagram Of Solar Powered Multipurpose Autonomous Agricultural Robot

3.2 Advantages And Disadvantages:

ADVANTAGES:

1. This increases the efficiency of seed sowing, pesticide spraying and grass cutting.
2. Reduces the problem encountered in manual planting.
3. No need of man power.
4. Efficient .

DISADVANTAGES:

1. System with high cost and high complexity

2. Breaking queues

3. Power issue in case of weather instability.

Applications:

1. Agricultural Sector
2. Local Farming

CHAPTER 4: HARDWARE & SOFTWARE DESCRIPTION

1. HARDWARE DESCRIPTION:

4.1 Introduction:

**Hardware components are**

1. Rechargeable battery.
2. Solar panel.
3. Microcontroller (ESP32).
4. Soil sensor.
5. L293D motor drivers.
6. DC motors.

4.2 SOLAR PANEL:

Renewable energy is critical to our fight against climate change. We simply must shift our world to a low-carbon economy and away from oil and coal.

Experts agree we need a substantial reduction in CO2 over the next 40-50 years and this means we need renewable energy to replace fossil fuels now.

Presently, most of the energy what we are using is from non renewable sources like petrol, coal etc.., which are very limited.

Apart from the non-renewable energy sources, renewable energy sources like wind energy, solar energy etc.., are used then we can save non-renewable sources for longer time.

* A solar cell (also called photovoltaic cell) is a solid state device that converts the energy of sunlight directly into electricity by the photovoltaic effect.
* Assemblies of cells are used to make solar modules, also known as solar panels.
* The energy generated from these solar modules, referred to as solar power, is an example of solar energy.



* AdvantagesThey have no moving parts and hence require little maintenance and work quite satisfactorily without any focusing device.
* It does not cause any environmental pollution like the fossil fuels and nuclear power.
* Solar cells last a longer time and have low running costs
* Low power consumption.
* Conservation of energy.
* Utilization of free available source of energy from sun
* Storage of energy into rechargeable battery.
* Stored energy is used for grass cutter.
* Motor automation.
* High efficiency can be achieved with relay switch.
* By using this project we can save more power. That is we can reduce the wastage of power.
* Here no need of man. The circuit itself checks the presence of vehicle and also checks weather it is day or night time. Once we switch on the circuit it automatically performs all this actions without manual operation.
* At night time also whenever vehicle comes at that time only light brightness increases after few seconds it will come to normal position that is decreases light brightness.
* It is the most advantage of this project. For these all reasons in future this project may be used in all streets to save power.

4.3 RECHARGEABLE BATTERY:

A **rechargeable battery**, **storage battery**, **secondary cell**, or **accumulator** is a type of [electrical battery](https://en.wikipedia.org/wiki/Electrical_battery) which can be charged, discharged into a load, and recharged many times, as opposed to a disposable or [primary battery](https://en.wikipedia.org/wiki/Primary_battery), which is supplied fully charged and discarded after use. It is composed of one or more [electrochemical cells](https://en.wikipedia.org/wiki/Electrochemical_cell). The term "accumulator" is used as it [accumulates](https://en.wikipedia.org/wiki/Accumulator_(energy)) and [stores energy](https://en.wikipedia.org/wiki/Energy_storage) through a reversible [electrochemical](https://en.wikipedia.org/wiki/Electrochemical) [reaction](https://en.wikipedia.org/wiki/Chemical_reaction). Rechargeable batteries are produced in many different shapes and sizes, ranging from [button cells](https://en.wikipedia.org/wiki/Button_cell#Rechargeable_variants) to megawatt systems connected to [stabilize](https://en.wikipedia.org/wiki/Grid_energy_storage) an [electrical distribution network](https://en.wikipedia.org/wiki/Electrical_distribution_network). Several different combinations of [electrode](https://en.wikipedia.org/wiki/Electrode) [materials](https://en.wikipedia.org/wiki/Material) and [electrolytes](https://en.wikipedia.org/wiki/Electrolyte) are used, including [lead–acid](https://en.wikipedia.org/wiki/Lead%E2%80%93acid_battery), [nickel–cadmium](https://en.wikipedia.org/wiki/Nickel%E2%80%93cadmium_battery) (NiCd), [nickel–metal hydride](https://en.wikipedia.org/wiki/Nickel%E2%80%93metal_hydride_battery) (NiMH), [lithium-ion](https://en.wikipedia.org/wiki/Lithium-ion_battery) (Li-ion), and [lithium-ion polymer](https://en.wikipedia.org/wiki/Lithium_polymer_battery)(Li-ion polymer).

Rechargeable batteries typically initially cost more than disposable batteries, but have a much lower [total cost of ownership](https://en.wikipedia.org/wiki/Total_cost_of_ownership) and [environmental impact](https://en.wikipedia.org/wiki/Environmental_impact), as they can be recharged inexpensively many times before they need replacing. Some rechargeable battery types are available in the same [sizes](https://en.wikipedia.org/wiki/List_of_battery_sizes) and voltages as disposable types, and can be used interchangeably with them.

Fig:A rechargeable polymer mobile battery

Charging and discharging:

During charging, the positive active material is [oxidized](https://en.wikipedia.org/wiki/Oxidized), producing [electrons](https://en.wikipedia.org/wiki/Electron), and the negative material is [reduced](https://en.wikipedia.org/wiki/Redox), consuming electrons. These electrons constitute the [current](https://en.wikipedia.org/wiki/Electric_current) flow in the external [circuit](https://en.wikipedia.org/wiki/Electrical_network). The [electrolyte](https://en.wikipedia.org/wiki/Electrolyte) may serve as a simple buffer for internal [ion](https://en.wikipedia.org/wiki/Ion) flow between the [electrodes](https://en.wikipedia.org/wiki/Electrode), as in [lithium-ion](https://en.wikipedia.org/wiki/Lithium-ion_battery) and [nickel-cadmium](https://en.wikipedia.org/wiki/Nickel-cadmium_battery) cells, or it may be an active participant in the [electrochemical](https://en.wikipedia.org/wiki/Electrochemical) reaction, as in [lead–acid](https://en.wikipedia.org/wiki/Lead%E2%80%93acid_battery)cells.

The energy used to charge rechargeable batteries usually comes from a [battery charger](https://en.wikipedia.org/wiki/Battery_charger) using AC [mains electricity](https://en.wikipedia.org/wiki/Mains_electricity), although some are equipped to use a vehicle's 12-volt DC power outlet. The voltage of the source must be higher than that of the battery to force current to flow into it, but not too much higher or the battery may be damaged.

Chargers take from a few minutes to several hours to charge a battery. Slow "dumb" chargers without voltage or temperature-sensing capabilities will charge at a low rate, typically taking 14 hours or more to reach a full charge. Rapid chargers can typically charge cells in two to five hours, depending on the model, with the fastest taking as little as fifteen minutes. Fast chargers must have multiple ways of detecting when a cell reaches full charge (change in terminal voltage, temperature, etc.) to stop charging before harmful overcharging or overheating occurs. The fastest chargers often incorporate cooling fans to keep the cells from overheating. Battery packs intended for rapid charging may include a temperature sensor that the charger uses to protect the pack; the sensor will have one or more additional electrical contacts.

Different battery chemistries require different charging schemes. For example, some battery types can be safely recharged from a constant voltage source. Other types need to be charged with a regulated current source that tapers as the battery reaches fully charged voltage. Charging a battery incorrectly can damage a battery; in extreme cases, batteries can overheat, catch fire, or explosively vent their contents.

### Rate of discharge

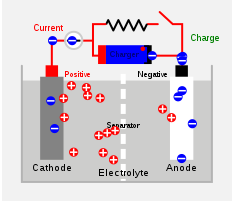


Fig: Diagram of the charging of a secondary cell or battery.

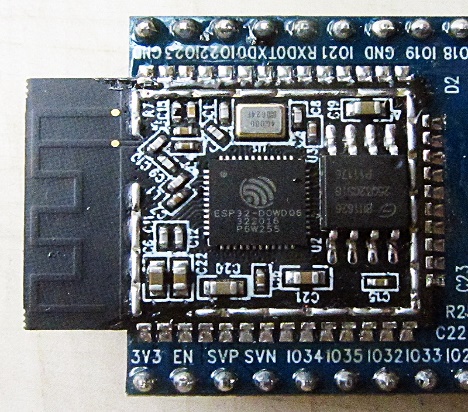
Battery charging and discharging rates are often discussed by referencing a "C" rate of current. The C rate is that which would theoretically fully charge or discharge the battery in one hour. For example, [trickle charging](https://en.wikipedia.org/wiki/Trickle_charging) might be performed at C/20 (or a "20 hour" rate), while typical charging and discharging may occur at C/2 (two hours for full capacity). The available capacity of electrochemical cells varies depending on the discharge rate. Some energy is lost in the internal resistance of cell components (plates, electrolyte, interconnections), and the rate of discharge is limited by the speed at which chemicals in the cell can move about. For lead-acid cells, the relationship between time and discharge rate is described by [Peukert's law](https://en.wikipedia.org/wiki/Peukert%27s_law" \o "Peukert's law); a lead-acid cell that can no longer sustain a usable terminal voltage at a high current may still have usable capacity, if discharged at a much lower rate. Data sheets for rechargeable cells often list the discharge capacity on 8-hour or 20-hour or other stated time; cells for [uninterruptible power supply](https://en.wikipedia.org/wiki/Uninterruptible_power_supply) systems may be rated at 15 minute discharge.

The terminal voltage of the battery is not constant during charging and discharging. Some types have relatively constant voltage during discharge over much of their capacity. Non-rechargeable alkaline and [zinc–carbon cells](https://en.wikipedia.org/wiki/Zinc%E2%80%93carbon_cell) output 1.5V when new, but this voltage drops with use. Most NiMH [AA](https://en.wikipedia.org/wiki/AA_battery) and [AAA cells](https://en.wikipedia.org/wiki/AAA_cell) are rated at 1.2 V, but have a flatter [discharge curve](https://en.wikipedia.org/w/index.php?title=Discharge_curve&action=edit&redlink=1) than alkalines and can usually be used in equipment designed to use [alkaline batteries](https://en.wikipedia.org/wiki/Alkaline_batteries).

Battery manufacturers' technical notes often refer to voltage per cell (VPC) for the individual cells that make up the battery. For example, to charge a 12 V lead-acid battery (containing 6 cells of 2 V each) at 2.3 VPC requires a voltage of 13.8 V across the battery's terminals.

