

LIST OF CONTENTS	PAGE NO
ACKNOWLEDGEMENT.....	I
DECLARATION.....	II
LIST OF CONTENTS.....	III
LIST OF FIGURES.....	IV
ABSTRACT.....	V
CHAPTER 1: INTRODUCTION TO EMBEDDED SYSTEMS.....	1
1.1 APPLICATIONS OF EMBEDDED SYSTEM.....	2
1.2 CHARACTERISTICS OF EMBEDDED SYSTEM.....	3
1.3 PROCESORS IN EMBEDDED SYSTEMS.....	4
1.4 DEBUGGING IN EMBEDDED SYSTEMS.....	5
1.5 RELIABILITY.....	6
1.6 TRACING.....	7
CHAPTER 2: THERMAL MANAGEMENT SYSTEM FOR EV BATTERIES.....	8
INTRODUCTION.....	8
ADVANTAGES.....	9
APPLICATIONS.....	11
BLOCK DIAGRAM.....	13
CHAPTER 3: SOFTWARE REQUIREMENTS.....	15
3.1 INTRODUCTION TO ARDUINO IDE.....	15
3.1.1 THE SEMICOLON.....	17
3.1.2 THE DOUBLE BACKSLASH FOR SINGLE LINE COMMENTS //.....	18
3.1.3 THE CURLY BRACES.....	18

3.1.4 FUNCTIONS	18
3.1.5 VOID SETUP	19
3.1.6 VOID LOOP.....	20
3.2 INTRODUCTION ARDUINO LIBRARIES.....	21
3.2.1 HOW TO INSTALL A LIBRARY.....	21
3.3 HOW TO CONNECT ARDUINO BOARD.....	23
3.4 HOW TO UPLOAD A PROGRAM.....	24
CHAPTER 4: HARDWARE REQUIREMENTS.....	25
4.1 INTRODUCTIUON TO ARDUINO UNO.....	25
4.1.1 WHY ARDUINO.....	26
4.1.2 ADVANTAGES OF ARDUINO.....	27
4.1.3 FEATURES OF ARDUINO UNO.....	28
4.2 INTODUCTION TO L293D MOTOR DRIVER.....	32
4.2.1 PIN DESCRIPTION.....	32
4.2.2 CONTROLLING DC MOTOR.....	33
4.3 INTRODUCTION TO 16X2 LCD MODULE.....	29
4.3.1 LCD PIN DESCRIPTION.....	31
4.4 INTRODUCTION TO 12V DC FAN.....	34
4.4.1PHYSICAL CHARACTERISTICS.....	34
4.4.2 DIMENTIONS.....	35
4.4.3 ROTATIONAL SPEED.....	35
4.5 DHT 11 SENSOR.....	36
4.5.1 TECHNICAL DETAILS.....	37
4.6 INTODUCTION TO VOLTAGE SENSOR.....	37

4.7 INTRODUCTION TO BUZZER.....	38
4.7.1 PIN CONFIGURATION.....	38
4.8 INTRODUCTION TO 18650 BATTERY.....	39
CHAPTER 5: PROGRAM CODE.....	40
CHAPTER 6: RESULTS.....	43
CHAPTER 7: CONCLUSION.....	44

CHAPTER 1

INTRODUCTION TO EMBEDDED SYSTEMS

Embedded Technology is now in its prime and the wealth of knowledge available is mind blowing. However, most embedded systems engineers have a common complaint. There are no comprehensive resources available over the internet which deal with the various design and implementation issues of this technology. Intellectual property regulations of many corporations are partly to blame for this and also the tendency to keep technical know-how within a restricted group of researchers.

An embedded computer is frequently a computer that is implemented for a particular purpose. In contrast, an average PC computer usually serves a number of purposes: checking email, surfing the internet, listening to music, word processing, etc... However, embedded systems usually only have a single task, or a very small number of related tasks that they are programmed to perform.

Every home has several examples of embedded computers. Any appliance that has a digital clock, for instance, has a small embedded micro-controller that performs no other task than to display the clock. Modern cars have embedded computers onboard that control such things as ignition timing and anti-lock brakes using input from a number of different sensors.

Embedded computers rarely have a generic interface, however. Even if embedded systems have a keypad and an LCD display, they are rarely capable of using many different types of input or output. An example of an embedded system with I/O capability is a security alarm with an LCD status display, and a keypad for entering a password.

An embedded system can be defined as a control system or computer system designed to perform a specific task. Common examples of embedded systems include MP3 players, navigation systems on aircraft and intruder alarm systems. An embedded system can also be defined as a single purpose computer.

Most embedded systems are time critical applications meaning that the embedded system is working in an environment where timing is very important: the results of an operation are only relevant if they take place in a specific time frame. An autopilot in an aircraft is a time critical embedded system.

1.1 APPLICATIONS OF EMBEDDED SYSTEM

Embedded systems are commonly found in consumer, cooking, industrial, automotive, medical, commercial and military applications.

Telecommunications systems employ numerous embedded systems from telephone switches for the network to cell phones at the end user. Computer networking uses dedicated routers and network bridges to route data.

Consumer electronics include MP3 players, mobile phones, videogame consoles, digital cameras, GPS receivers, and printers. Household appliances, such as microwave ovens, washing machines and dishwashers, include embedded systems to provide flexibility, efficiency and features. Advanced HVAC systems use networked thermostats to more accurately and efficiently control temperature that can change by time of day and season. Home automation uses wired- and wireless-networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.

Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Various electric motors brushless DC motors, induction motors and DC motors use electric/electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles increasingly use embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems include anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.

Medical equipment uses embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging (PET, SPECT, CT, and MRI) for non-invasive internal inspections. Embedded systems within medical equipment are often powered by industrial computers.

Embedded systems are used in transportation, fire safety, safety and security, medical applications and life critical systems, as these systems can be isolated from hacking and thus, be more reliable. For fire safety, the systems can be designed to have greater ability to handle higher temperatures and continue to operate. In dealing with security, the embedded systems can be self-sufficient and be able to deal with cut electrical and communication systems.

1.2 CHARACTERISTICS OF EMBEDDED SYSTEM

Embedded systems are designed to do some specific task rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

Embedded systems are not always standalone devices. Many embedded systems consist of small parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music. Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

Embedded systems range from no user interface at all, in systems dedicated only to one task, to complex graphical user interfaces that resemble modern computer desktop operating systems. Simple embedded devices use buttons, LEDs, graphic or character LCDs (HD44780 LCD for example) with a simple menu system.

More sophisticated devices which use a graphical screen with touch sensing or screen-edge buttons provide flexibility while minimizing space used: the meaning of the buttons can change with the screen, and selection involves the natural behavior of pointing at what is desired. Handheld systems often have a screen with a "joystick button" for a pointing device.

Some systems provide user interface remotely with the help of a serial (e.g. RS232, USB, I²C, etc.) or network (e.g. Ethernet) connection. This approach gives several advantages: extends the capabilities of embedded system, avoids the cost of a display, simplifies BSP and allows one to build a rich user interface on the PC. A good example of this is the combination of an embedded web server running on an embedded device (such as an IP camera) or a network router. The user interface is displayed in a web browser on a PC connected to the device, therefore needing no software to be installed.

1.3 PROCESSORS IN EMBEDDED SYSTEMS

Embedded processors can be broken into two broad categories. Ordinary microprocessors Embedded processors can be broken into two broad categories. Ordinary microprocessors (μ P) use separate integrated circuits for memory and peripherals. Microcontrollers (μ C) have onchip peripherals, thus reducing power consumption, size and cost. In contrast to the personal computer market, many different basic CPU architectures are used, since software is

custom-developed for an application and is not a commodity product installed by the end user. Both Von Neumann as well as various degrees of Harvard architectures are used. RISC as well as non-RISC processors are found. Word lengths vary from 4-bit to 64-bits and beyond, although the most typical remain 8/16-bit. Most architectures come in a large number of different variants and shapes, many of which are also manufactured by several different companies.

Numerous microcontrollers have been developed for embedded systems use. General-purpose microprocessors are also used in embedded systems, but generally require more support circuitry than microcontrollers.

(μ P) use separate integrated circuits for memory and peripherals. Microcontrollers (μ C) have on-chip peripherals, thus reducing power consumption, size and cost. In contrast to the personal computer market, many different basic CPU architectures are used, since software is custom-developed for an application and is not a commodity product installed by the end user. Both Von Neumann as well as various degrees of Harvard architectures are used. RISC as well as non-RISC processors are found. Word lengths vary from 4-bit to 64-bits and beyond, although the most typical remain 8/16-bit. Most architectures come in a large number of different variants and shapes, many of which are also manufactured by several different companies.

Numerous microcontrollers have been developed for embedded systems use. General-purpose microprocessors are also used in embedded systems, but generally require more support circuitry than microcontrollers.

1.4 DEBUGGING IN EMBEDDED SYSTEMS

Embedded debugging may be performed at different levels, depending on the facilities available. The different metrics that characterize the different forms of embedded debugging are: does it slow down the main application, how close is the debugged system or application to the actual system or application, how expressive are the triggers that I can set for debugging (e.g., I want to inspect the memory when a particular program counter value is reached), and what can I inspect in the debugging process (such as, only memory, or memory and registers, etc.).

From simplest to most sophisticated 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 multicore 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 (ICE) 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. The downsides are expense and slow operation, in some cases up to 100 times slower than the final system.

For SoC designs, the typical approach is to verify and debug the design on an FPGA prototype board. Tools such as Certus^[11] are used to insert probes in the FPGA RTL that make signals available for observation. This is used to debug hardware, firmware and software interactions across multiple FPGA with capabilities similar to a logic analyzer.

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 HLL source-code, assembly code or mixture of both.

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, and 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.

1.5 RELIABILITY

Embedded systems often reside in machines that are expected to run continuously for years without errors, and in some cases recover by themselves 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 backups are selected by an operator. Examples include aircraft navigation, reactor control systems, safety-critical chemical factory controls, train signals.
- 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^[12] ensures a highly secure & reliable system environment
- A hypervisor designed for embedded systems, 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

1.6 TRACING

Real-time operating systems (RTOS) often supports tracing of operating system events. A graphical view is presented by a host PC tool, based on a recording of the system behavior. The trace recording can be performed in software, by the RTOS, or by special tracing hardware. RTOS tracing allows developers to understand timing and performance issues of the software system and gives a good understanding of the high-level system behaviors. Commercial tools like RTX C Quadros or IAR Systems exists.

