

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieure et de la Recherche Scientifique
Université d'Oran 2 Mohamed Ben Ahmed
Institut de Maintenance et de Sécurité Industrielle
Département de Maintenance en Instrumentation



MÉMOIRE

Pour l'obtention du diplôme de
Master

Filière : Génie Industriel

Spécialité : Maintenance en
automatique et instrumentation

Topic

Color Sorting System based on Arduino

Présenté et soutenu publiquement par :

BOUBEKER ISLAM

BELAIDI ABDELLAH

Devant le jury composé de :

Nom et Prénom	Grade	Etablissement	Qualité
Mr. Abdallah HAMIDA	MAA	Univ. Oran-2, IMSI	Président
Mr. Mawloud TITAH	MCA	Univ. Oran-2, IMSI	Examineur
Mme. Farida DJELGHOUM	MAA	Univ. Oran-2, IMSI	Encadreur
Mr. Samir ADDA NEGGAZ	MAA	Univ. Oran-2, IMSI	Co-encadreur

Academic year: 2024/2025

Abstract

This project presents the design and implementation of an automated color-based sorting system using an Arduino Uno microcontroller. The system identifies object colors using a TCS34725 color sensor and sorts them via a conveyor mechanism powered by a gearmotor and two servomotors. Sorting decisions are based on color assignments, which are defined remotely through a web application that communicates with the Arduino via a local HTTP server. The system features real-time monitoring on an LCD display, alert messages for bin overflow, and a general stockpile for unclassified or excess items. It also includes a manual mode for direct control of the system components. This solution offers an efficient, flexible, and scalable approach to automated sorting for industrial and educational applications.

Acknowledgement



First, we thank God for granting us the courage, patience, health, and determination to complete this work.

We would like to express our sincere gratitude to our supervisor, Mme. F, DJELGHOUM, and our co-supervisor Mr. S, ADDA NEGGAZ for thier assistance, availability, and the valuable guidance they have continuously provided throughout the course of this modest work. They have our deepest thanks, recognition, and respect.

Our heartfelt thanks also go to the members of the jury for their interest in our research, for agreeing to evaluate this work, and for enriching it with their suggestions.

Finally, we extend our thanks to all those who contributed in any way to the completion of this project. May all the teachers who participated in our training receive our deepest respect.

Dedication



This work, and so much more, I owe to my beloved parents, who have provided me with unwavering support and trust every single day. Words alone cannot express my deep gratitude. May God protect you and keep you safe for us.

To my sister

To my brothers

And finally, to all my friends and family

Islam

Dedication



This work, and so much more, I owe to my dearest parents, who have provided me with unwavering daily support and trust. For that, words alone cannot express the depth of my gratitude. May God protect you and keep you safe for us.

To my sisters

To my brothers

To my cousins

And finally, to all my dear friends

Abdellah

SOMMAIRE

Introduction	1
--------------------	---

Chapitre 1 : Automation of Production Lines

1.1 Introduction	4
1.2 Definition of production lines	4
1.3 Historical evolution of the production lines	4
1.4 Automation of production lines	6
1.4.1 Definition of automation	6
1.4.2 Automation of certain tasks in production lines	6
1) Painting and coating	7
2) Quality control	8
3) Packaging and conditioning	9
4) Selective sorting	10
1.5 Work presentation (Problem statement).....	11
1.6 Discussion	11

Chapitre 2 : Hardware Design

2.1 Introduction	13
2.2 Developement boards.....	13
2.3 Arduino UNO board	14
2.3.1 Key features	15
2.4 Arduino communication protocols	15
2.4.1 UART Communication protocol.....	15
2.4.2 Serial peripheral interface (SPI).....	17
2.4.3 I2C Communications.....	23
2.4.4 Comparaison of UART, SPI and I2C.....	23
2.4.5 Advantages of arduino boards	24
2.4.6 Applications in Prototyping and Research.....	25