4.4 ESP 32 DEVELOPMENT BOARD:

**ESP32** is a series of low-cost, low-power [system on a chip](https://en.wikipedia.org/wiki/System_on_a_chip) [microcontrollers](https://en.wikipedia.org/wiki/Microcontroller) with integrated [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) and dual-mode [Bluetooth](https://en.wikipedia.org/wiki/Bluetooth). The ESP32 series employs a [Tensilica](https://en.wikipedia.org/wiki/Tensilica" \o "Tensilica) [Xtensa LX6](https://en.wikipedia.org/wiki/Xtensa_LX6" \o "Xtensa LX6) microprocessor in both dual-core and single-core variations and includes in-built antenna switches, RF [balun](https://en.wikipedia.org/wiki/Balun" \o "Balun), power amplifier, low-noise receive amplifier, filters, and power-management modules. ESP32 is created and developed by [Espressif Systems](https://en.wikipedia.org/w/index.php?title=Espressif_Systems&action=edit&redlink=1" \o "Espressif Systems (page does not exist)), a Shanghai-based Chinese company, and is manufactured by [TSMC](https://en.wikipedia.org/wiki/TSMC) using their 40 nm process.[[2]](https://en.wikipedia.org/wiki/ESP32#cite_note-EspressifESP32ProductPage-2) It is a successor to the [ESP8266](https://en.wikipedia.org/wiki/ESP8266) microcontroller.

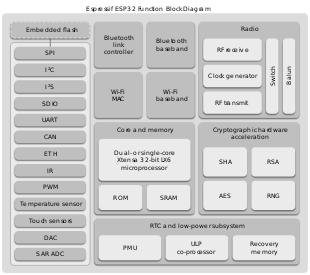


ESP-WROOM-32 module with ESP32‑D0WDQ6 chip

4.4.1.FEATURES:

Features of the ESP32 include the following:

ESP32 function block diagram.

[](https://en.wikipedia.org/wiki/File:Espressif_ESP32_Function_Block_Diagram.svg)

Processors:

* + CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 [DMIPS](https://en.wikipedia.org/wiki/Dhrystone)
  + Ultra low power (ULP) co-processor
* Memory: 520 KiB SRAM
* Wireless connectivity:
  + Wi-Fi: [802.11](https://en.wikipedia.org/wiki/IEEE_802.11) b/g/n
  + Bluetooth: v4.2 BR/EDR and BLE
* Peripheral interfaces:
  + 12-bit [SAR ADC](https://en.wikipedia.org/wiki/Successive_approximation_ADC) up to 18 channels
  + 2 × 8-bit [DACs](https://en.wikipedia.org/wiki/Digital-to-analog_converter)
  + 10 × touch sensors ([capacitive sensing](https://en.wikipedia.org/wiki/Capacitive_sensing) GPIOs)
  + Temperature sensor
  + 4 × [SPI](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)
  + 2 × [I²S](https://en.wikipedia.org/wiki/I%C2%B2S) interfaces
  + 2 × [I²C](https://en.wikipedia.org/wiki/I%C2%B2C) interfaces
  + 3 × [UART](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter)
  + [SD](https://en.wikipedia.org/wiki/Secure_Digital)/[SDIO](https://en.wikipedia.org/wiki/Secure_Digital#SDIO_cards)/[CE-ATA](https://en.wikipedia.org/wiki/CE-ATA)/[MMC](https://en.wikipedia.org/wiki/MultiMediaCard)/[eMMC](https://en.wikipedia.org/wiki/MultiMediaCard" \l "eMMC" \o "MultiMediaCard) host controller
  + SDIO/SPI slave controller
  + [Ethernet](https://en.wikipedia.org/wiki/Ethernet) MAC interface with dedicated DMA and [IEEE 1588 Precision Time Protocol](https://en.wikipedia.org/wiki/Precision_Time_Protocol) support
  + [CAN bus](https://en.wikipedia.org/wiki/CAN_bus) 2.0
  + Infrared remote controller (TX/RX, up to 8 channels)
  + Motor [PWM](https://en.wikipedia.org/wiki/Pulse-width_modulation)
  + LED [PWM](https://en.wikipedia.org/wiki/Pulse-width_modulation) (up to 16 channels)
  + [Hall effect sensor](https://en.wikipedia.org/wiki/Hall_effect_sensor)
  + Ultra low power analog pre-amplifier
* Security:
  + IEEE 802.11 standard security features all supported, including WFA, WPA/WPA2 and WAPI
  + Secure boot
  + Flash encryption
  + 1024-bit OTP, up to 768-bit for customers
  + Cryptographic hardware acceleration: [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard), [SHA-2](https://en.wikipedia.org/wiki/SHA-2), [RSA](https://en.wikipedia.org/wiki/RSA_(cryptosystem)), [elliptic curve cryptography](https://en.wikipedia.org/wiki/Elliptic_curve_cryptography) (ECC), [random number generator](https://en.wikipedia.org/wiki/Random_number_generator) (RNG)
* Power management:
  + Internal [low-dropout regulator](https://en.wikipedia.org/wiki/Low-dropout_regulator)
  + Individual power domain for RTC
  + 5uA deep sleep current
  + Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

## 4.4.2 QFN packaged chip and module

ESP32 is housed in Quad-Flat No-leads (QFN) packages of varying sizes with 49 pads. Specifically, 48 connection pads along the sides and one large thermal pad (connected to ground) on the bottom.

### Chips

The ESP32 system on a chip integrated circuit is packaged in both 6 mm × 6 mm and 5 mm × 5 mm sized QFN packages.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier** | **Processor cores** | **Embedded flash memory ([MiB](https://en.wikipedia.org/wiki/MiB" \o "MiB))** | **Package size** | **Description** |
| ESP31B | 2 | 0 | 6×6 mm2 | Pre-release [SoC](https://en.wikipedia.org/wiki/System_on_a_chip" \o "System on a chip) used for beta testing; no longer available. |
| ESP32-D0WDQ6 | 2 | 0 | 6×6 mm2 | Initial production release chip of the ESP32 series. |
| ESP32‑D0WD | 2 | 0 | 5×5 mm2 | Smaller physical package variation similar to ESP32-D0WDQ6. |
| ESP32‑D2WD | 2 | 2 | 5×5 mm2 | 2 MiB (16 Mibit) embedded flash memory variation. |
| ESP32‑S0WD | 1 | 0 | 5×5 mm2 | Single-core processor variation. |

### Module

The ESP32-PICO-D4 [system in package](https://en.wikipedia.org/wiki/System_in_package) module combines an ESP32 silicon chip, crystal oscillator, flash memory chip, filter capacitors, and RF matching links into a single 7 mm × 7 mm sized QFN package.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier** | **Processor cores** | **Embedded flash memory ([MiB](https://en.wikipedia.org/wiki/MiB" \o "MiB))** | **Package size** | **Description** |
| ESP32-PICO-D4 | 2 | 4 | 7×7 mm2 | Includes ESP32 chip, crystal oscillator, flash memory, filter capacitors, and RF matching links.[[4]](https://en.wikipedia.org/wiki/ESP32#cite_note-EspressifESP32PICOD4-4) |

## Printed circuit boards

### Surface-mount module boards

ESP32 based surface-mount printed circuit board modules directly contain the ESP32 SoC and are designed to be easily integrated onto other circuit boards. Meandered inverted-F antenna designs are used for the PCB trace antennas on the modules listed below. In addition to flash memory, some modules include pseudostatic RAM (pSRAM).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Vendor** | **Name** | **Antenna** | **Flash memory ([MiB](https://en.wikipedia.org/wiki/Mebibyte" \o "Mebibyte))** | **pSRAM ([MiB](https://en.wikipedia.org/wiki/Mebibyte" \o "Mebibyte))** | **Description** |
| Espressif | ESP-WROOM-03 | PCB trace | 4 | 0 | Discontinued. Limited distribution, pre-production module created by Espressif for beta testing purposes; this module used the ESP31B, the beta testing chip for the ESP32 series. FCC Part 15.247 tested (FCC ID: 2AC7Z-ESP32). |
| ESP-WROOM-32 | PCB trace | 4 | 0 | First publicly available ESP32 module board created by Espressif.  FCC Part 15.247 tested (FCC ID: 2AC7Z-ESPWROOM32).  Based on ESP32-D0WDQ6 chip. |
| ESP-WROOM-32D | PCB trace | 4 | 0 | Revision of the ESP-WROOM-32 module which uses an ESP32-D0WD chip instead of an ESP32-D0WDQ6 chip. |
| ESP32-WROOM-32U | [U.FL socket](https://en.wikipedia.org/wiki/Hirose_U.FL) | 4 | 0 | Alternative to the ESP-WROOM-32D module which has a U.FL connector for external antenna in lieu of a PCB trace antenna. |
| ESP32-WROVER | PCB trace | 4 | 4 | ESP32 module board with 4 MiBpSRAM created by Espressif. FCC part 15.247 tested (FCC ID 2AC7Z-ESP32WROVER). Uses 40 MHz crystal oscillator. Does not include U.FL connector.  Based on ESP32-D0WDQ6 chip. |
| ESP32-WROVER-I | U.FL socket, PCB trace | 4 | 4 | Variation of ESP32-WROVER module configured to use an on-board U.FL compatible connector. PCB trace antenna not connected by default. |
| Ai-Thinker | ESP-32S | PCB trace | 4 | 0 | ESP32 module based on the form factor of the Espressif ESP-WROOM-32 module. The ESP-32S module replaced the unreleased ESP3212 module. |
| AnalogLamb | ESP-32S-ALB | PCB trace | 4 | 0 | Clone of the ESP-32S module (ESP-WROOM-32 compatible footprint). Seen with a green solder mask coating. |
| ALB-WROOM | PCB trace | 16 | 0 | Variation of ESP-32S-ALB with 16 MiB of flash memory. |
| ALB32-WROVER | PCB trace | 4 | 4 | ESP32 module board with 4 MiBpSRAM with the same footprint as the ESP-WROOM-32 module. |
| DFRobot | ESP-WROOM-32 | PCB trace | 4 | 0 | Module board similar to EspressifSystems's ESP-WROOM-32, but is not FCC certified, and uses 26 MHz or 32 kHz crystal oscillator. |
| eBox&Widora | ESP32-Bit | Ceramic, U.FL socket | 4 | 0 | Module has a ceramic antenna and an U.FL antenna connector. This module has a different footprint than the ESP-WROOM-32/ESP-32S modules. |
| Goouuu Tech | ESP-32F | PCB trace | 4 | 0 | Module board similar to EspressifSystems's ESP-WROOM-32. FCC certified (ID 2AM77-ESP-32F). |
| IntoRobot | W32 | PCB trace | 4 | 0 | Module similar in appearance to Espressif's ESP-WROOM-32, but footprint pinout differs. |
| W33 | Ceramic, U.FL socket | 4 | 0 | Differs from IntoRobot W32 module in its antenna configuration. |
| ITEAD | PSH-C32 | PCB trace | 1[[19]](https://en.wikipedia.org/wiki/ESP32#cite_note-IteadPSHC32Schematic-19) | 0 | Module has unusually small flash memory on board. Also, footprint is unique and differs from all other ESP32 modules. |
| Pycom[[21]](https://en.wikipedia.org/wiki/ESP32" \l "cite_note-PycomOEMProducts-21) | W01 | (Not included.) | 8 | 4 | OEM module version of the WiPy 2.0. Supports Wi-Fi and Bluetooth. FCC ID 2AJMTWIPY01R. |
| L01 | (Not included.) | 8 | 4 | OEM module version of the LoPy. Supports Wi-Fi, Bluetooth, and LoRa. FCC ID 2AJMTLOPY01R. |
| L04 | (Not included.) | 8 | 4 | OEM module version of the LoPy4. Supports Wi-Fi, Bluetooth, LoRa, and Sigfox. |
| S01 | (Not included.) | 8 | 4 | Discontinued. OEM module version of the SiPy. Supported Wi-Fi, Bluetooth, and Sigfox (14 dBm and 22 dBm). |
| G01 | (Not included.) | 8 | 4 | OEM module version of the GPy. Supports Cellular LTE-CAT M1/NB1, Wi-Fi and Bluetooth. |
| u-blox | NINA-W131 | (Not included.) | 2 | 0 | Belongs to the u-blox NINA-W13 series of Wi-Fi modules. |
| NINA-W132 | PIFA | 2 | 0 | Belongs to the u-blox NINA-W13 series of Wi-Fi modules. Onboard planar inverted-F antenna (PIFA) is shaped (cut & bent) metal, not a PCB trace. |