CHAPTER 2

Electric Vehicle with Thermal Management and cooling System

2.1 INTRODUCTION

In recent years, the automotive industry has witnessed a significant shift towards electric vehicles (EVs) as a cleaner and more sustainable mode of transportation. Electric vehicle batteries serve as the heart of these eco-friendly vehicles, powering them with energy stored in lithium-ion cells. However, efficient and reliable thermal management is crucial for the optimal performance and longevity of these batteries. The efficient management of temperature within an electric vehicle's battery pack is a paramount concern for automakers and engineers. This is due to the fact that temperature greatly impacts the performance, safety, and overall lifespan of the battery. As the demand for electric vehicles continues to grow, the need for advanced thermal management systems becomes increasingly imperative. This introduction explores the significance of thermal management systems for electric vehicle batteries, highlighting their impact on battery performance, safety, and the overall sustainability of EVs.

2.2 ADVANTAGES

Thermal management systems for electric vehicle batteries offer a multitude of advantages that are critical for the performance, safety, and longevity of these vehicles. Here are some of the key advantages:

1. **Enhanced Battery Life:** Effective thermal management systems help maintain the battery's operating temperature within an optimal range. This prevents extreme temperature fluctuations that can accelerate degradation, thus extending the lifespan of the battery pack.
2. **Improved Performance:** Maintaining the battery within the ideal temperature range ensures consistent and efficient energy output. This results in better acceleration, regenerative braking, and overall vehicle performance.

3. **Safety:** Proper temperature control minimizes the risk of thermal runaway, a dangerous situation where the battery overheats and may catch fire. Thermal management systems can quickly dissipate excess heat, reducing the chances of battery-related safety incidents.
4. **Fast Charging:** Thermal management systems enable faster charging by preventing the battery from overheating during high-power charging sessions. This reduces charging time and enhances the convenience of electric vehicles.
5. **Increased Range:** By optimizing the battery's temperature, thermal management systems can help extend the driving range of electric vehicles. Battery efficiency is higher when operating at the right temperature, allowing for longer trips on a single charge.
6. **Energy Efficiency:** Maintaining a consistent battery temperature reduces energy waste. This is achieved by minimizing the need for active heating or cooling, ensuring that more of the battery's energy goes toward propelling the vehicle.
7. **Climate Adaptability:** Electric vehicles with effective thermal management systems can operate in a wide range of climates, from extremely cold to hot environments. This adaptability makes EVs suitable for use in various regions.
8. **User Comfort:** Electric vehicle cabins can also benefit from thermal management systems, as they allow for more efficient climate control. This improves the comfort of passengers and reduces the reliance on heating or cooling systems that may otherwise drain the battery.
9. **Maintenance Cost Reduction:** Longer battery life and reduced wear and tear due to temperature extremes mean lower maintenance costs for EV owners. This makes electric vehicles more cost-effective over the long term.

10. Environmental Benefits: Electric vehicles are already more environmentally friendly than their internal combustion engine counterparts. When combined with efficient thermal management, these benefits are amplified. Lower energy waste, longer battery life, and safer operation contribute to a reduced carbon footprint.

11. Regulatory Compliance: As governments worldwide introduce stricter emission standards and safety regulations, electric vehicles equipped with robust thermal management systems are more likely to meet these requirements, ensuring the long-term viability of electric mobility.

In summary, thermal management systems are crucial for optimizing the performance, safety, and overall efficiency of electric vehicle batteries. They are an essential component in making electric vehicles a practical and attractive choice for consumers, promoting sustainability, and addressing the challenges of a changing automotive landscape.

2.3 APPLICATIONS

Thermal management systems for electric vehicle (EV) batteries find a wide range of applications to ensure the optimal performance and safety of the battery packs. These applications include:

1. **Battery Cooling:** One of the primary functions of thermal management systems is to cool the battery pack during operation. This prevents the batteries from overheating, which can lead to performance degradation and safety risks. Cooling may be achieved through liquid cooling or air cooling systems.
2. **Battery Heating:** In cold climates, thermal management systems can provide battery heating to maintain optimal operating temperatures. This is essential for preventing reduced efficiency and range in low-temperature conditions.

3. **Fast Charging:** During high-power charging, batteries can generate significant heat. Thermal management systems dissipate this heat to prevent over-temperature conditions and ensure safe, rapid charging.
4. **Battery Preconditioning:** In preparation for extreme temperatures, some thermal management systems pre-condition the battery by heating or cooling it before the vehicle is in use. This ensures that the battery is at the right temperature for optimal performance.
5. **Regenerative Braking:** Regenerative braking generates heat in the battery as energy is recovered during deceleration. Thermal management systems help dissipate this heat and prevent overheating.
6. **Environmental Adaptability:** Electric vehicles with thermal management systems can operate in a wide range of climates, from extremely cold to hot environments, thanks to their ability to regulate battery temperature effectively.
7. **Safety Systems:** Thermal management systems play a crucial role in safety by preventing thermal runaway and fires due to battery overheating. They can quickly detect and mitigate temperature-related issues.
8. **Energy Efficiency:** Maintaining the battery at an optimal temperature range improves energy efficiency. This means that a higher percentage of the energy stored in the battery is available for vehicle propulsion, increasing overall efficiency.
9. **User Comfort:** Some thermal management systems extend their benefits to the vehicle's cabin, ensuring that the interior remains at a comfortable temperature without relying heavily on the battery for heating or cooling.

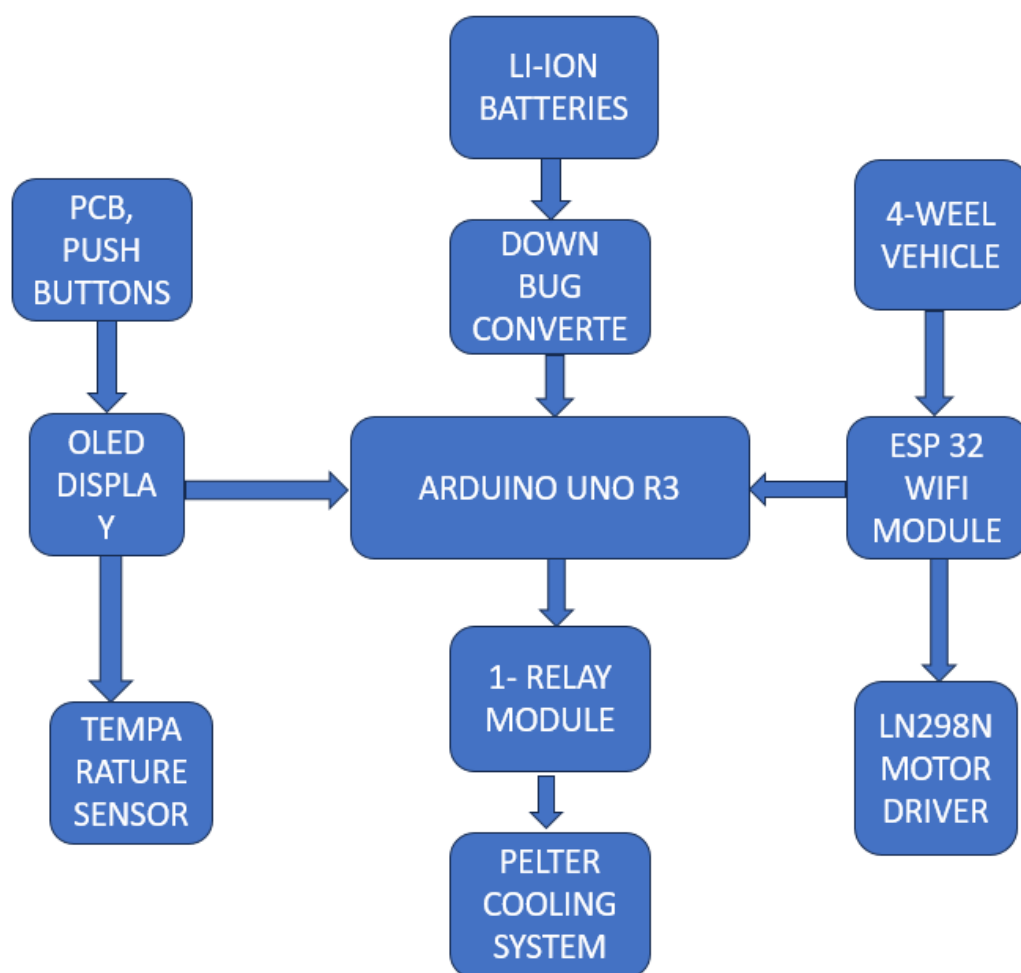
10. Maintenance Optimization: Effective thermal management systems help extend the life of the battery pack by reducing wear and tear due to extreme temperatures. This, in turn, reduces maintenance costs and enhances the overall ownership experience for EV owners.

11. Optimizing Battery Usage: Thermal management systems can enable intelligent battery usage strategies. For instance, they can prioritize using the battery's power for propulsion and rely on external power sources for cabin heating or cooling to maximize range and efficiency.

12. Compliance with Regulations: As governments and regulatory bodies implement stricter standards for electric vehicle safety and efficiency, thermal management systems are crucial for ensuring that EVs meet these requirements.

In conclusion, thermal management systems for electric vehicle batteries play a multifaceted role in ensuring the efficient, safe, and reliable operation of EVs. They enable these vehicles to perform optimally under various conditions, offer enhanced safety, and extend the life of the battery pack, making electric mobility a practical and sustainable choice for consumers. thermal management systems for electric vehicle batteries play a multifaceted role in ensuring the efficient, safe, and reliable operation of EVs. They enable these vehicles to perform optimally under various conditions, offer enhanced safety, and extend the life of the battery pack, making electric mobility a practical and sustainable choice for consumers.