2.4.7 The features of different types of Arduino Boards	26
2.5 The L298N Motor Driver	27
2.5.1 Operational Principles	28
2.5.2 Pulse width modulation (PWM)	30
2.5.3 H-Bridge configuration	31
2.5.4 Applications	32
2.5.5 L298N Pinout.....	35
2.5.6 Advantages and limitations	35
2.6 Actuators	35
2.6.1 Gear Motor 12V 120 RPM	37
1) Technical specifications	37
2) Operational principles.....	37
3) Applications	38
2.6.2 MG5010 Servomotor.....	39
1) Technical specifications	40
2) Operational principles	41
3) Controlling Servomotors: the PWM technique.....	41
4) Applications	42
2.7 Sensors	42
2.7.1 Color sensor TCS34725	42
1) Module connections	43
2) Working principles	44
3) Applications	46
2.8 LCD I2C Module	46
2.8.1 Overview	46
2.8.2 Technical specifications	47
2.8.3 Working principles.....	47

2.8.4 Pin configuration.....	48
2.8.5 Applications in the sorting system	49
2.8.6 I2C Serial interface adapter module for LCD.....	49

Chapitre 3: Control logic and circuit simulation

3.1 Introduction.....	52
3.2 Software tools overview	52
3.2.1 Arduino IDE	52
1) Arduino IDE Installation (Windows)	52
2) General program structure	53
3) Program implementation.....	54
4) Program description.....	55
5) Installing required libraries	60
6) Compilation and upload process	62
3.3 Tinkercad.....	63
3.3.1 Overview	63
3.3.2 Interface	64
3.3.3 Circuit design workspace	65
3.3.4 Simulation steps in Tinkercad	66

Chapitre 4 : Implementation & testing

4.1 Intrduction	70
4.2 Project envirenment	70
4.2.1 Sorting logic	70
4.3 Simulation	71
4.3.1 Power supply and wiring.....	72
4.3.2 Code and library integration.....	72
4.3.3 Color detection emulation	72
4.3.4 Motor control : Convoyer mechanism	72

4.3.5 Sorting mechanism	72
4.4 Electronics components wiring & programming.....	73
4.4.1 Servomotors.....	73
4.4.2 Gear motor with L298N	74
4.4.3 LCD I2C.....	76
4.4.4 TCS34725 Color sensor	79
4.5 Color calibration methodology for an automated sorting system.....	81
4.5.1 Primary color testing using raw RGB thresholds	82
4.5.2 Enhanced calibration using Hex values in Processing IDE	83
4.6 Full system's circuit design	84
4.7 Mechanical structure and material choices	86
4.8 Mechanical validation	88
4.8.1 Conveyor belt functionality	88
4.8.2 Drive mechanism testing	89
4.9 Electronic component calibration	89
4.9.1 Servomotor precision	89
4.10 Software and control systems	89
4.10.1 Web application implementation	89
4.10.2 LCD Feedback logic	91
4.10.3 Manual mode efficiency	91
4.10.4 Remote control advantages.....	92
4.11 Lessons learned and improvements	92

CONCLUSION

Conclusion.....	93
-----------------	----

Liste of figures

Chapter 1: Automation of production lines

Figure 1.1: Operation of a production line	4
Figure 1.2: Pharmaceutical production line	6
Figure 1.3: Painting robots.....	8
Figure 1.4: Quality control with sensor	9
Figure 1.5: Automated packaging line.....	10
Figure 1.6: Waste sorting system	11

Chapter 2: Hardware design

Figure 2.1: Arduino uno.....	14
Figure 2.2: UART Communication	16
Figure 2.3: UART Interface	16
Figure 2.4: Arduino SPI.....	18
Figure 2.5: SPI Protocol.....	18
Figure 2.6: SPI bus	19
Figure 2.7: SDA-SCL signal graph	21
Figure 2.8: I2C Bus communications	22
Figure 2.9: Different types of Arduino boards	25
Figure 2.10: L298N Driver schematic diagram	26
Figure 2.11: 25%, 50%, and 90% Duty Cycle Examples.....	29