### Development and other boards

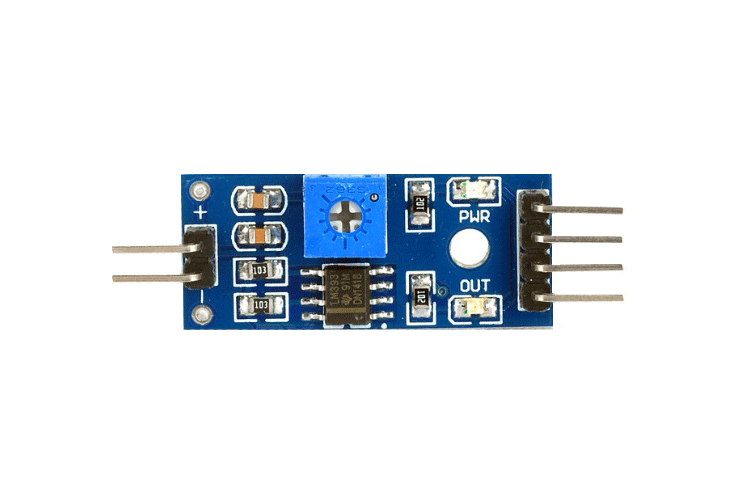
Development & break-out boards extend wiring and may add functionality, often building upon ESP32 module boards and making them easier to use for development purposes (especially with [breadboards](https://en.wikipedia.org/wiki/Breadboard)).

|  |  |  |  |
| --- | --- | --- | --- |
| **Vendor** | **Name** | **Surface-mount module used** | **Description** |
| Espressif | ESP\_Module\_Testboard | ESP-WROOM-03 | Break-out board included with ESP-WROOM-03 beta modules. |
| ESP32\_Demo Board\_V2 | ESP-WROOM-32 | Development & demonstration board created by Espressif. |
| ESP32-DevKitC | ESP-WROOM-32 | Compact development board created by Espressif. Silkscreen labeling on PCB reads "Core Board". |
| ESP-WROVER-KIT | ESP-WROOM-32 or ESP32-WROVER | Large development board created by Espressif. Previously named ESP32-DevKitJ. |
| ESP32-PICO-KIT | ESP32-PICO-D4 | Small development board created by Espressif. FCC ID 2AC7Z-ESP32PICOKIT. |
| Adafruit | HUZZAH32 | ESP-WROOM-32 | Also referred to as the "ESP32 Feather Board", the HUZZAH32 is a compact development board/module that is compatible with the Adafruit Feather family of products. |
| Ai-Thinker | NodeMCU-32S | ESP-32S | NodeMCU-like development board. |
| AnalogLamb | ESP32 Development Board | ESP-32S-ALB or ALB-WROOM | Development board similar to Espressif's ESP32-DevKitC with on board a CP2102 USB/serial bridge. 4 MiB variation uses ESP-32S-ALB; 16 MiB variation uses ALB-WROOM module. |
| Maple ESP32 | ESP-32S-ALB | Development board with Arduino-style connections and CP2104 USB/serial interface. |
| April Brother | ESPea32 | † | Development board with [perfboard](https://en.wikipedia.org/wiki/Perfboard" \o "Perfboard) area that may be optionally cut-off. |
| ArduCAM | ESP32 UNO | ESP-32S | Arduino Uno-like development board based on ESP32 IoT UNO framework with support for SPI ArduCAM, Battery pins and uSD card slot. |
| DoIT | ESPduino32 | ESP-WROOM-32 | Full-featured Arduino Uno-like development board compatible with Arduino Shields. It also adds additional SPI & IO pins. |
| EzSBC | ESP32-01 Breakout and Development Board | ESP-WROOM-32 | Full-featured development board with two tri-color LEDs and fits on a breadboard. |
| Gravitech&MakerAsia | Nano32 | † | Development board that directly incorporates the ESP32 chip. |
| HydraBus | HydraESP32 | ESP-WROOM-32 or ESP-32S | HydraESP32 HydraBus v1.1 Rev1 shield/breakout board for ESP-WROOM-32 or ESP-32S. This shield can be used with or without a HydraBus board. |
| Noduino | Quantum | † | Arduino-style development board that directly incorporates the ESP32 chip. |
| Pycom | WiPy | † | [MicroPython](https://en.wikipedia.org/wiki/MicroPython) programmable Wi-Fi & Bluetooth IoT development platform with a 1 km Wi-Fi range. WiPy versions 2.0 and 3.0 use ESP32. |
| LoPy | † | Triple network Pycom board featuring LoRa, Wi-Fi (1 km range), and BLE. |
| LoPy4 | ? | Quadruple network Pycom board featuring LoRa, Sigfox, Wi-Fi (1 km range), and BLE. |
| SiPy | † | Triple network Pycom board featuring Sigfox, Wi-Fi (1 km range), and BLE. |
| GPy | † | Triple network Pycom board featuring LTE-M, Wi-Fi (1 km range), and BLE. |
| FiPy | † | Quintuple network Pycom board featuring LTE-M, LoRa, Sigfox, Wi-Fi (1 km range), and BLE. |
| SparkFun | ESP32 Thing | † | Compact development board with FTDI FT231x USB/serial interface and LiPo charger built-in. |
| SunDUINO | ESP32 MiniBoard | ESP-WROOM-32 | Breakout compatible with the Espressif ESP32-DevKitC. Lacks on-board USB-UART. |
| ESP32 SunDUINO | ESP-WROOM-32 or ESP-32S | Arduino-style development board. Lacks on-board USB-UART. |
| SwitchDoc Labs | BC24 | ESP-WROOM-32 | ESP32 Breakout with 24 SK6812RGBW LEDs with Grove Connectors for easy prototyping. Comes with USB-UART and Feather compatible pinout. |
| Watterott | ESP-WROOM32-Breakout | ESP-WROOM-32 | Breakout which is compatible with the Espressif ESP32-DevKitC. |
| WEMOS[[32]](https://en.wikipedia.org/wiki/ESP32#cite_note-WEMOSProducts-32) | LOLIN32 | ESP-WROOM-32 |  |
| LOLIN32 Lite | † |  |
| LOLIN32 Pro | ESP32-WROVER | Has MicroSD card slot (supports SD and SPI mode) |
| Widora | Air | † | Compact ESP32 development board. |

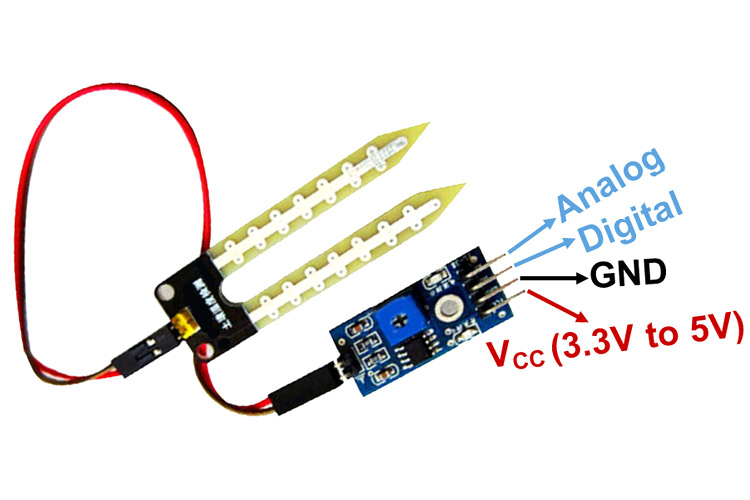
† ESP32 SoC incorporated directly onto development board; no module board used.

4.5 SOIL MOISTURE SENSOR:

Soil moisture sensor module



Soil moisture sensor pin out



The Moisture sensor is used to measure the water content (moisture) of soil. When the soil is having water shortage, the module output is at high level; else the output is at low level. This sensor reminds the user to water their plants and also monitors the moisture content of soil. It has been widely used in agriculture, land irrigation and botanical gardening.

Moisture Sensor Module Pin out Configuration

|  |  |
| --- | --- |
| **Pin Name** | **Description** |
| VCC | The Vcc pin powers the module, typically with +5V |
| GND | Power Supply Ground |
| DO | Digital Out Pin for Digital Output. |
| AO | Analog Out Pin for Analog Output |

4.5.1 Working Principle of Moisture Sensor

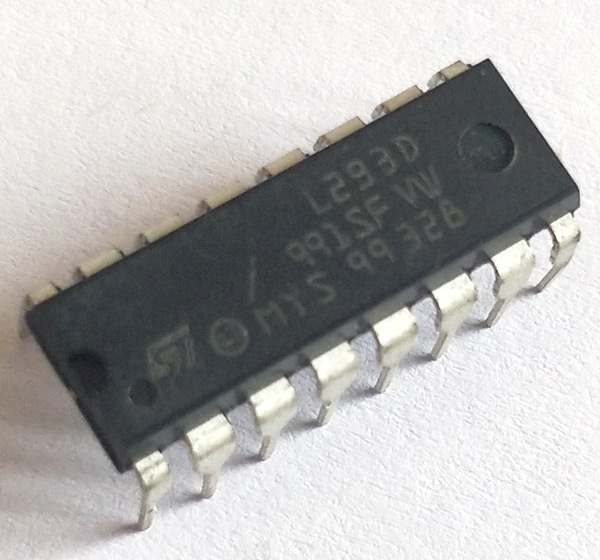
The Soil Moisture Sensor uses capacitance to measure dielectric permittivity of the surrounding medium. In soil, dielectric permittivity is a function of the water content. The sensor creates a voltage proportional to the dielectric permittivity, and therefore the water content of the soil. The sensor averages the water content over the entire length of the sensor. There is a 2 cm zone of influence with respect to the flat surface of the sensor, but it has little or no sensitivity at the extreme edges.The Soil Moisture Sensor is used to measure the loss of moisture over time due to evaporation and plant uptake,evaluate optimum soil moisture contents for various species of plants,monitor soil moisture content to control irrigation in greenhouses and enhance bottle biology experiments.