2.4 BLOCK DIAGRAM



CHAPTER 3 SOFTWARE REQUIREMENTS

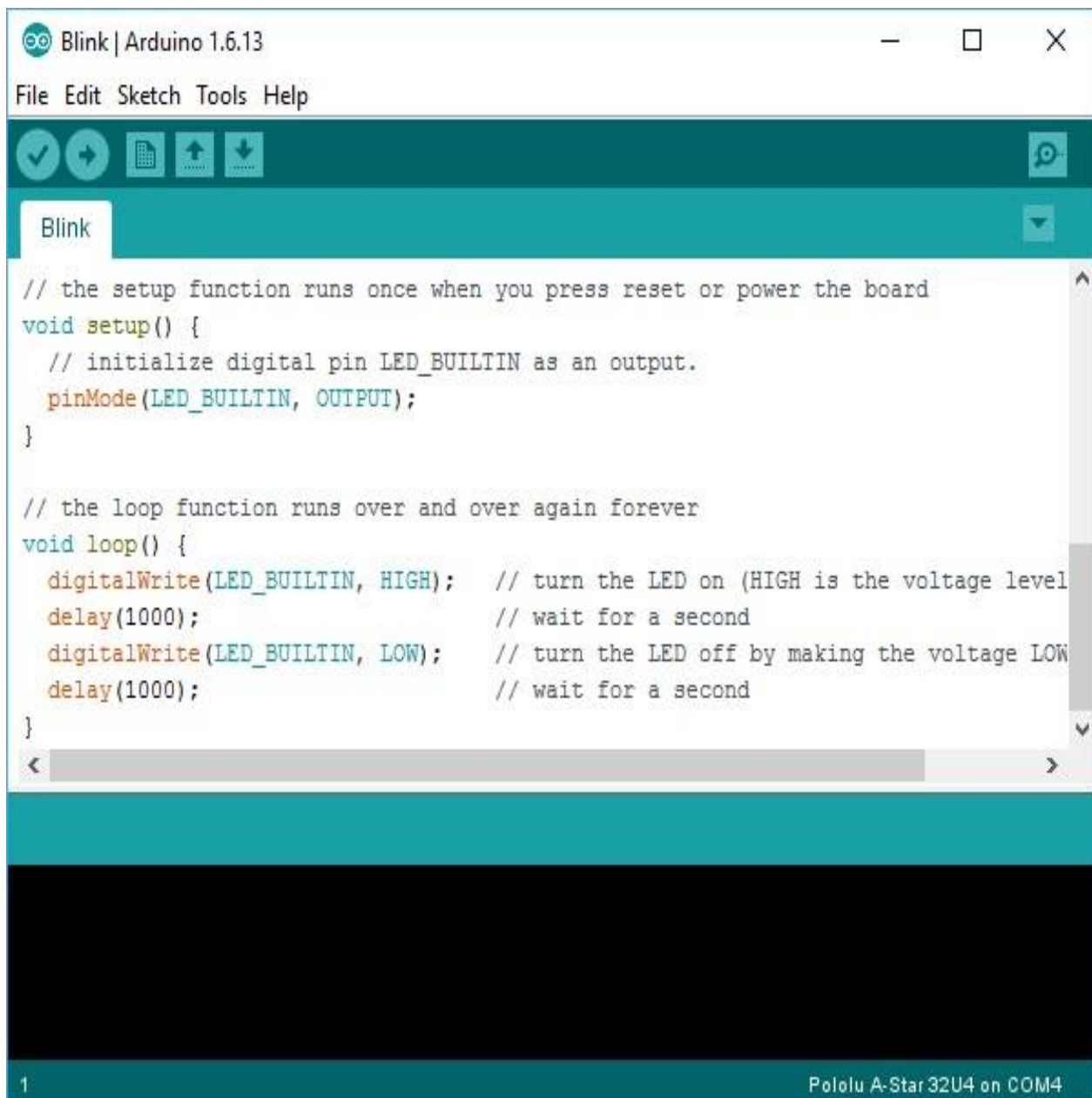
Software used in this project for uploading code onto Arduino is Arduino IDE.

3.1 INTRODUCTION TO ARDUINO IDE

IDE stands for Integrated Development Environment. Pretty fancy sounding, and should make you feel smart any time you use it. The IDE is a text editor-like program that allows you to write Arduino code. When you open the Arduino program, you are opening the IDE. It is intentionally streamlined to keep things as simple and straightforward as possible. When you save a file in Arduino, the file is called a sketch – a sketch is where you save the computer code you have written. The coding language that Arduino uses is very much like C++ (“see plus plus”), which is a common language in the world of computing. The code you learn to write for Arduino will be very similar to the code you write in any other computer language – all the basic concepts remain the same – it is just a matter of learning a new dialect should you pursue other programming languages. which is a common language in the world of computing. The code you learn to write for Arduino will be very similar to the code you write in any other computer language – all the basic concepts remain the same – it is just a matter of learning a new dialect should you pursue other programming languages. The code you learn to write for Arduino will be very similar to the code you write in any other computer language – all the basic concepts remain the same – it is just a matter of learning a new dialect should you pursue other programming languages. The process of compiling is seamless to the user. All you have to do is press a button. If you have errors in your computer code, the compiler will display an error message at the bottom of the IDE and highlight the line of code that seems to be the issue. The error message is meant to help you identify what you might have done wrong – sometimes the message is very explicit, like saying, “Hey – you forget a semicolon”, sometimes the error message is vague. Why be concerned with a semicolon you ask? A semicolon is part of the Arduino language syntax, the rules that govern how the code is written



The code you write is “human readable”, that is, it will make sense to you (sometimes), and will be organized for a human to follow. Part of the job of the IDE is to take the human readable code and translate it into machine-readable code to be executed by the Arduino. This process is called compiling. The process of compiling is seamless to the user. All you have to do is press a button. If you have errors in your computer code, the compiler will display an error message at the bottom of the IDE and highlight the line of code that seems to be the issue. The error message is meant to help you identify what you might have done wrong – sometimes the message is very explicit, like saying, “Hey – you forget a semicolon”, sometimes the error message is vague. Why be concerned with a semicolon you ask? A semicolon is part of the Arduino language syntax, the rules that govern how the code is written. It is like grammar in writing. Say for example we didn’t use periods when we wrote – everyone would have a heck of a time trying to figure out when sentences started and ended. Or if we didn’t employ how would we convey a dramatic pause to the reader And let me tell you, if you ever had an English teacher with an overactive red pen, the compiler is ten times worse. In fact – your programs WILL NOT compile without perfect syntax. This might drive you crazy at first because it is very natural to forget syntax. As you gain experience programming you will learn comma, how

A screenshot of the Arduino IDE interface. The title bar reads "Blink | Arduino 1.6.13". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for checking, running, uploading, and downloading. The main text area shows the "Blink" sketch with the following code:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

The status bar at the bottom shows "1" on the left and "Pololu A-Star 32U4 on COM4" on the right.

would we convey a dramatic pause to the reader And let me tell you, if you ever had an English teacher with an overactive red pen, the compiler is ten times worse. In fact – your programs WILL NOT compile without perfect syntax. This might drive you crazy at first because it is very natural to forget syntax. As you gain experience programming you will learn

fact – your programs WILL NOT compile without perfect syntax. This might drive you crazy at first because it is very natural to forget syntax. As you gain experience programming you will learn to be assiduous about coding grammar.

3.2 THE SEMICOLON

A semicolon needs to follow every statement written in the Arduino programming language. For example, ...

```
Int LedPin=9;
```

In this statement, I am assigning a value to an integer variable (we will cover this later), notice the semicolon at the end. This tells the compiler that you have finished a chunk of code and are moving on to the next piece. A semicolon is to Arduino code, as a period is to a sentence. It signifies a complete statement.

3.1.2 THE DOUBLE BACKSLASH FOR SINGLE LINE COMMENTS //

Comments are what you use to annotate code. Good code is commented well. Comments are meant to inform you and anyone else who might stumble across your code, what the heck you were thinking when you wrote it. A good comment would be something like this...

Now, in 3 months when I review this program, I know where to stick my LED. Comments will be ignored by the compiler – so you can write whatever you like in them. If you have a lot you need to explain, you can use a multi-line comment, shown below...

```
//This is an example
```

Comments are like the footnotes of code, except far more prevalent and not at the bottom of the page.

3.3 THE CURLY BRACES

Curly braces are used to enclose further instructions carried out by a function (we discuss functions next). There is always an opening curly bracket and a closing curly bracket. If you forget to close a curly bracket, the compiler will not like it and throw an error code.

Void loop {} Remember – no curly brace may go unclosed!

3.4 FUNCTION ()

Functions are pieces of code that are used so often that they are encapsulated in certain keywords so that you can use them more easily. For example, a function could be the following set of instructions...

This set of simple instructions could be encapsulated in a function that we call WashDog. Every time we want to carry out all those instructions we just type WashDog and voila – all the instructions are carried out. In Arduino, there are certain functions that are used so often they have been built into the IDE. When you type them, the name of the function will appear orange. The function pinMode(), for example, is a common function used to designate the mode of an Arduino pin.

What's the deal with the parentheses following the function pinMode? Many functions require *arguments* to work. An argument is information the function uses when it runs. For our WashDog function, the arguments might be dog name and soap type, or temperature and size of a bucket.

```
pinMode(13, OUTPUT);
```

The argument 13 refers to pin 13, and OUTPUT is the mode in which you want the pin to operate. When you enter these arguments the terminology is called passing. You pass the necessary information to the functions. Not all functions require arguments, but opening and closing parentheses will stay regardless though empty.

Notice that the word OUTPUT is blue. There are certain keywords in Arduino that are used frequently and the color blue helps identify them. The IDE turns them blue automatically. Now we won't get into it here, but you can easily make your own functions in Arduino, and you can even get the IDE to color them for you. We will, however, talk about the two functions used in nearly EVERY Arduino program.

3.5 VOID SETUP ()

The function, `setup()`, as the name implies, is used to set up the Arduino board. The Arduino executes all the code that is contained between the curly braces of `setup()` only once. Typical things that happen in `setup()` are setting the modes of pins, starting You might be wondering what void means before the function `setup()`. Void means that the function does not return information. Some functions do return values – our `DogWash` function might return the number of buckets it required to clean the dog. The function `analogRead()` returns an integer value between 0-1023. If this seems a bit odd now, don't worry as we will cover every common Arduino function in depth as we continue the course.