Figure 2.12: 25%, 50%, and 75% Duty Cycle Examples.....	29
Figure 2.13: Schematic of an H-Bridge	30
Figure 2.14: L298N in a Line-Following Robot.....	31
Figure 2.15: L298N Pinout.....	32
Figure 2.16: L298N Direction	33
Figure 2.17: L298N Arduino Hookup.....	34
Figure 2.18: DC Gear-motors.....	36
Figure 2.19: Servomotor	38
Figure 2.20: The draft of the servomotor	40
Figure 2.21: The inside of the servomotor	40
Figure 2.22: TCS34725 Color Sensor.....	42
Figure 2.23: TCS34725 Pinout Diagram	43
Figure 2.24: Functional Block Diagram of TCS34725	44
Figure 2.25: Color recognition sensor (CMOS technology)	44
Figure 2.26: TCS34725 Photodiode Spectral Responsivity RGBC	45
Figure 2.27: 16x2-Character-LCD-5x8-Pixel-Matrix-Grid	47
Figure 2.28: Liquid Crystal technology	48
Figure 2.29: PCF8574 I2C LCD adapter	49
 Chapter 3: Control logic and circuit simulation	
Figure 3.1: Arduino IDE interface.....	53
Figure 3.2: Selecting Arduino Board (STEP1).....	54
Figure 3.3: Selecting Port (STEP2).....	55

Figure 3.4: Program structure.....	56
Figure 3.5: Declaring variables, button (pin 2), LED (pin 13)	57
Figure 3.6: Void setup().....	58
Figure 3.7: Void loop().....	59
Figure 3.8: Selecting Library manager	61
Figure 3.9: Installing wanted library.	61
Figure 3.10: Verification of library.	62
Figure 3.11: Verification & uploading sketch.	63
Figure 3.12: Tinkercad overall interface.	64
Figure 3.13: Tinkercad circuit design interface.	65
Figure 3.14: STEP1 Circuit design.	67
Figure 3.15: STEP2 Write or import code.....	67
Figure 3.16: Start the simulation & check for Wiring/Code errors.	68

Chapter 4: Implementation & Testing

Figure 4.1: System's Logigram.....	71
Figure 4.2: Tinkercad simulation.	73
Figure 4.3: Servomotors wiring.	74
Figure 4.4: Gearmotor+L298N Wiring.	75
Figure 4.5: LCD I2C Wiring.....	76
Figure 4.6: I2C Adress scanner code.....	77
Figure 4.7: LCD I2C Adress.....	78
Figure 4.8: LCD I2C Display test.	78
Figure 4.9: TCS34725 Wiring.	79

Figure 4.10: TCS34725 I2C Adress.....	80
Figure 4.11: TCS34725 Test.....	81
Figure 4.12: Calibration circuit design.....	82
Figure 4.13: Hardware implementation.....	83
Figure 4.14: Calibration via Processing interface.....	83
Figure 4.15: Lightness, Hue, Saturation.....	84
Figure 4.16.: Final circuit design	85
Figure 4.17: Conveyor body structure.....	87
Figure 4.18: PU Belt material.	88
Figure 4.19: Toothed belt and pulleys drive system.	88
Figure 4.20: Servos angle calibration.....	89
Figure 4.21: Web application user interface.....	90
Figure 4.22: Color sorting system communication architecture.....	90
Figure 4.23: LCD Feedback screen.....	91
Figure 4.24: Manual mode.....	91

Liste of tables

Table 2.3.1: Key features of Arduino uno.....	15
Table 2.4.4: Protocols Comparaison.	23
Table 2.4.7: Arduino boards features.....	25
Table 2.6.2.1: MG5010 technical characteristics.....	39
Table 2.8.6: PCF8574 Pin configuration.....	50
Table 4.6: Pin configuration and wiring connections of the components.	84

Glossary

IOT: Internet of things.

USB: Universal Serial Bus.

ICSP: In-Circuit Serial Programming.

IDE: Integrated Development Environment.

UART: Universal Asynchronous Receiver Transmitter.

PWM: Pulse Width Modulation.