4.5.2 Moisture Sensor Module Features & Specifications

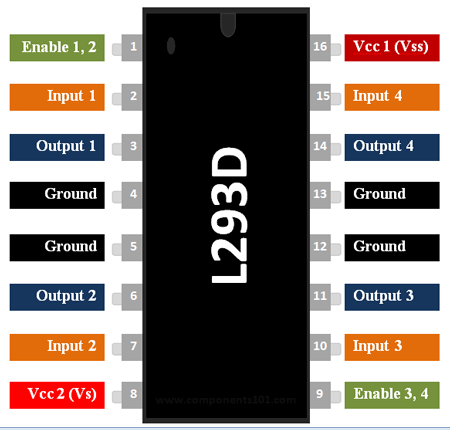
* Operating Voltage: 3.3V to 5V DC
* Operating Current: 15mA
* Output Digital - 0V to 5V, Adjustable trigger level from preset
* Output Analog - 0V to 5V based on infrared radiation from fire flame falling on the sensor
* LEDs indicating output and power
* PCB Size: 3.2cm x 1.4cm
* LM393 based design
* Easy to use with Microcontrollers or even with normal Digital/Analog IC
* Small, cheap and easily available

4.6 L293D MOTOR DRIVER:

L293D Motor Driver IC



Motor Driver IC L293D Pin out



4.6.1 L293D PIN DESCRIPTION:

|  |  |  |
| --- | --- | --- |
| Pin Number | Pin Name | Description |
| 1 | Enable 1,2 | This pin enables the input pin Input 1(2) and Input 2(7) |
| 2 | Input 1 | Directly controls the Output 1 pin. Controlled by digital circuits |
| 3 | Output 1 | Connected to one end of  Motor 1 |
| 4 | Ground | Ground pins are connected to ground of circuit (0V) |
| 5 | Ground | Ground pins are connected to ground of circuit (0V) |
| 6 | Output 2 | Connected to another end of  Motor 1 |
| 7 | Input 2 | Directly controls the Output 2 pin. Controlled by digital circuits |
| 8 | Vcc2 (Vs) | Connected to Voltage pin for running motors (4.5V to 36V) |
| 9 | Enable 3,4 | This pin enables the input pin Input 3(10) and Input 4(15) |
| 10 | Input 3 | Directly controls the Output 3 pin. Controlled by digital circuits |
| 11 | Output 3 | Connected to one end of Motor 2 |
| 12 | Ground | Ground pins are connected to ground of circuit (0V) |
| 13 | Ground | Ground pins are connected to ground of circuit (0V) |
| 14 | Output 4 | Connected to another end of Motor 2 |
| 15 | Input 4 | Directly controls the Output 4 pin. Controlled by digital circuits |
| 16 | Vcc2 (Vss) | Connected to +5V to enable IC function |

4.6.2 FEATURES

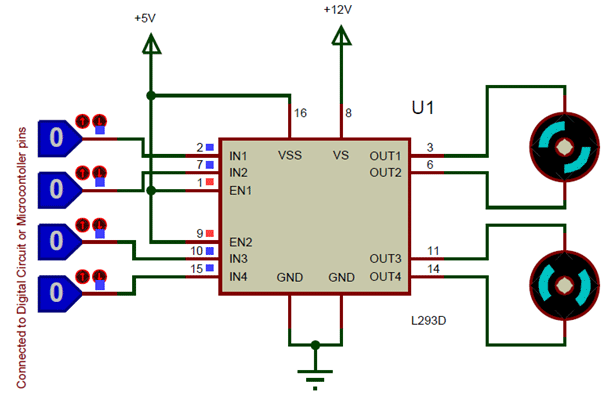
* Can be used to run Two DC motors with the same IC.
* Speed and Direction control is possible
* Motor voltage Vcc2 (Vs): 4.5V to 36V
* Maximum Peak motor current: 1.2A
* Maximum Continuous Motor Current: 600mA
* Supply Voltage to Vcc1(vss): 4.5V to 7V
* Transition time: 300ns (at 5Vand 24V)
* Automatic Thermal shutdown is available
* Available in 16-pin DIP, TSSOP, SOIC packages

4.6.3 Where to use L293D IC

The L293D is a popular 16-Pin **Motor Driver IC**. As the name suggests it is mainly used to drive motors. A single **L293D IC** is capable of running two [DC motors](https://components101.com/motors/toy-dc-motor) at the same time; also the direction of these two motors can be controlled independently. So if you have motors which has operating voltage less than 36V and operating current less than 600mA, which are to be controlled by digital circuits like Op-Amp, [555 timers](https://components101.com/555-timer-ic-pinout-datasheet), digital gates or even Micron rollers like Arduino, PIC, ARM etc.. This IC will be the right choice for you.

4.6.4 How to use a L293D Motor Driver IC

**Using this L293D motor driver IC** is very simple. The IC works on the principle of **Half H-Bridge**, let us not go too deep into what H-Bridge means, but for now just know that H bridge is a set up which is used to run motors both in clock wise and anti clockwise direction. As said earlier this IC is capable of running two motors at the any direction at the same time, the circuit to achieve the same is shown below.



All the Ground pins should be grounded. There are two power pins for this IC, one is the Vss(Vcc1) which provides the voltage for the IC to work, this must be connected to +5V. The other is Vs(Vcc2) which provides voltage for the motors to run, based on the specification of your motor you can connect this pin to anywhere between 4.5V to 36V, here I have connected to +12V.

The Enable pins (Enable 1,2 and Enable  3,4)  are used to Enable Input pins for Motor 1 and Motor 2 respectively. Since in most cases we will be using both the motors both the pins are held high by default by connecting to +5V supply. The input pins Input 1,2 are used to control the motor 1 and Input pins 3,4 are used to control the Motor 2. The input pins are connected to the any Digital circuit or microcontroller to control the speed and direction of the motor. You can toggle the input pins based on the following table to control your motor.

|  |  |  |
| --- | --- | --- |
| Input 1 = HIGH(5v) | Output 1 = HIGH | Motor 1 rotates in Clock wise Direction |
| Input 2 = LOW(0v) | Output 2 = LOW |  |
| Input 3 = HIGH(5v) | Output 1 = HIGH | Motor 2 rotates in Clock wise Direction |
| Input 4 = LOW(0v) | Output 2 = LOW |  |

|  |  |  |
| --- | --- | --- |
| Input 1 = LOW(0v) | Output 1 = LOW | Motor 1 rotates in Anti-Clock wise Direction |
| Input 2 = HIGH(5v) | Output 2 = HIGH |  |
| Input 3 = LOW(0v) | Output 1 = LOW | Motor 2 rotates in Anti -Clock wise Direction |
| Input 4 = HIGH(5v) | Output 2 = HIGH |  |

|  |  |  |
| --- | --- | --- |
| Input 1 = HIGH(5v) | Output 1 = HIGH | Motor 1 stays still |
| Input 2 = HIGH(5v) | Output 2 = HIGH |  |
| Input 3 = HIGH(5v) | Output 1 = LOW | Motor 2 stays still |
| Input 4 = HIGH(5v) | Output 2 = HIGH |  |

4.6.5 APPLICATIONS

* Used to drive high current Motors using Digital Circuits
* Can be used to drive [Stepper motors](https://components101.com/motors/28byj-48-stepper-motor)
* High current LED’s can be driven
* [Relay](https://components101.com/5v-relay-pinout-working-datasheet) Driver module (Latching Relay is possible)

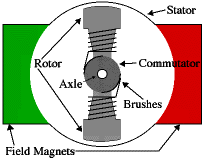
4.7 DC MOTOR:

****

DC motors are configured in many types and sizes, including brush less, servo, and gear motor types. A motor consists of a rotor and a permanent magnetic field stator. The magnetic field is maintained using either permanent magnets or electromagnetic windings. DC motors are most commonly used in variable speed and torque.  
 Motion and controls cover a wide range of components that in some way are used to generate and/or control motion. Areas within this category include bearings and bushings, clutches and brakes, controls and drives, drive components, encoders and resolves, Integrated motion control, limit switches, linear actuators, linear and rotary motion components, linear position sensing, [motors](http://dc-motors.globalspec.com/Industrial-Directory/motors) (both AC and DC motors), orientation position sensing, pneumatics and pneumatic components, positioning stages, slides and guides, power transmission (mechanical), seals, slip rings, solenoids, springs.   
  
 Motors are the devices that provide the actual speed and torque in a drive system.  This family includes AC motor types (single and multiphase motors, universal, servo motors, induction, synchronous, and gear motor) and DC motors (brush less, servo motor, and gear motor) as well as linear, stepper and air motors, and motor contactors and starters.

In any electric motor, operation is based on simple electromagnetism. A current-carrying conductor generates a magnetic field; when this is then placed in an external magnetic field, it will experience a force proportional to the current in the conductor, and to the strength of the external magnetic field. As you are well aware of from playing with magnets as a kid, opposite (North and South) polarities attract, while like polarities (North and North, South and South) repel. The internal configuration of a DC motor is designed to harness the magnetic interaction between a current-carrying conductor and an external magnetic field to generate rotational motion.

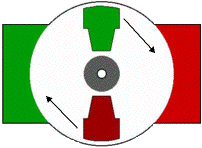
Let's start by looking at a simple 2-pole DC electric motor (here red represents a magnet or winding with a "North" polarization, while green represents a magnet or winding with a "South" polarization).



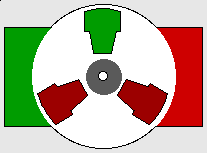
Every DC motor has six basic parts -- axle, rotor (a.k.a., armature), stator, commutator, field magnet(s), and brushes. In most common DC motors (and all that Beamers will see), the external magnetic field is produced by high-strength permanent magnets1. The stator is the stationary part of the motor -- this includes the motor casing, as well as two or more permanent magnet pole pieces. The rotor (together with the axle and attached commutator) rotates with respect to the stator. The rotor consists of windings (generally on a core), the windings being electrically connected to the commutator. The above diagram shows a common motor layout -- with the rotor inside the stator (field) magnets.