Let us review a couple things you should know about `setup()`...

1. `setup()` only runs once.
2. `setup()` needs to be the first function in your Arduino sketch.
3. `setup()` must have opening and closing curly braces.

3.6 VOID LOOP ()

You have to love the Arduino developers because the function names are so telling. As the name implies, all the code between the curly braces in `loop()` is repeated over and over again – in a loop. The `loop()` function is where the body of your program will reside. As with `setup()`, the function `loop()` does not return any values, therefore the word void precedes it.

Does it seem odd to you that the code runs in one big loop? This apparent lack of variation is an illusion. Most of your code will have specific conditions laying in wait which will trigger new actions.

If you have a temperature sensor connected to your Arduino for example, then when the temperature gets to a predefined threshold you might have a fan kick on. The looping code

is constantly checking the temperature waiting to trigger the fan. So even though the code loops over and over, not every piece of the code will be executed every iteration of the loop.

3.7 INTRODUCTION ARDUINO LIBRARIES

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the built-in LiquidCrystal library makes it easy to talk to character LCD displays. There are hundreds of additional libraries available on the Internet for download. The built-in libraries and some of these additional libraries are [listed in the reference](#). To use the additional libraries, you will need to install them.

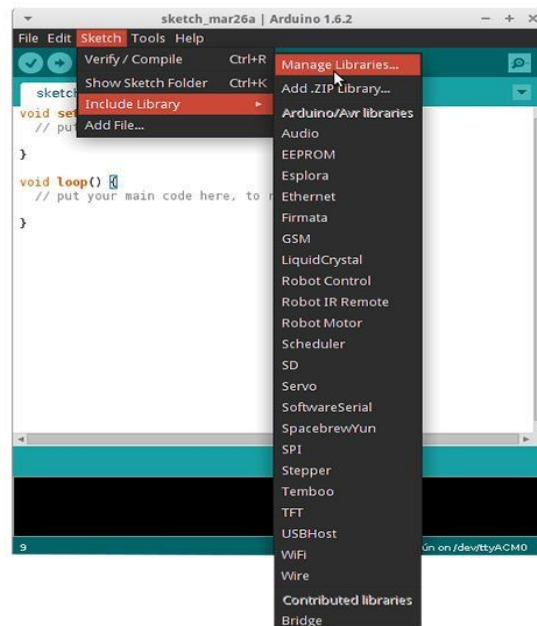
Arduino libraries are managed in three different places: inside the IDE installation folder, inside the core folder and in the libraries folder inside your sketchbook. The way libraries are chosen during compilation is designed to allow the update of libraries present in the distribution. This means that placing a library in the “libraries” folder in your sketchbook overrides the other libraries versions.

The same happens for the libraries present in additional cores installations. It is also important to note that the version of the library you put in your sketchbook may be lower than the one in the distribution or core folders, nevertheless it will be the one used during compilation. When you select a specific core for your board, the libraries present in the core’s folder are used instead of the same libraries present in the IDE distribution folder.

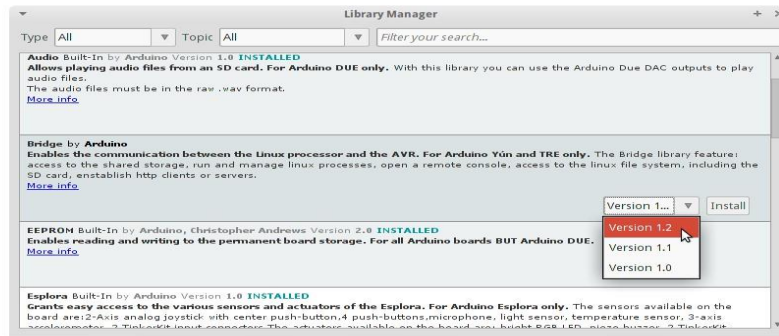
Last, but not least important is the way the Arduino Software (IDE) upgrades itself: all the files in Programs/Arduino (or the folder where you installed the IDE) are deleted and a new folder is created with fresh content. This is why we recommend that you only install libraries to the sketchbook folder so they are not deleted during the Arduino IDE update process.

3.8 HOW TO INSTALL A LIBRARY

To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.6.2). Open the IDE and click to the "Sketch" menu and then *Include Library > Manage Libraries*.



Then the Library Manager will open and you will find a list of libraries that are already installed or ready for installation. In this example we will install the Bridge library. Scroll the list to find it, click on it, then select the version of the library you want to install. Sometimes only one version of the library is available. If the version selection menu does not appear, don't worry: it is normal.



Finally click on install and wait for the IDE to install the new library. Downloading may take time depending on your connection speed. Once it has finished, an *Installed* tag should appear next to the Bridge library. You can close the library manager. You can now find the new library available in the menu. If you want to add your own library to Library Manager, follow these instructions.

3.9 HOW TO CONNECT ARDUINO BOARD

If you're using a serial board, power the board with an external power supply (6 to 25 volts DC, with the core of the connector positive). Connect the board to a serial port on your computer. On the USB boards, the power source is selected by the jumper between the USB and power plugs. To power the board from the USB port (good for controlling low power devices like LEDs), place the jumper on the two pins closest to the USB plug. To power the board from an external power supply (needed for motors and other high current devices), place the jumper on the two pins closest to the power plug. Either way, connect the board to a USB port on your computer. On Windows, the Add New Hardware wizard will open; tell it you want to specify the location to search for drivers and point to the folder containing the USB drivers you unzipped in the previous step.

The power LED should go on.

3.10 HOW TO UPLOAD A PROGRAM

The content of circuits and Arduino sketches can vary greatly. Before you get started, there is one simple process for uploading a sketch to an Arduino board that you can refer back to.

Follow these steps to upload your sketch:

1. Connect your Arduino using the USB cable.

The square end of the USB cable connects to your Arduino and the flat end connects to a USB port on your computer.

2. Choose Tools→Board→Arduino Uno to find your board in the Arduino menu.

You can also find all boards through this menu, such as the Arduino MEGA 2560 and Arduino Leonardo.

3. Choose the correct serial port for your board.

You find a list of all the available serial ports by choosing Tools→Serial Port→ comX or /dev/tty.usbmodemXXXXX. X marks a sequentially or randomly assigned number. In Windows, if you have just connected your Arduino, the COM port will normally be the highest number, such as com 3 or com 15.

Many devices can be listed on the COM port list, and if you plug in multiple Arduinos, each one will be assigned a new number. On Mac OS X, the /dev/tty.usbmodem number will be randomly assigned and can vary in length, such as /dev/tty.usbmodem1421 or /dev/tty.usbmodem262471. Unless you have another Arduino connected, it should be the only one visible.

4. Click the Upload button. This is the button that points to the right in the Arduino environment. You can also use the keyboard shortcut Ctrl+U for Windows or Cmd+U for Mac OS X.

CHAPTER 4 HARDWARE

REQUIREMENTS

Hardware Components of this project are

1. ARDIUNO UNO
2. 16X2 LCD

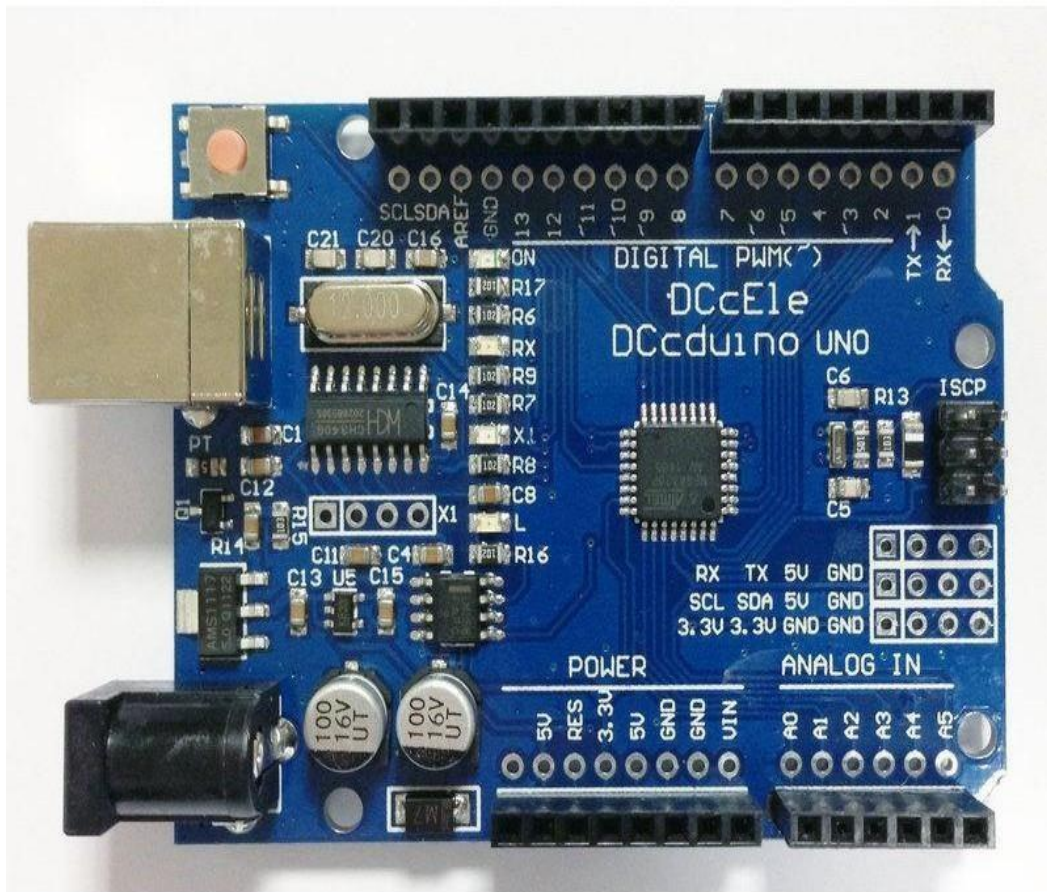
3. L298N
4. DC FAN 12V
5. DHT11
6. VOLTAGE SENSOR
7. BUZZER
8. 18650 BATTERY

4.1 INTRODUCTION TO ARDUINO UNO

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The [software](#), too, is open-source, and it is growing through the contributions of users worldwide.