I2C: Inter-Integrated Circuit. **SPI :** Serial Peripheral Interface.

E/S: Entrées/Sorties.

Tx-Rx: Transmission et Réception.

EEPROM: Electrically-Erasable Programmable Read-Only Memory

OUT: OUTPUT.

GND: Ground.

VCC, VDD: Alimentation Tension Continue.

CMOS: Complementary Metal Oxide Semiconductor.

TTL: Transistor-Transistor Logic.

RPM: Revolutions per minute.

IR: Infrared.

LED: Light Emitting Diode.

LCD: Liquid Crystal Display.

SDA: Serial Data Line.

SCL: Serial Clock Line.

AC: Alternating current.

DC: Direct current.

VDD: Voltage at drain.

VCC: Voltage at collector.

MCU: Microcontroller Unit.

MPU: Motion Processing Unit (or) Microprocessor Unit.

PU: Polyethurane.

VIN: Voltage input.

LAN: Local Area Network.

HTTP: HyperText Transfer Protocol.

Introduction

Introduction

In recent years, industrial production and distribution lines have become increasingly automated. Automation is mainly applied in the manufacturing sector to handle complex, arduous, and repetitive tasks, such as assembly, welding, painting, selective sorting, material handling, and managing hazardous substances (e.g., in nuclear industries). It also enables operations in environments that are inaccessible to humans.

Some tasks within a production line are still manual or semi-automated, creating challenges and safety risks for workers in certain situations. This inevitably results in reduced productivity and lower product quality. As a result, it negatively impacts the company's economic growth, particularly in terms of expanding market share or profitability. Additionally, the growth of e-commerce demands a more efficient distribution process to cater to a larger customer base, with thousands of transactions being carried out on online platforms. This requires highly optimized automated systems that surpass human capabilities.

In this context, our project aims to design and implement a selective sorting system positioned in the middle of a production line or at the end. The sorting is based on the color of the products. To achieve this, we used a color sensor, servomotors, and a conveyor belt. The entire system is controlled by an Arduino Uno board.

To better present our work, this report is divided into four chapters. The first chapter highlights the importance of automation in production lines. Although the issue of sorting in a distribution chain is quite similar, this study focuses primarily on production lines. The second chapter discusses the hardware design of our system and the various steps required for its implementation. The third chapter focuses on the software design, simulation and the programs that enable the system to operate. Finally, the fourth chapter is exclusively dedicated to illustrating the practical implementation.

Chapter 1

Automation of production lines

1.1 Introduction

This chapter highlights the significance of automating production and distribution lines. Automation enhances productivity and improves the quality of services by efficiently handling tasks that are repetitive, labor-intensive, hazardous, or difficult for humans to perform manually.

1.2 Definition of production lines

A production line is a set of sequential operations established in a factory where components are assembled to make a finished article or where materials are put through a refining process to produce an end-product that is suitable for onward consumption.

Typically, raw materials such as metal ores or agricultural products such as foodstuffs or textile source plants like cotton and flax require a sequence of treatments to render them useful.[1] Figure 1.1 illustrates the operating principle of a production line.

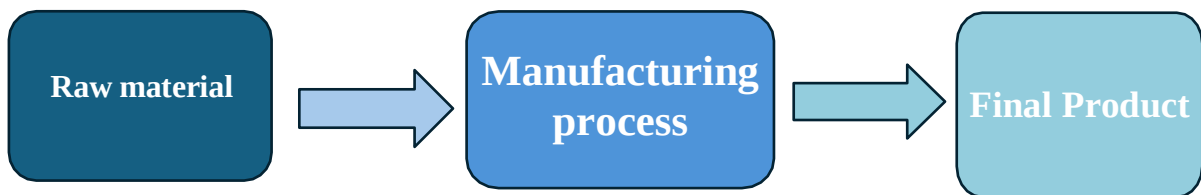


Figure 1.1. Operation of a production line.