The geometry of the brushes, commutator contacts, and rotor windings are such that when power is applied, the polarities of the energized winding and the stator magnet(s) are misaligned, and the rotor will rotate until it is almost aligned with the stator's field magnets. As the rotor reaches alignment, the brushes move to the next commutator contacts, and energize the next winding. Given our example two-pole motor, the rotation reverses the direction of current through the rotor winding, leading to a "flip" of the rotor's magnetic field, and driving it to continue rotating.

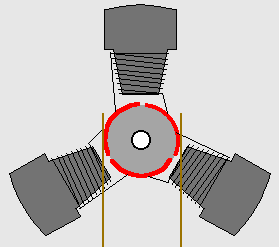
In real life, though, DC motors will always have more than two poles (three is a very common number). In particular, this avoids "dead spots" in the commutator. You can imagine how with our example two-pole motor, if the rotor is exactly at the middle of its rotation (perfectly aligned with the field magnets), it will get "stuck" there. Meanwhile, with a two-pole motor, there is a moment where the commutator shorts out the power supply (i.e., both brushes touch both commutator contacts simultaneously). This would be bad for the power supply, waste energy, and damage motor components as well. Yet another disadvantage of such a simple motor is that it would exhibit a high amount of torque” ripple" (the amount of torque it could produce is cyclic with the position of the rotor).



So since most small DC motors are of a three-pole design, let's tinker with the workings of one via an interactive animation (JavaScript required):



You'll notice a few things from this -- namely, one pole is fully energized at a time (but two others are "partially" energized). As each brush transitions from one commutator contact to the next, one coil's field will rapidly collapse, as the next coil's field will rapidly charge up (this occurs within a few microsecond). We'll see more about the effects of this later, but in the meantime you can see that this is a direct result of the coil windings' series wiring:



There's probably no better way to see how an average dc motor is put together, than by just opening one up. Unfortunately this is tedious work, as well as requiring the destruction of a perfectly good motor.

1. SOFTWARE DESCRIPTION

## 4.8 Programming:

Programming languages, frameworks, platforms, and environments used for ESP32 programming:

* [Arduino IDE](https://en.wikipedia.org/wiki/Arduino#IDE) with the ESP32 Arduino Core
* Espressif IoT Development Framework – Official Espressif development framework for ESP32.
* [Espruino](https://en.wikipedia.org/wiki/Espruino) – JavaScript SDK and firmware closely emulating Node.js.
* Lua RTOS for ESP32
* [Mongoose OS](https://en.wikipedia.org/wiki/Mongoose_OS) – An operating system for connected products on microcontrollers; programmable with JavaScript or C. A recommended platform by Espressif Systems, AWS IoT, and Google Cloud IoT.
* [mruby](https://en.wikipedia.org/wiki/Mruby) for the ESP32
* PlatformIO Ecosystem and IDE
* Pymakr IDE – IDE designed for use with Pycom devices; handles firmware upgrades and includes MicroPython [REPL](https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop) console.
* Simba Embedded Programming Platform
* Whitecat Ecosystem Blockly Based Web IDE
* [MicroPython](https://en.wikipedia.org/wiki/MicroPython)
* Zerynth – Python for IoT and microcontrollers, including the ESP32.

## Reception and use

Commercial and industrial use of ESP32:

### Use in commercial devices

* Alibaba Group's IoT LED wristband, used by participants at the group's 2017 annual gathering. Each wristband operated as a "pixel", receiving commands for coordinated LED light control, allowing formation of a "live and wireless screen.
* DingTalk's M1, a biometric attendance-tracking system.
* [LIFX](https://en.wikipedia.org/wiki/LIFX) Mini, a series of remotely controllable, LED based light bulbs.
* Pium, a home fragrance and aromatherapy device.
* HardKernel's [Odroid](https://en.wikipedia.org/wiki/ODROID" \o "ODROID) Go, an ESP32 based handheld gaming device kit made to commemorate Odroid's 10th anniversary.

### Use in industrial devices

* TECHBASE's Moduino X series X1 and X2 modules are ESP32-WROVER based computers for industrial automation and monitoring, supporting digital inputs/outputs, analog inputs, and various computer networking interfaces.

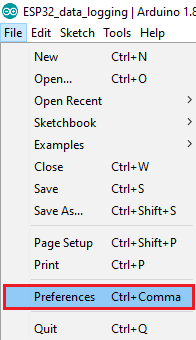
In this project we used Arduino IDE for programming ESP32. The language used for programming is Embedded C.

4.8.2 HOW TO INSTALL ESP32 BOARD IN ARDUINO IDE:

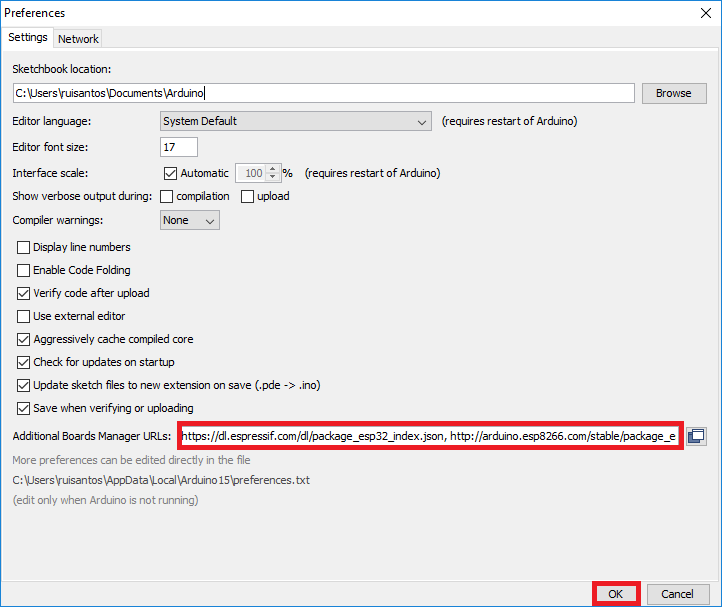
There’s an add-on for the Arduino IDE that allows us to program the ESP32 using the Arduino IDE and its programming language.

To install the ESP32 board in your Arduino IDE, follow these next instructions:

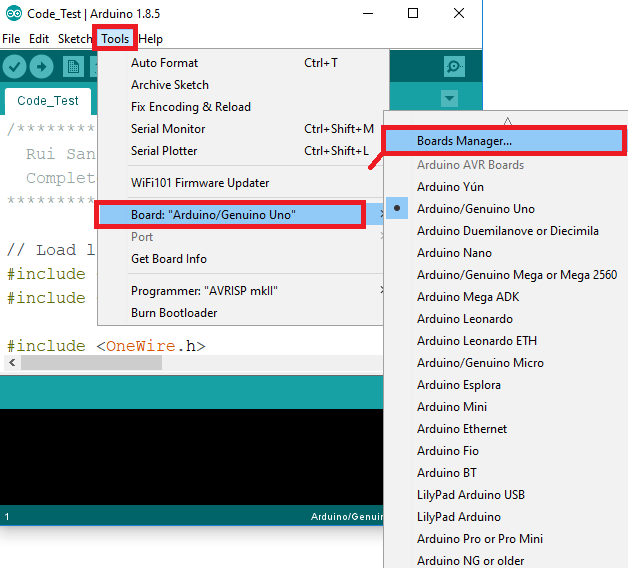
* 1. In your Arduino IDE, go to **File**> **Preferences**



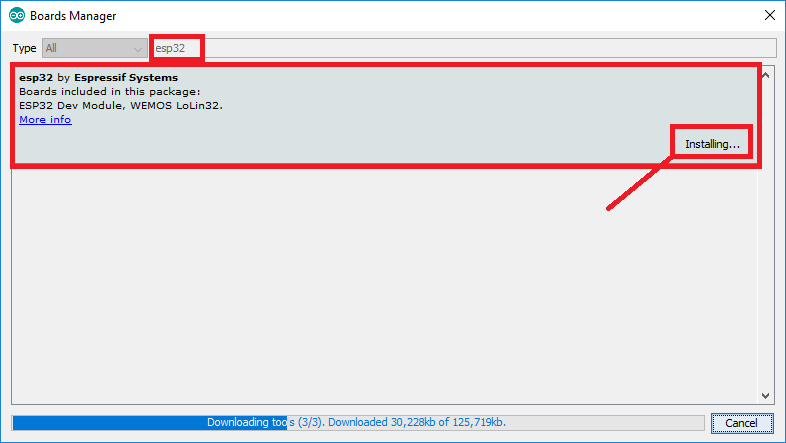
* 1. Enter **https://dl.espressif.com/dl/package\_esp32\_index.json** into the “Additional Board Manager URLs” field as shown in the figure below. Then, click the “OK” button:



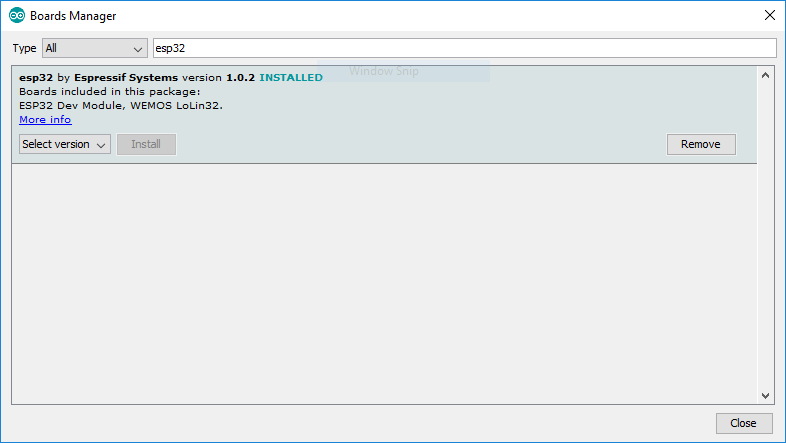
* Open the Boards Manager. Go to **Tools** > **Board** > **Boards Manager…**



* Search for **ESP32** and press install button for the “**ESP32 by Espressif Systems**“:



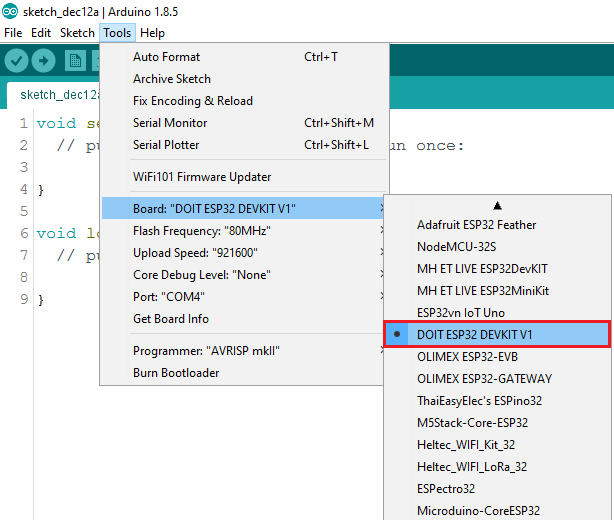
* That’s it. It should be installed after a few seconds.