4.2 WHY ARDUINO

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

4.3 ADVANTAGES OF ARDUINO

- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

4.4 FEATURES OF ARDUINO UNO

The **Arduino Uno** is a microcontroller board based on the ATmega328. Arduino is an opensource, prototyping platform and its simplicity makes it ideal for hobbyists to use as well as professionals. The Arduino Uno has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-toDC adapter or battery to get started.

Features of the Arduino UNO:

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

The **Arduino Uno** is a microcontroller board based on the ATmega328. Arduino is an open-source, prototyping platform and its simplicity makes it ideal for hobbyists to use as well as professionals. The Arduino Uno has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The diagram illustrates the pin configuration of an Arduino Uno R3 board. The pins are color-coded and labeled as follows:

- Power Pins:** IOREF, RESET, 3.3V, 5V, GND, VIN, A0, A1, A2, A3, A4, A5.
- Digital Pins:** PC5, PC4, AREF, GND, PB5, PB4, PB3, PB2, PB1, PB0, PD7, PD6, PD5, PD4, PD3, PD2, PD1, PD0.
- Analog Pins:** A0, A1, A2, A3, A4, A5.
- Special Function Pins:** SCL, SDA, INT1, INT0, TX, RX.

The board is shown from a top-down perspective, with the USB port on the left and the power jack on the right.

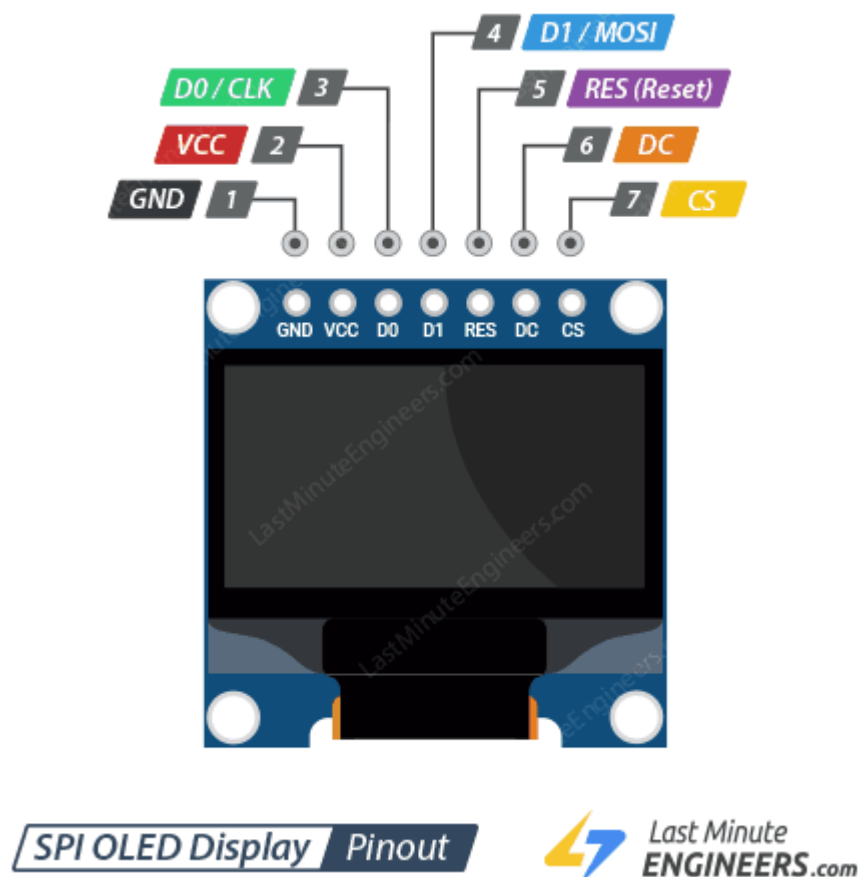


A liquid-crystal display (LCD) is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals. Liquid crystals do not emit light directly, instead using a backlight or reflector to produce images in color or monochrome.^[1] LCDs are available to display arbitrary images (as in a general-purpose computer display) or fixed images with low information content, which can be displayed or hidden, such as preset words, digits, and 7-segment displays, as in a digital clock. They use the same basic technology, except that arbitrary images are made up of a large number of small pixels, while other displays have larger elements.

28

LCD screens available in sizes ranging from tiny digital watches to huge, big-screen television sets.

Since LCD screens do not use phosphors, they do not suffer image burn-in when a static image is displayed on a screen for a long time (e.g., the table frame for an aircraft schedule on an indoor sign). LCDs are, however, susceptible to image persistence.^[2] The LCD screen is more energy-efficient and can be disposed of more safely than a CRT can. Its low electrical power consumption enables it to be used in battery-powered electronic equipment more efficiently than CRTs can be. By 2008, annual sales of televisions with LCD screens exceeded sales of CRT units worldwide, and the CRT became obsolete for most purposes.



A OLED Display for Arduino

Overview:

OLED (Organic Light Emitting Diode) displays are popular for Arduino projects.

They provide high contrast, bright displays, and wide viewing angles.

Common sizes: 0.96-inch, 1.3-inch, etc.

Types:

Monochrome: Single color, typically white or blue.

Color: Multiple colors, usually RGB.

Communication Interfaces:

I2C: Uses two pins (SDA and SCL). Easy to connect and wire.

SPI: Faster, uses more pins (MOSI, MISO, SCK, and SS).

Connecting to Arduino:

I2C Connection:

VCC to 3.3V or 5V

GND to GND

SDA to A4

SCL to A5

SPI Connection:

VCC to 3.3V or 5V

GND to GND

MOSI to D11

MISO to D12

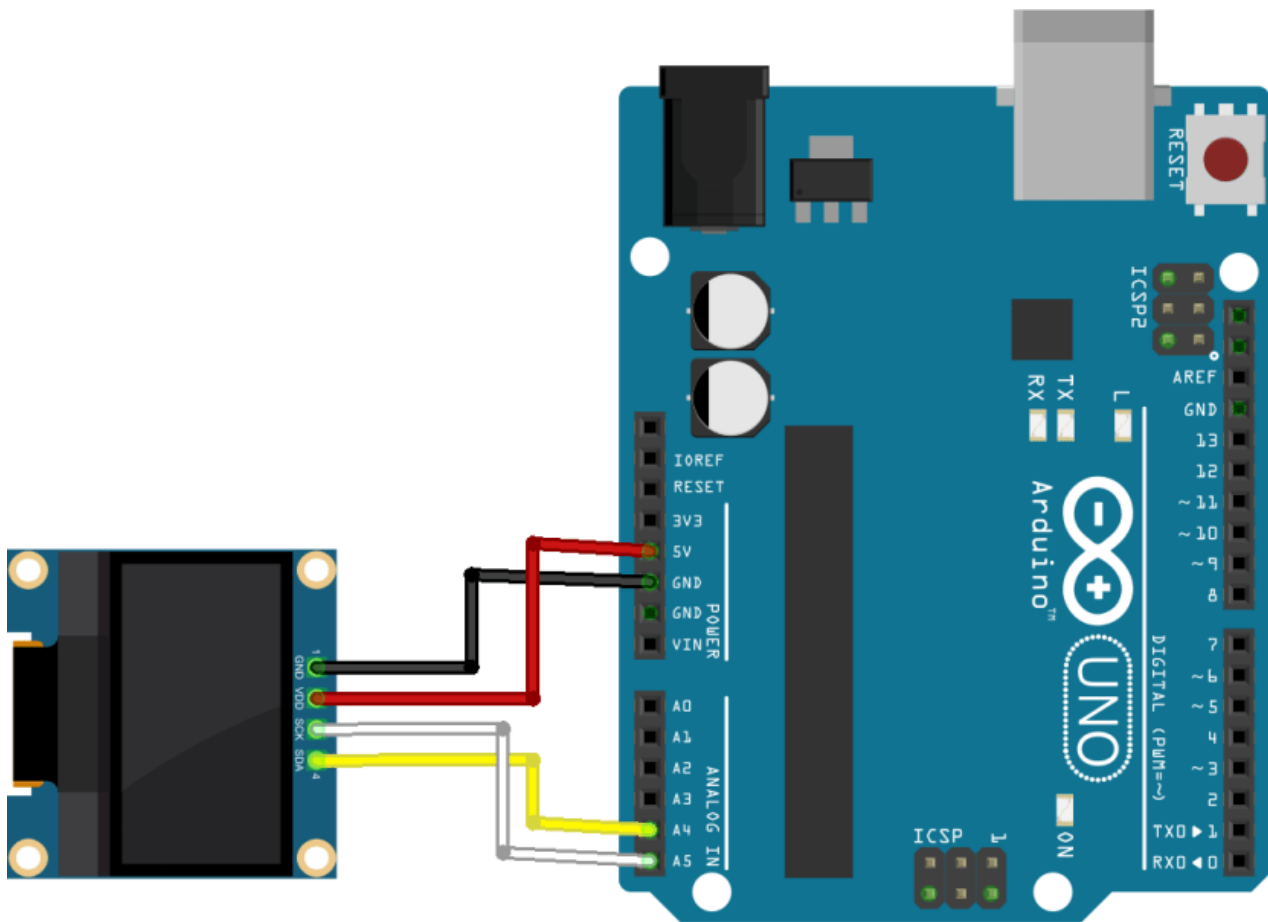
SCK to D13

SS to D10 (or any other digital pin)

Libraries:

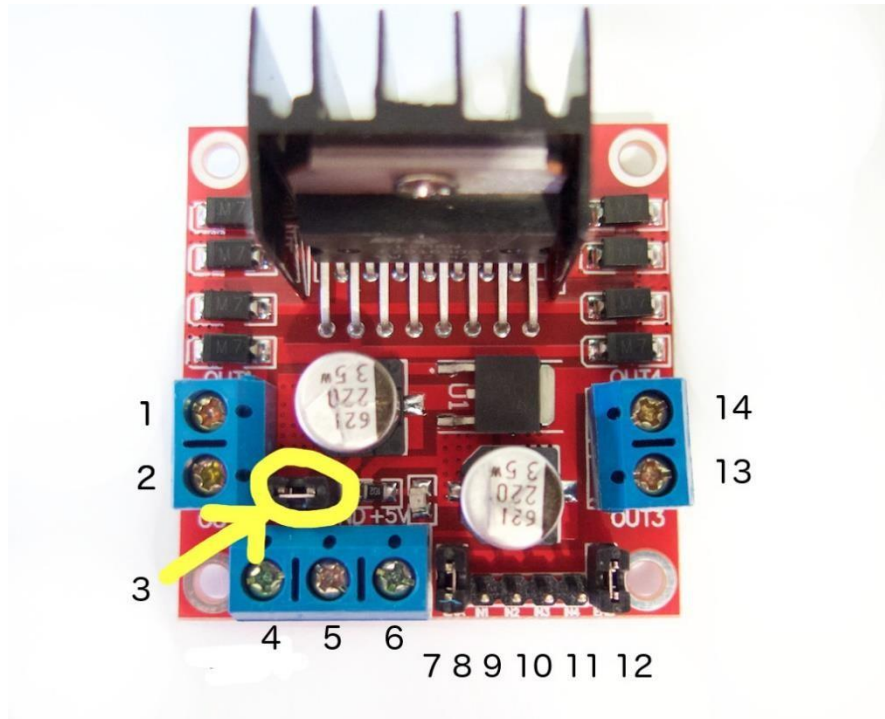
Adafruit SSD1306: For controlling the display.

Adafruit GFX: For graphics and text.



4.6 INTRODUCTION TO L293D MOTOR DRIVER

Dual Motor Controller Module 2A with Arduino. This allows you to control the speed and direction of two DC motors, or control one bipolar stepper motor with ease. The L298N Hbridge module can be used with motors that have a voltage of between 5 and 35V DC. There is also an onboard 5V regulator, so if your supply voltage is up to 12V you can also source 5V from the board. These L298 H-bridge dual motor controller modules .



4.7 PIN DESCRIPTION

1. DC motor 1 "+" or stepper motor A+
2. DC motor 1 "-" or stepper motor A-
3. 12V jumper - remove this if using a supply voltage greater than 12V DC. This enables power to the onboard 5V regulator
4. Connect your motor supply voltage here, maximum of 35V DC. Remove 12V jumper if >12V DC
5. GND
6. 5V output if 12V jumper in place, ideal for powering your Arduino (etc)
7. DC motor 1 enable jumper. Leave this in place when using a stepper motor. Connect to PWM output for DC motor speed control.
8. IN1
9. IN2
10. IN3
11. IN4
12. DC motor 2 enable jumper. Leave this in place when using a stepper motor. Connect to PWM output for DC motor speed control.
13. DC motor 2 "+" or stepper motor B+

14. DC motor 2 "-" or stepper motor B-

4.8 CONTROLLING DC MOTOR

To control one or two DC motors is quite easy. First connect each motor to the A and B connections on the L298N module. If you're using two motors for a robot (etc) ensure that the polarity of the motors is the same on both inputs. Otherwise you may need to swap them over when you set both motors to forward and one goes backwards!

Next, connect your power supply - the positive to pin 4 on the module and negative/GND to pin 5. If your supply is up to 12V you can leave in the 12V jumper (point 3 in the image above) and 5V will be available from pin 6 on the module. This can be fed to your Arduino's 5V pin to power it from the motors' power supply. Don't forget to connect Arduino GND to pin 5 on the module as well to complete the circuit.

Now you will need six digital output pins on your Arduino, two of which need to be PWM (pulse-width modulation) pins. PWM pins are denoted by the tilde ("~") next to the pin number, for example: Finally, connect the Arduino digital output pins to the driver module. In our example we have two DC motors, so digital pins D9, D8, D7 and D6 will be connected to pins IN1, IN2, IN3 and IN4 respectively. Then connect D10 to module pin 7 (remove the jumper first) and D5 to module pin 12 (again, remove the jumper).

The motor direction is controlled by sending a HIGH or LOW signal to the drive for each motor (or channel). For example for motor one, a HIGH to IN1 and a LOW to IN2 will cause it to turn in one direction, and a LOW and HIGH will cause it to turn in the other direction. However the motors will not turn until a HIGH is set to the enable pin (7 for motor one, 12 for motor two). And they can be turned off with a LOW to the same pin(s). However if you need to control the speed of the motors, the PWM signal from the digital pin connected to the enable pin can take care of it.

4.9 INTRODUCTION TO 12V DC FAN

A **computer fan** is any fan inside, or attached to, a computer case used for active cooling, and may refer to fans that draw cooler air into the case from the outside, expel warm air from inside, or move air across a heat sink to cool a particular component. Generally, these are found in axial and sometimes centrifugal forms. The former is sometimes called a "electric"

fan, after the Rotron Vertical line, while the latter may be called a "biscuit blower" in some product literature.

4.10 PHYSICAL CHARACTERISTICS

Due to the low pressure, high volume air flows they create, most fans used in computers are of the axial flow type; centrifugal and crossflow fans type. Two important functional specifications are the airflow that can be moved, typically stated in cubic feet per minute (CFM), and static pressure.^[9] Given in decibels, the sound volume figure can be also very important for home and office computers; larger fans are generally quieter for the same CFM. Many gamers, case modders, and enthusiasts utilize fans illuminated with colored LED lights. Multi-colored fans are also available.



4.11 Peltier Module:

A semiconductor device with two sides - one gets cold (absorbs heat) and the other gets hot (releases heat).

Heat Sink: Attached to the hot side to dissipate heat and improve efficiency.

Power Supply: Provides the necessary current to the Peltier module.

Applications:

Cooling Electronics: Keeps CPUs, GPUs, and other components cool.

Portable Coolers: Used in mini refrigerators and beverage coolers.

Temperature Control: In scientific instruments and sensors.

Advantages:

Compact and Lightweight: No moving parts, making it silent and reliable.

Precise Temperature Control: Can be adjusted by changing the current.

Disadvantages:

Efficiency: Less efficient compared to traditional refrigeration.

Heat Management: Requires effective heat dissipation for the hot side.

Example Use in Arduino Projects:

You can control a Peltier module with Arduino using a MOSFET and a temperature sensor to create a feedback loop for maintaining a specific temperature.

Basic Circuit:

Connect the Peltier module to a heat sink.

Use a MOSFET to control the power to the Peltier module.

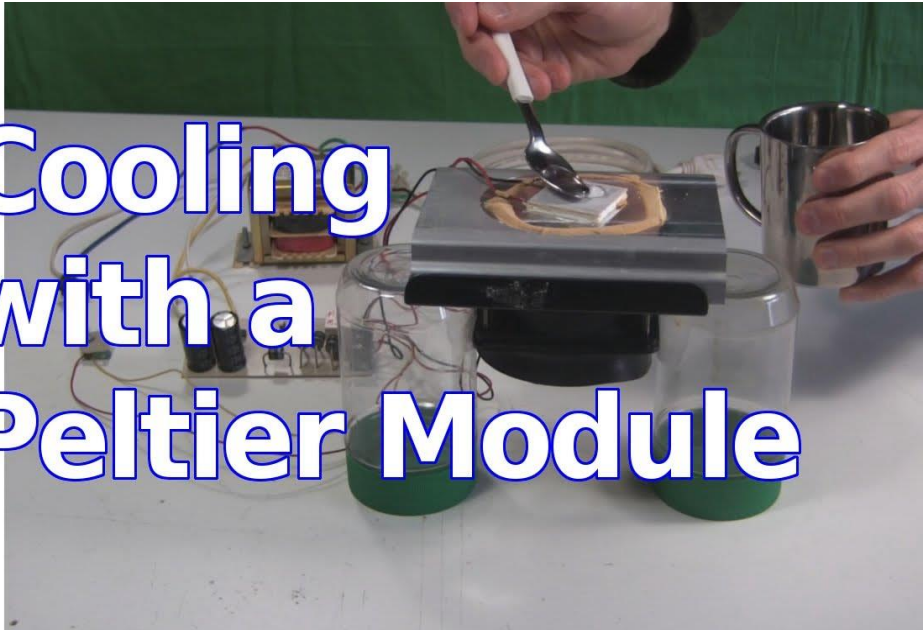
Connect a temperature sensor (like DS18B20) to Arduino.

Write a code to control the MOSFET based on the temperature readings.

The dimensions and mounting holes must suit the equipment that uses the fan. Squareframed fans are usually used, but round frames are also used, often so that a larger fan than the mounting holes would otherwise allow can be used (e.g., a 120 mm fan with holes for the corners of a 90 mm square fan).

The width of square fans and the diameter of round ones are usually stated in millimeters. The dimension given is the outside width of the fan, not the distance between mounting holes. although

8 mm, 17 mm, 20 mm, 25 mm, 30 mm, 35 mm, 38 mm, 45 mm, 50 mm, 70 mm, 250 mm and 360 mmsizes are also available. Heights are typically 10 mm, 25 mm or 38 mm, but this is usually not an important dimension as it does not affect mounting holes or apertures in the case.



Cooling with a Peltier Module

Typically, square 120 mm and 140 mm case and power supply fans are used where cooling requirements are demanding, as for computers used to play games, and for quieter operation at lower speeds. 80 mm and 92 mm fans are used in less demanding applications, or where larger fans would not be compatible. Smaller fans are usually used for cooling CPUs, graphics cards, northbridges, etc.