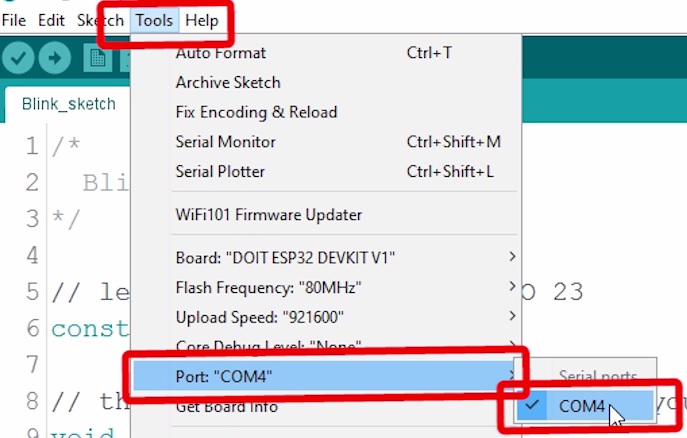
4.8.3 Testing The Installation:

Plug the ESP32 board to the computer. With your Arduino IDE open, follow these steps:

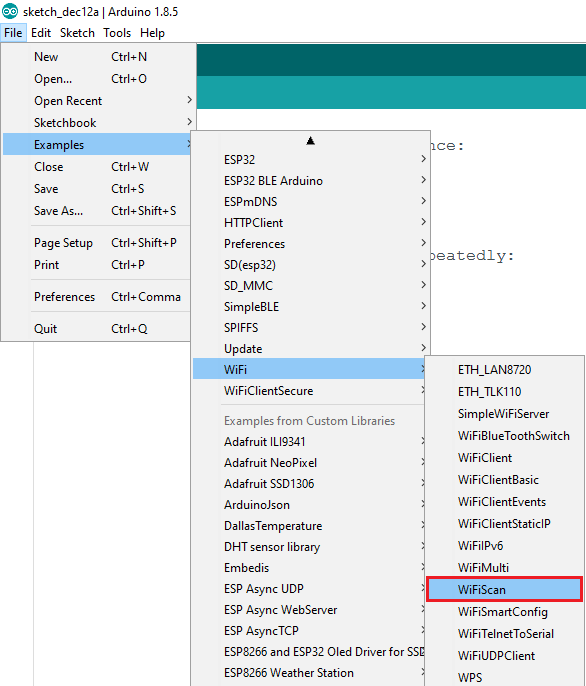
1. Select your Board in **Tools** > **Board** menu (in my case it’s the **DOIT ESP32 DEVKIT V1**)



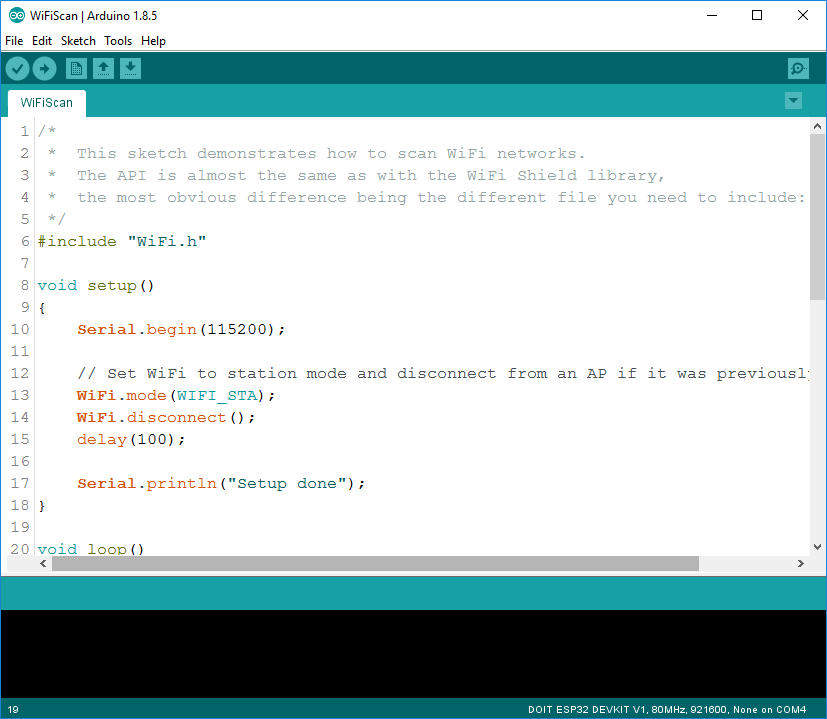
1. Select the Port (if you don’t see the COM Port in your Arduino IDE, you need to install the [CP210x USB to UART Bridge VCP Drivers](https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers)):



1. Open the following example under **File** > **Examples** > **WiFi (ESP32)** > **WiFiScan**



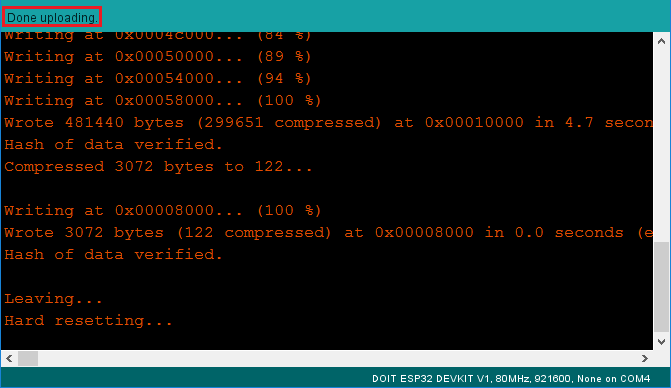
1. A new sketch opens in your Arduino IDE:



1. Press the **Upload** button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board.

Arduino IDE upload WiFiScan sketch to ESP32

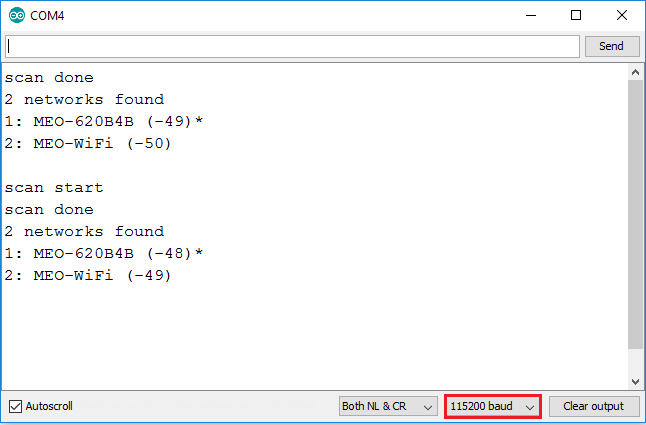
1. If everything went as expected, you should see a “**Done uploading.**” message.



1. Open the Arduino IDE Serial Monitor at a baud rate of 115200:

Open Arduino IDE Serial Monitor at baud rate 115200

1. Press the ESP32 on-board **Enable** button and you should see the networks available near the ESP32:

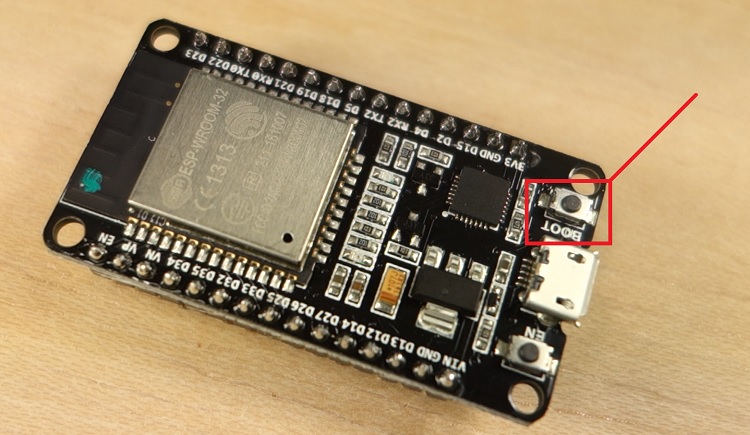


4.8.4 Trouble Shooting:

If we try to upload a new sketch to ESP32 and we get this error message “*A fatal error occurred: Failed to connect to ESP32: Timed out… Connecting…*“. It means that the ESP32 is not in flashing/uploading mode.

Having the right board name and COM per selected, follow these steps:

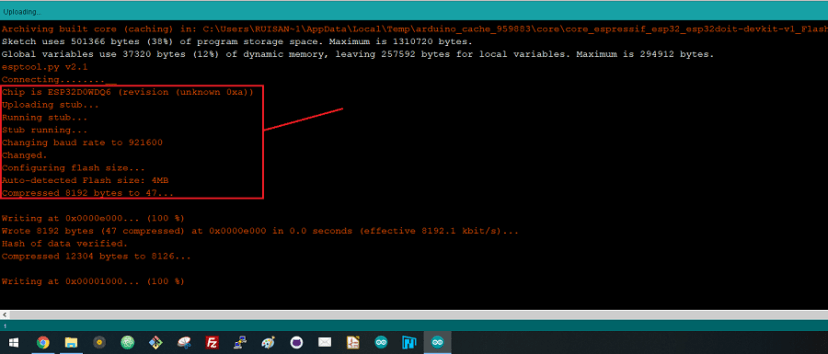
* 1. Hold-down the “**BOOT**” button in the ESP32 board



* 1. Press the “**Upload**” button in the Arduino IDE to upload the sketch:

Arduino IDE uploading new sketch to ESP32

* 1. After we see the  “**Connecting….**” message in your Arduino IDE, release the finger from the “**BOOT**” button:



* 1. After that, you should see the “**Done uploading**” message

That’s it. The ESP32 should have the new sketch running. Press the “**ENABLE**” button to restart the ESP32 and run the new uploaded sketch.

4.9 PROGRAMMING IN EMBEDDED C:

Earlier, many embedded applications were developed using assembly level programming. However, they did not provide portability. This disadvantage was overcome by the advent of various high-level languages like C, Pascal, and COBOL. However, it was the C language that got extensive acceptance for embedded systems, and it continues to do so. The C code written is more reliable, scalable, and portable; and in fact, much easier to understand. Embedded C Programming is the soul of the processor functioning inside each and every embedded system we come across in our daily life, such as mobile phones, washing machines, and digital cameras. Each processor is associated with embedded software. The first and foremost thing is the embedded software that decides to function of the embedded system. Embedded C language is most frequently used to program the microcontroller.

4.9.1 What is C language?

C language was developed by Dennis Ritchie in 1969. It is a collection of one or more functions, and every function is a collection of statements performing a specific task.  
C language is a middle-level language as it supports high-level applications and low-level applications. Before going into the details of embedded C programming, we should know about RAM memory organization.

The main features of the C language include the following.

1. C language is software designed with different keywords, data types, variables, constants, etc.
2. Embedded C is a generic term given to a programming language written in C, which is associated with particular hardware architecture.
3. Embedded C is an extension to the C language with header files.
4. These header files may change from controller to controller.
5. The microcontroller 8051 #include<reg51.h> is used.

4.9.2 What is an Embedded C Programming?

In every embedded system based projects, Embedded C programming plays a key role to make the microcontroller run & perform the preferred actions. At present, we normally utilize several electronic devices like mobile phones, washing machines, security systems, refrigerators, digital cameras, etc. The controlling of these embedded devices can be done with the help of an embedded C program. For example in a digital camera, if we press a camera button to capture a photo then the microcontroller will execute the required function to click the image as well as to store it.

Embedded C programming builds with a set of functions where every function is a set of statements that are utilized to execute some particular tasks. Both the embedded C and C languages are the same and implemented through some fundamental elements like a variable, character set, keywords, data types, declaration of variables, expressions, statements. All these elements play a key role while writing an embedded C program.

The embedded system designers must know about the hardware architecture to write programs. These programs play a prominent role in monitoring and controlling external devices. They also directly operate and use the internal architecture of the microcontroller, such as interrupt handling, timers, serial communication, and other available features.

4.9.3 Embedded System Programming:

As we discussed earlier, the designing of an embedded system can be done using Hardware & Software. For instance, in a simple embedded system, the processor is the main module that works like the heart of the system. Here a processor is nothing but a microprocessor, DSP, microcontroller, CPLD & FPGA. All these processors are programmable so that it defines the working of the device.

An Embedded system program allows the hardware to check the inputs & control outputs accordingly. In this procedure, the embedded program may have to control the internal architecture of the processor directly like Timers, Interrupt Handling, I/O Ports, serial communications interface, etc.

So embedded system programming is very important to the processor. There are different programming languages are available for embedded systems such as C, C++, assembly language, JAVA, JAVA script, visual basic, etc. So this programming language plays a key role while making an embedded system but choosing the language is very essential.

4.9.4 Steps to build an Embedded C programming:

There are different steps involved in designing an embedded c program like the following.

1. Comments

2. Directives of Processor

3. Configuration of Port

4. Global variables

5. Core Function/Main Function

6. Declaration of Variable

7. The logic of the Program

4.9.4.1 Comments:

In programming languages, comments are very essential to describe the program’s function. The code of the comments is non-executable but used to provide program documentation. To understand the function of the program, this will make a simple method to understand the function of the program. In embedded C, comments are available in two types namely single line and mainline comment.

In an embedded C programming language, we can place comments in our code which helps the reader to understand the code easily.

C=a+b; /\* add two variables whose value is stored in another variable C\*/

Single Line Comment:

Generally, for the programming languages, single-line comments are very useful to clarify a fraction of the program. These comments begin with a double slash (//) and it can be located anywhere within the programming language. By using this, the whole line can be ignored within a program.

Multi-Line Comment

Multi-line comments begin with a single slash (/) & an asterisk (/\*) in the programming languages which explains a block of code. These types of comments can be arranged anywhere within the programming language and mainly used to ignore a whole block of code within a program.

4.9.4.2 Directives of Processor:

The lines included within the program code are called preprocessor directives which can be followed through a hash symbol (#). These lines are the pre-processor directives but not programmed statements.  
The code can be examined through a preprocessor before real code compilation starts & resolves these directives before generating a code through regular statements. There are several special preprocessor directives are available although two directives are extremely helpful within the programming language

like the following.

#include  
#include<reg51.h>  
Sbit LED = P2^3;  
Main();  
{  
LED = 0x0ff  
Delay();  
LED=0x00;  
}  
#define  
#include<reg51.h>  
#define LED P0  
Main();  
{  
LED = 0x0ff  
Delay();  
LED=0x00;  
}

In the above program, the #include directive is generally used to comprise standard libraries like study and. h is used to allow I/O functions using the library of ‘C’. The #define directive usually used to describe the series of variables & allocates the values by executing the process within a particular instruction like macros.

Configuration of Port:

The microcontroller includes several ports where every port has different pins. These pins can be used for controlling the interfacing devices. The declaration of these pins can be done within a program with the help of keywords. The keywords in the embedded c program are standard as well as predefined like a bit, sbit, SFR which are used to state the bits & single pin within a program.

There are certain words that are reserved for doing specific tasks. These words are known as keywords. They are standard and predefined in the Embedded C. Keywords are always written in lowercase. These keywords must be defined before writing the main program. The main functions of the keywords include the following.

#include< >  
Sbit a = P 2^2;  
SFR 0x00 = PoRT0;  
Bit C;  
main()  
{  
……………..  
……………..  
}

Sbit:

This is one kind of data type, used to access a single bit within an SFR register.

The syntax for this data type is : sbit variable name = SFR bit ;

Example: sbit a=P2^1;

If we assign p2.1 as ‘a’ variable, then we can use ‘a’ instead of p2.1 anywhere in the program, which reduces the complexity of the program.

Bit:

This type of data type is mainly used for allowing the bit addressable memory of random access memory like 20h to 2fh.

The syntax of this data type is : name of bit variable;

Example: bit c;

It is a bit series setting within a small data region that is mainly used with the help of a program to memorize something.

SFR:

This kind of data type is used to obtain the peripheral ports of the SFR register through an additional name. So, the declaration of all the SFR registers can be done in capital letters.

The syntax of this data type is: SFR variable name = SFR address for SFR register;

Example: SFR port0 = 0×80;

If we allocate 0×80 like ‘port0’, after that we can utilize 0×80 in place of port0 wherever in the programming language to decrease the difficulty of the program.

SFR Register:

The SFR stands for Special Function Register. In 8051 microcontroller, it includes the RAM memory with 256 bytes, which is divided into two main elements: the first element of 128 bytes is mainly utilized for storing the data whereas the other element of 128 bytes is mainly utilized to SFR registers. All the peripheral devices such as timers, counters & I/O ports are stored within the SFR register & every element includes a single address.

4.9.4.3 Global Variables:

When the variable is declared before the key function is known as the global variable. This variable can be allowed on any function within the program. The global variable’s life span mainly depends on the programming until it reaches an end.

#include<reg51.h>  
Unsigned int a, c =10;  
Main()  
{  
……………  
…………..  
}

4.9.4.4 Core Function / Main Function

The main function is a central part while executing any program and it begins with the main function simply. Each program utilizes simply one major function since if the program includes above one major function, next the compiler will be confused in begin the execution of the program.

#include<reg51.h>  
Main()  
{  
……………  
…………..  
}

4.9.4.5 Declaration of Variable:

The name like the variable is used for storing the values but this variable should be first declared before utilized within the program. The variable declaration states its name as well as a data type. Here, the data type is nothing but the representation of storage data. In embedded C programming, it uses four fundamental data types like integer, float, character for storing the data within the memory. The data type size, as well as range, can be defined depending on the compiler.

The data type refers to an extensive system for declaring variables of different types like integer, character, float, etc. The embedded C software uses four data types that are used to store data in memory.

The ‘char’ is used to store any single character; ‘int’ is used to store integer value, and ‘float’ is used to store any precision floating-point value. The size and range of different data types on a 32-bit machine are given in the following table. The size and range may vary on machines with different word sizes.

The char/signed char data type size is 1 byte and its range is from -128 to +128

The unsigned char data type size is 1 byte and its range is from 0 to 255

Int/signed int data type size is 2 byte and its range is from -32768 to 32767

Unsigned int data type size is 2 byte and its range is from 0 to 65535

Main();  
{  
Unsigned int a,b,c;  
}

The Structure of an Embedded C Program is shown below.

comments

preprocessor directives

global variables

main() function

{

local variables

statements

…………..

…………..

}

fun(1)

{

local variables

statements

…………..

…………..

}

4.9.4.6 The logic of the Program:

The logic of the program is a plan of the lane that appears in the theory behind & predictable outputs of actions of the program. It explains the statement otherwise theory regarding why the embedded program will work and shows the recognized effects of actions otherwise resources.

Main  
{  
LED = 0x0f;  
delay(100);  
LED = 0x00;  
delay(100);  
}

4.9.5 Main Factors of Embedded C Program:

The main factors to be considered while choosing the programming language for developing an embedded system include the following.

Program Size:

Every programming language occupies some memory where embedded processor like microcontroller includes an extremely less amount of random access memory.

Speed of the Program:

The programming language should be very fast, so should run as quickly as possible. The speed of embedded hardware should not be reduced because of the slow-running software.

Portability:

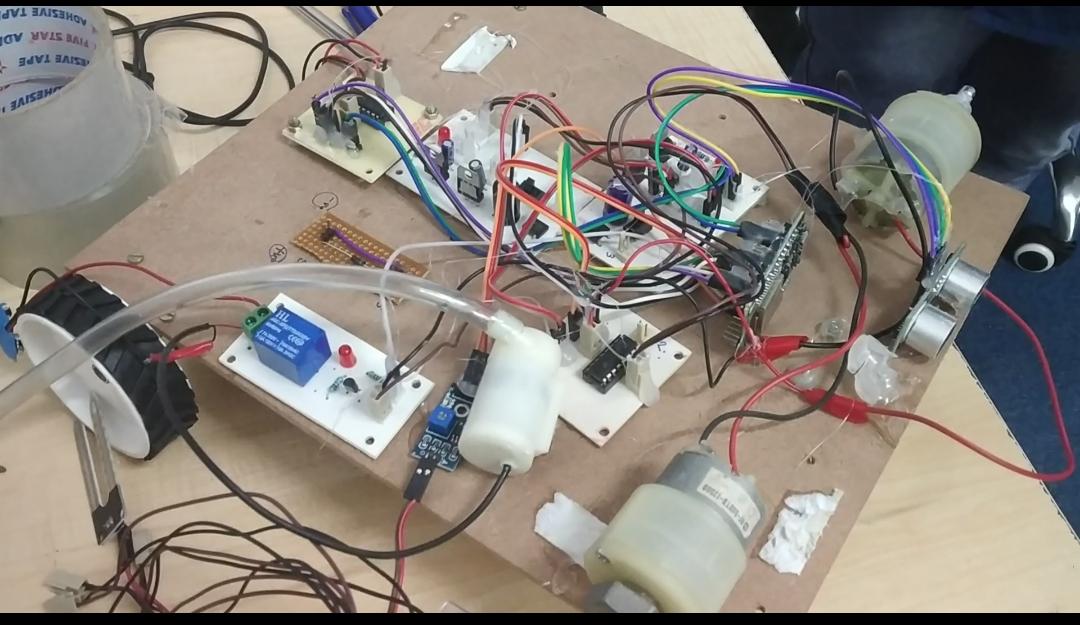
For the different embedded processors, the compilation of similar programs can be done.