4.12 ROTATIONAL SPEED

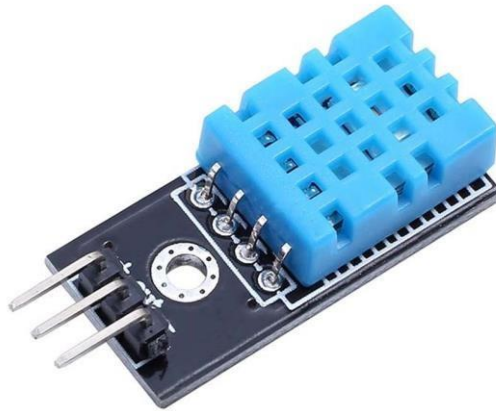
The speed of rotation (specified in revolutions per minute, RPM) together with the static pressure determine the airflow for a given fan. Where noise is an issue, larger, slower turning fans are quieter than smaller, faster fans that can move the same airflow. Fan noise has been found to be roughly proportional to the fifth power of fan speed; halving the speed reduces the noise by about 15 db. Axial fans may rotate at speeds of up to around 23,000 rpm for smaller sizes. faster fans that can move the same airflow. Fan noise has been found to be roughly proportional to the fifth power of fan speed; halving the speed reduces the noise by about 15 db. Axial fans may rotate at speeds of up to around 23,000 rpm for smaller sizes. Where noise is an issue, larger, slower-turning fans are quieter than smaller, faster fans that can move the same airflow. Fan noise has been found to be roughly proportional to the fifth power of fan speed; halving the speed reduces the noise by about 15 db. Axial fans may rotate at speeds of up to around 23,000 rpm for smaller sizes. faster fans that can move the same airflow.



|| Lisa Store

4.13 INTRODUCTION TO DHT 11 SENSOR

The DHT11 is a basic, ultra-low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.



bulemon

4.14 TECHNICAL DETAILS

- Low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20-80% humidity readings with 5% accuracy
- Good for 0-50°C temperature readings $\pm 2^{\circ}\text{C}$ accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

INTRODUCTION TO 1-CHANNEL RELAY MODULE:

1-Channel Relay Module

Overview:

A relay is an electrically operated switch.

It allows you to control high voltage devices with a low voltage signal, such as from an Arduino.



Components:

Relay: The switch itself.

Optocoupler: Provides electrical isolation between the Arduino and the high voltage side.

Transistor: Amplifies the control signal from the Arduino.

Diode: Protects against back EMF generated when the relay coil is de-energized.

LED: Indicates the status of the relay.

Pins:

VCC: Connects to the 5V pin on the Arduino.

GND: Connects to the GND pin on the Arduino.

IN: Control signal pin, connects to a digital pin on the Arduino.

Relay Output Terminals:

COM (Common): Common terminal.

NO (Normally Open): Connected to COM when the relay is activated.

NC (Normally Closed): Connected to COM when the relay is deactivated.

Connecting to Arduino:

VCC to 5V on Arduino.

GND to GND on Arduino.

IN to a digital pin (e.g., D2) on Arduino.

Controlling the Relay:

Set the digital pin HIGH to activate the relay (connect COM to NO).

Set the digital pin LOW to deactivate the relay (connect COM to NC).

4.18 INTRODUCTION TO 18650 BATTERY



A **lithium-ion battery** or **Li-ion battery** is a type of rechargeable battery in which lithium ions move from the negative electrode through an electrolyte to the positive electrode during discharge, and back when charging. Li-ion batteries use an intercalated lithium compound as the material at the positive electrode and typically graphite at the negative electrode.

Li-ion batteries have a high energy density, no memory effect (other than LFP cells)^[9] and low self-discharge. Cells can be manufactured to either prioritize energy or power density.^[10] They can however be a safety hazard since they contain flammable electrolytes, and if damaged or incorrectly charged can lead to explosions and fires. Samsung was forced to recall Galaxy Note 7 handsets following lithium-ion fires,^[11] and there have been several incidents involving batteries on Boeing 787s.

A prototype Li-ion battery was developed by Akira Yoshino in 1985, based on earlier research by John Goodenough, M. Stanley Whittingham, Rachid Yazami and Koichi Mizushima during the 1970s–1980s,^{[12][13][14]} and then a commercial Li-ion battery was developed by a Sony and Asahi Kasei team led by Yoshio Nishi in 1991.^[15]

Lithium-ion batteries are commonly used for portable electronics and electric vehicles and are growing in popularity for military and aerospace applications.^[16]

Chemistry, performance, cost and safety characteristics vary across types of lithium-ion batteries. Handheld electronics mostly use lithium polymer batteries (with a polymer gel as electrolyte), a lithium cobalt oxide (LiCoO₂), 3-based lithium rich layered materials, LMR-NMC), and lithium nickel manganese cobalt oxide (LiNiMnCoO₂)

2 or NMC) may offer longer lives and may have better rate capability. Such batteries are widely used for electric tools, medical equipment, and other roles. NMC and its derivatives are widely used in electric vehicles.

Research areas for lithium-ion batteries include extending lifetime, increasing energy density, improving safety, reducing cost, and increasing charging speed,^[19] among others. Research has been under way in the area of non-flammable electrolytes as a pathway to increased safety based on the flammability and volatility of the organic solvents used in the typical electrolyte. Strategies include aqueous lithium-ion batteries, ceramic solid electrolytes, polymer electrolytes, ionic liquids, and heavily fluorinated systems

CHAPTER 5 PROGRAM CODE

```
#include "U8glib.h"
```

```
#include <DHT.h>
```

```
U8GLIB_SSD1306_128X64 u8g(U8G_I2C_OPT_NONE);
```

```
int posicao = 0;
```

```
int temp_max = 45;
```

```
bool relayActivated = false;
```

```

bool showCoolingMessage = false;

#define pino_aumenta 5
#define pino_diminui 4
#define pino_rele 7
#define pino_DHT A0
#define DHTTYPE DHT11

DHT dht(pino_DHT, DHTTYPE);

int temperatura;
int aumenta, diminui;
int prev_temp_max = temp_max;

unsigned long previousMillis = 0;
const long interval = 10000;
const long welcomeDuration = 5000; // 5 seconds
const long coolingMessageDuration = 5000; // 5 seconds

void draw()
{
    // Retangulo temperatura atual
    u8g.drawRFrame(0, 17, 84, 46, 2);
    u8g.setFont(u8g_font_profont29 );// u8g_font_fub20
    // Atualiza a temperatura no display
    u8g.setPrintPos(9, 50);

```

```

u8g.print(temperatura);

// Circulo grau

u8g.drawCircle(51, 33, 3);

u8g.drawStr( 56, 50, "C");


// Box superior amarelo

u8g.setFont(u8g_font_tpss);

u8g.drawRBox(0, 0, 128, 18, 0);

u8g.setColorIndex(0);

u8g.setPrintPos(posicao, 11);

u8g.print("Ev Cooling System by ECE Department...!");

u8g.setColorIndex(1);


// Title "Battery Temperature"

u8g.setFont(u8g_font_tpss);

u8g.setPrintPos(10, 30);

u8g.print("Battery Temp");


// Box temperatura maxima

u8g.drawRFrame(86, 17, 42, 46, 2);

if (temp_max <= temperatura)

{

    // Temperature greater than temp. SET

    u8g.drawRBox(86, 17, 42, 22, 2);

    u8g.setColorIndex(0);

    u8g.drawStr(96, 33, "SET");

```



```

    u8g.setColorIndex(1);

    // Aciona saida do rele

    digitalWrite(pino_rele, LOW);

    relayActivated = true;
}

else
{
    // Temperature lower than temp. SET

    u8g.drawRFrame(86, 17, 42, 22, 2);

    u8g.drawStr(96, 33, "SET");

    // Desliga saida do rele

    digitalWrite(pino_rele, HIGH);

    relayActivated = false;
}

// Atualiza na tela o valor da temp. maxima

u8g.setPrintPos(100, 55);

u8g.print(temp_max);

u8g.drawCircle(120, 47, 2);

u8g.setColorIndex(1);
}

void welcomeMessage()
{
    u8g.setRot180(); // Rotate the display by 180 degrees for welcome message

    u8g.firstPage();

    do

```

```

{
    u8g.setFont(u8g_font_profont29 );
    u8g.drawStr(10, 30, "Welcome to");
    u8g.setFont(u8g_font_profont15 );
    u8g.drawStr(10, 50, "Vaagdevi College");
}

while(u8g.nextPage());

delay(welcomeDuration);
}

```

```

void coolingActivatedMessage()
{
    u8g.firstPage();
    do
    {
        u8g.setFont(u8g_font_profont15 );
        u8g.drawStr(10, 30, "Cooling System");
        u8g.drawStr(10, 50, "Activated");
        // Add fan symbol here if available
    }
    while(u8g.nextPage());
}

```

```

void coolingDeactivatedMessage()
{

```

```

u8g.firstPage();

do
{
    u8g.setFont(u8g_font_profont15 );
    u8g.drawStr(10, 30, "Cooling System");
    u8g.drawStr(10, 50, "Deactivated");

    // Add fan symbol here if available

}
while(u8g.nextPage());
}

```

```

void setup(void)
{
    Serial.begin(9600);

    pinMode(pino_rele, OUTPUT);
    pinMode(pino_aumenta, INPUT);
    pinMode(pino_diminui, INPUT);
    dht.begin();

    // Display welcome message

    welcomeMessage();
}

```

```

void loop(void)
{
    unsigned long currentMillis = millis();

```

```
// Timer para ler o valor da temperatura

if (currentMillis - previousMillis >= interval)

{

    temperatura = dht.readTemperature();

    previousMillis = currentMillis;

}
```

```
// Testa botao aumenta temperatura

aumenta = digitalRead(pino_aumenta);

if (aumenta == 1)

{

    temp_max++;

    if (temp_max != prev_temp_max) {

        showCoolingMessage = false;

    }

    prev_temp_max = temp_max;

}

while (digitalRead(pino_aumenta) == 1)

{

    delay(100);

}
```

```
// Testa botao diminui temperatura

diminui = digitalRead(pino_diminui);

if (diminui == 1)

{
```

```

temp_max--;

if (temp_max != prev_temp_max) {
    showCoolingMessage = false;
}

prev_temp_max = temp_max;
}

while (digitalRead(pino_diminui) == 1)
{
    delay(100);
}

// Scroll the text

posicao--;

if (posicao < -300)
{
    posicao = 128; // Reset position to start
}

if (relayActivated && !showCoolingMessage)
{
    coolingActivatedMessage();

    showCoolingMessage = true;

    delay(coolingMessageDuration);
}

else if (!relayActivated && showCoolingMessage)
{