1. Simple Implementation
2. Simple Maintenance
3. Readability

CHAPTER-5: RESULTS AND DISCUSSIONS**.**

5.1 Result:

The project ‘**SOLAR POWERED MULTIPURPOSE ROBOT USING ESP32**’ is a great helping device which can be used in agriculture. The basic aim of this project is to eliminate the use of bulls and human labour, and reduces the dependency of farmers on diesel vehicles and animals.This device works with the solar power and*ESP32.* When powering the ESP32 using solar panels or batteries, it is important to save power.

The designed robot gets power from solar panel which converts sunlight into electricity, where solar cells last a longer time and low running costs. The electrical energy is given to this device in order to work properly.

This robot will perform the seed sowing and ploughing. This will be very helpful in agricultural fields and sectors.

The main thing is ESP32, this is low cost with integrated wifi. This is basically created and developed by Espressif Systems, in China.

In this advance world, this kind of robot is used in many fields such as defence, medical fields and industries and so on.

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

6.1 Conclusion:

In agriculture, by using this we can reduce the reduce the farmers effect and time. This way we can overcome the labour work. In this plouging, sensing dry land and giving signal operations can be done. This is very simple and easy to operate.

6.2 Future Scope:

We use ESP32 which is microcontroller, where it can be integrated with hybrid microcontrollers. At the same time by using solar energy environmental pollution can be reduced.

This can be controlled based on Internet Of Things (IOT) such as GPS and wifi controllable devices.

**REFERENCES**

[1] S. Umarkar and A. Karwankar, “Automated Seed Sowing Agribot using Arduino” in IEEE Conference on Communication and Signal Processing, April 2016, pp. 1379-1383.

[2] M.D.I. Sujon, R. Nasir, M.M.I. Habib, M.I.Nomaan J. Baidya and M.R. Islam “Agribot: Arduino Controlled Autonomous Multipurpose Farm Machinery Robot for Small to medium scale cultivation” in IEEE conference on intelligent autonomous systems, March 2018,pp. 155-159.

[3] H. Pota, R. Eaton, J. Katapriya and S.D. Pathirana, “Agricultural robotics: A streamlined approach to realisation autonomous farming,” in IEEE conference on industrial and information systems, 2007,pp. 85-90.

[4] S. Kareenulla, E. Prajwal, B. Sujeshkumar, B. Mahesh, and V Reddy, “GPS based Autonomous Agriculture Robot”, in IEEE International conference on design innovations for 3Cs compute communicate control, 2018,pp. 100-105.

[5] HC-05- Bluetooth Module, Available: <https://components101.com/wireless/HD-05-bluetooth-module>”, accessed on September 2018.

[6] P.V. Santhi, N. kapileswar, V.K.R.Chenchela and C.H.V.S Prasad, “Sensor and vision based autonomous Agribot for sowing seeds”, in IEEE International Conference on energy communications, data analysis and soft computing(ICECDS), 2017, pp. 242-245.

[7] P.V.S.Jayakrisna, M.S.Reddy, N.J.Sai, N.Susheel and K.P.Peeyush, “Autonomous seed sowing agricultural robot,” in IEEE Conference on advances in computing, communications and informatics (ICACCI), 2018, pp.2332-2336.

[8] N.S.Naik, V.V.Shete and S.R.Danve, “Precision agriculture robot for seeding function,” in IEEEE International conference on inventive computation technologies (ICICT), 2016, pp. 1-3.

[9] M.U.Hassan, M.Ullah and J.Iqbal, “Towards autonomy in agriculture: Design and prototyping of a robotic vehicle with seed selector,” in IEEEE International conference on robotics and artificial intelligence (ICRAI), 2016, pp.37-44.

[10] S.Konam, N.Srinivasa Rao and K.Mohan Krishna, “Design encompassing mechanical aspects of ROTAAI: Robot to aid agricultural industry,” in IEEEE International conference on soft computing and machine intelligence, 2014, pp.15-19.

[11] C.M.Barber, R.J.Shucksmith, B.M.Donald and B.C.Wunsche, “Sketch-based robot programming,” in IEEEE International conference of image and vision computing newzealand, 2010, pp. 1-8.

[12] A.Srinivastava, S.Vijay, A.Negi, P.Shrivastva, A.Singh, “DTMF based intelligent farming robot vehicle: An ease to farmers, “in IEEEE International conference on embedded systems, 2014, pp. 206-210.

**APPENDIX**

Program Code:

*#define TRIGGER 2*

*#define ECHO 4*

*#define BLYNK\_PRINT Serial*

*#include <WiFi.h>*

*#include <WiFiClient.h>*

*#include <BlynkSimpleEsp32.h>*

*char auth[] = "WzqKk3m3GoZRWhc2wMCJUz0wT0WjtjV6"; // You should get Auth Token in the Blynk App.*

*char ssid[] = "project12"; // Your Wi-Fi Credentials*

*char pass[] = "project123456";*

*WidgetLCD lcd1(V0);*

*BLYNK\_WRITE(V1)*

*{*

*int p = param.asInt(); // assigning incoming value from pin V1 to a variable*

*if(p==1)*

*{*

*digitalWrite(23,HIGH);*

*digitalWrite(22,LOW);*

*digitalWrite(21,HIGH);*

*digitalWrite(19,LOW);*

*}*

*}*

*BLYNK\_WRITE(V2)*

*{*

*int p = param.asInt(); // assigning incoming value from pin V1 to a variable*

*if(p==1)*

*{*

*digitalWrite(22,HIGH);*

*digitalWrite(23,LOW);*

*digitalWrite(19,HIGH);*

*digitalWrite(21,LOW);*

*}*

*}*

*BLYNK\_WRITE(V3)*

*{*

*int p = param.asInt(); // assigning incoming value from pin V1 to a variable*

*if(p==1)*

*{*

*digitalWrite(23,HIGH);*

*digitalWrite(22,LOW);*

*digitalWrite(21,LOW);*

*digitalWrite(19,LOW);*

*}*

*}*

*BLYNK\_WRITE(V4)*

*{*

*int p = param.asInt(); // assigning incoming value from pin V1 to a variable*

*if(p==1)*

*{*

*digitalWrite(23,LOW);*

*digitalWrite(22,LOW);*

*digitalWrite(21,HIGH);*

*digitalWrite(19,LOW);*

*}*

*}*

*BLYNK\_WRITE(V5)*

*{*

*int p = param.asInt(); // assigning incoming value from pin V1 to a variable*

*if(p==1)*

*{*

*digitalWrite(23,LOW);*

*digitalWrite(22,LOW);*

*digitalWrite(21,LOW);*

*digitalWrite(19,LOW);*

*}*

*}*

*BLYNK\_WRITE(V6)*

*{*

*int p = param.asInt(); // assigning incoming value from pin V1 to a variable*

*if(p==1)*

*{*

*digitalWrite(18,HIGH);*

*digitalWrite(12,LOW);*

*}*

*if(p==0)*

*{*

*digitalWrite(18,LOW);*

*digitalWrite(12,LOW);*

*}*

*}*

*BLYNK\_WRITE(V7)*

*{*

*int p = param.asInt(); // assigning incoming value from pin V1 to a variable*

*if(p==1)*

*{*

*digitalWrite(13,HIGH);*

*digitalWrite(14,LOW);*

*delay(1000);*

*digitalWrite(13,LOW);*

*digitalWrite(14,LOW);*

*}*

*if(p==0)*

*{*

*digitalWrite(14,HIGH);*

*digitalWrite(13,LOW);*

*delay(1000);*

*digitalWrite(13,LOW);*

*digitalWrite(14,LOW);*

*}*

*}*

*void setup() {*

*pinMode(TRIGGER, OUTPUT);*

*pinMode(ECHO, INPUT);*

*pinMode(23,OUTPUT);*

*pinMode(22,OUTPUT);*

*pinMode(21,OUTPUT);*

*pinMode(19,OUTPUT);*

*pinMode(18,OUTPUT);*

*pinMode(12,OUTPUT);*

*pinMode(13,OUTPUT);*

*pinMode(14,OUTPUT);*

*pinMode(33,OUTPUT);*

*pinMode(32,INPUT);*

*digitalWrite(23,LOW);*

*digitalWrite(22,LOW);*

*digitalWrite(21,LOW);*

*digitalWrite(19,LOW);*

*digitalWrite(18,LOW);*

*digitalWrite(12,LOW);*

*digitalWrite(13,LOW);*

*digitalWrite(14,LOW);*

*digitalWrite(33,LOW);*

*Serial.begin(115200);*

*delay(10);*

*Serial.print("Connecting to ");*

*Serial.println(ssid);*

*WiFi.begin(ssid, pass);*

*int wifi\_ctr = 0;*

*while (WiFi.status() != WL\_CONNECTED) {*

*delay(500);*

*Serial.print(".");*

*}*

*Serial.println("WiFi connected");*

*Blynk.begin(auth, ssid, pass);*

*lcd1.clear();*

*}*

*void loop(){*

*Blynk.run();*

*Blynk.run();*

*if(digitalRead(32)==1)*

*{*

*digitalWrite(33,HIGH);*

*lcd1.print(0,0,"NW MOTOR ON ");*

*}*

*else*

*{*

*digitalWrite(33,LOW);*

*lcd1.print(0,0,"WP MOTOR OFF");*

*}*

*long duration, d;*

*digitalWrite(TRIGGER, LOW);*

*delayMicroseconds(2);*

*digitalWrite(TRIGGER, HIGH);*

*delayMicroseconds(10);*

*digitalWrite(TRIGGER, LOW);*

*duration = pulseIn(ECHO, HIGH);*

*d = (duration/2)\*0.034 ;*

*Serial.println(d);*

*if(d<20)*

*{*

*digitalWrite(23,LOW);*

*digitalWrite(22,LOW);*

*digitalWrite(21,LOW);*

*digitalWrite(19,LOW);*

*lcd1.print(4,1,"OBSTACLE ");*

*}*

*else*

*{*

*lcd1.print(4,1,"NO OBSTACLE");*

*}*

*delay(1000);*

*}*