```

```

        coolingDeactivatedMessage();

        showCoolingMessage = false;

        delay(coolingMessageDuration);
    }

    else

    {

        // Rotate the display by 180 degrees for main GUI layout

        u8g.setRot180();

        u8g.firstPage();

        // Call the drawing routine on the screen

        do

        {

            draw();

        }

        while ( u8g.nextPage() );

    }

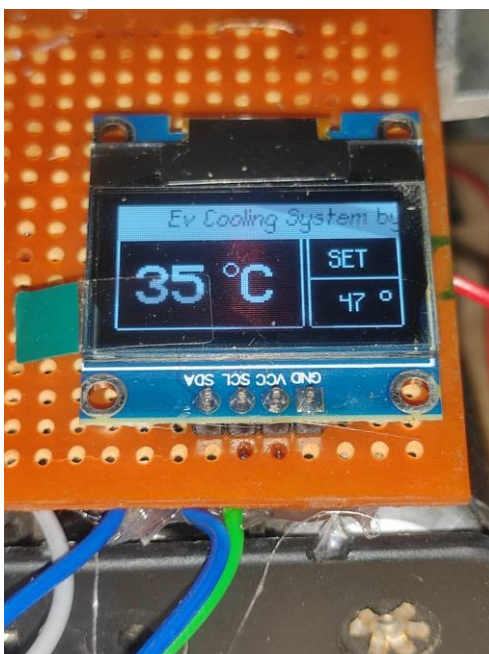
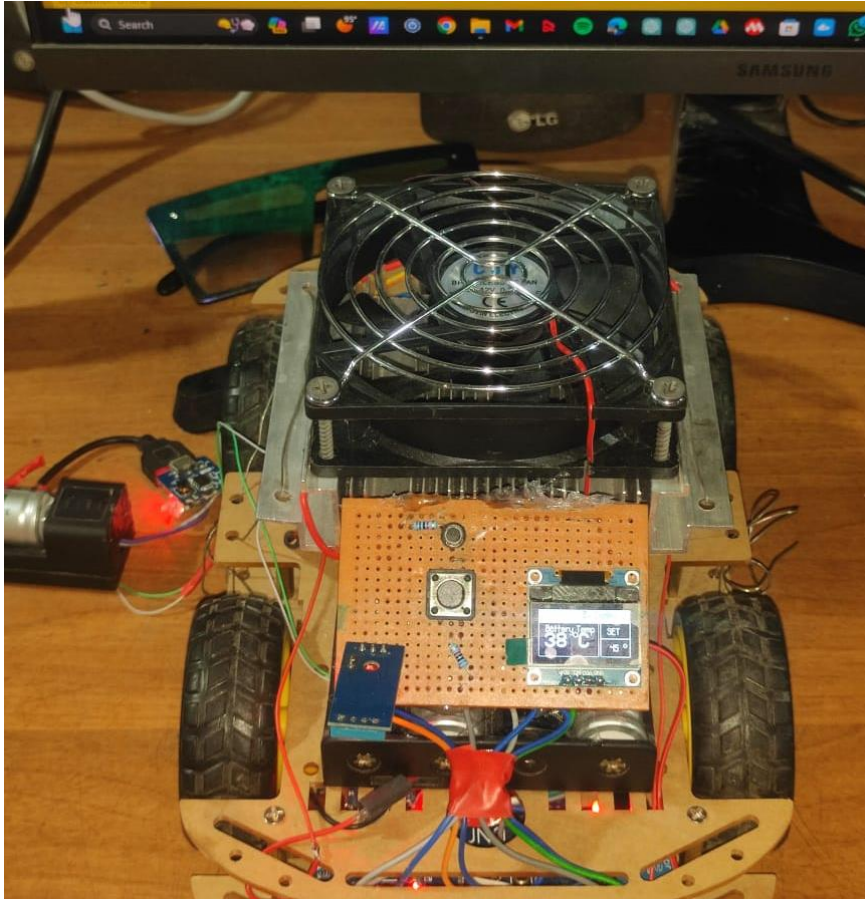
    delay(50);

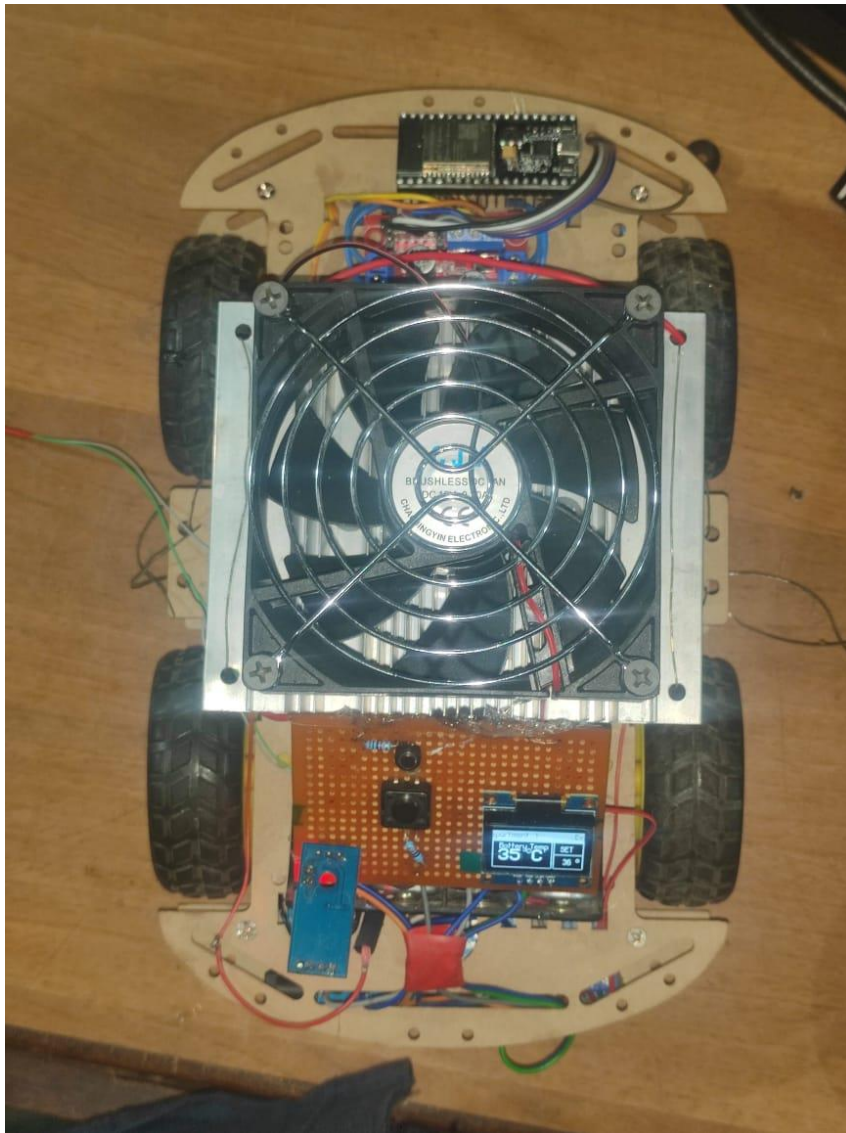
}

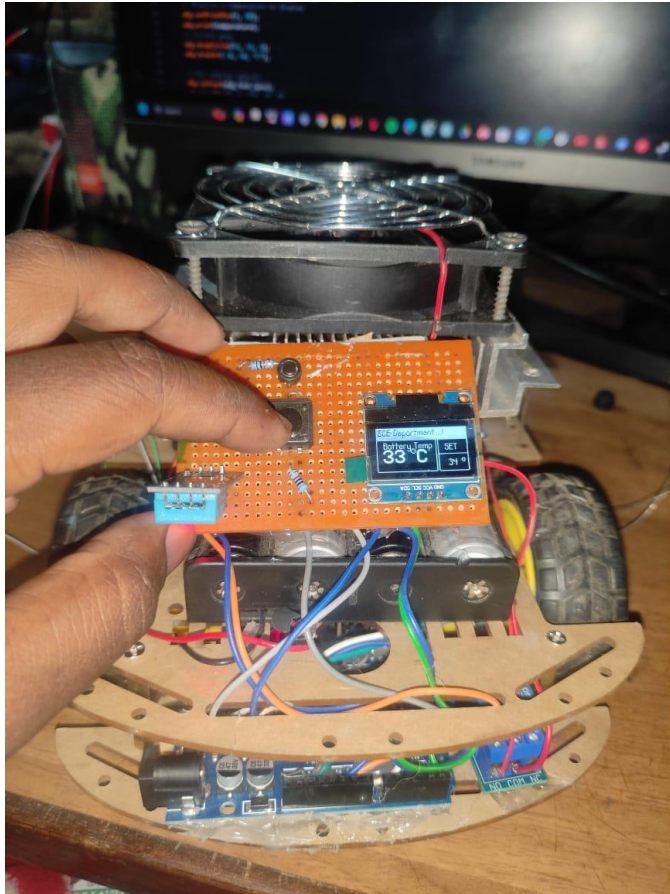
```

CHAPTER-6

RESULT :







CHAPTER 7

CONCLUSION

The role of thermal management systems in electric vehicles cannot be overstated. As the world progresses towards a greener and more sustainable future, the widespread adoption of electric vehicles is not just an option but a necessity. However, the successful integration and operation of electric vehicle batteries hinge on effective thermal management solutions. As explored in this discussion, these systems are instrumental in optimizing battery performance, enhancing safety, and ensuring the longevity of the battery pack.

In conclusion, the rapid advancements in thermal management technologies are a testament to the automotive industry's commitment to delivering reliable and efficient electric vehicles.

Engineers and innovators continue to explore novel solutions that will drive the electric vehicle revolution forward. With the continued development of thermal management systems, electric vehicles will become even more appealing to consumers, and the environmental benefits of EVs will be further realized. The journey towards a cleaner, more sustainable transportation future is undoubtedly intertwined with the evolution of thermal management systems for electric vehicle batteries, making them an indispensable component in the electrified automotive landscape.