1.3 Historical evolution of the production line

The 20th and 21st centuries have witnessed significant industrial progress, with manufacturing evolving through four distinct industrial revolutions (Industry 1.0 to 4.0). A key challenge for businesses has been maintaining a balance between supply and demand within value chains, influenced by factors such as production volume, product variety, delivery time, quality, pricing, and design.

Industry 1.0 (18th–19th centuries) marked the transition from an agricultural to an industrial economy, where production was primarily focused on increasing output. Supply often failed to meet demand, leading to price fluctuations as described by Adam Smith in *Wealth of nations*. The market was relatively simple, with limited product options available.

Industry 2.0 (late 19th century–1980s) introduced innovations such as electricity, automobiles, and mechanized production. This era saw an expansion in both product volume and variety, creating a more structured and predictable market. Key figures such as Frederick Taylor, through Scientific Management, contributed to improving efficiency, while Henry Ford revolutionized mass production with assembly lines. Additionally, Taiichi Ohno’s Toyota Production System (TPS) laid the foundation for modern lean manufacturing, addressing growing consumer demand for product variety.

Industry 3.0 (1980s–present) brought about a digital transformation, shifting from analog to digital technology and enabling modular product designs. With reduced product life cycles and evolving consumer needs, demand became more dynamic, now considering volume, variety, and delivery time. As a result, traditional manufacturing methods became less effective, leading to the adoption of flexible manufacturing systems (FMS) and lean production techniques to manage market volatility.

Industry 4.0, the current and emerging phase, is characterized by the integration of advanced technologies such as the Internet of Things (IoT), artificial intelligence (AI), big data, 3D printing, and cyber-physical systems. These innovations aim to enhance efficiency and automation within manufacturing, though uncertainties remain regarding future consumer demands and evolving product architectures, particularly in industries like electric vehicles. Figure 1.2 shows a pharmaceutical production line.



Figure 1.2. *Pharmaceutical production line.* [1]

1.4 Automation of production lines

1.4.1 Definition of automation

Automation describes a wide range of technologies that reduce human intervention in processes, mainly by predetermining decision criteria, subprocess relationships, and related actions, as well as embodying those predeterminations in machines. Automation has been achieved by various means including mechanical, hydraulic, pneumatic, electrical, electronic devices, and computers, usually in combination. Complicated systems, such as modern factories, airplanes, and ships typically use combinations of all of these techniques. The benefit of automation includes labor savings, reducing waste, savings in electricity costs, savings in material costs, and improvements to quality, accuracy, and precision. [2]

1.4.2 Automation of certain tasks in a production line

Automation plays a crucial role in modern production systems by increasing efficiency, ensuring consistency, and reducing human exposure to hazardous tasks. Despite these advancements, some production processes still rely on manual or semi-automated operations, which can introduce inefficiencies, safety risks, and

inconsistencies in product quality. Limited automation can hinder productivity and slow economic growth for businesses, particularly in competitive industries where innovation and speed are key. To remain competitive, companies must integrate advanced automation solutions to enhance production, improve workplace safety, and meet evolving customer demands, ultimately driving profitability and long-term success.

We will explore various applications and manual tasks that can be automated:

- Painting and coating;
- Quality control;
- Assembly;
- Selective sorting.

1) **Painting and coating**

A coating is a covering that is applied to the surface of an object, or substrate. The purpose of applying the coating may be decorative, functional, or both. Coatings may be applied

as liquids, gases or solids e.g. powder coatings.[3] Painting and coating are essential in industries such as automotive, aerospace, and construction, serving both aesthetic and protective functions. Coatings enhance durability by preventing corrosion, moisture damage, and wear while also offering advanced properties like antibacterial protection, self-cleaning surfaces, and thermal insulation. However, traditional manual painting methods can lead to inconsistencies, material waste, and health risks due to toxic fumes.

To address these challenges, automated painting systems have become increasingly prevalent. Using robotic arms and advanced spray technologies, these systems ensure precise application, reduce overspray, and optimize material usage. Modern automation allows for accurate control of factors such as spray width, pressure, and atomization air, leading to

consistent quality and reduced environmental impact by minimizing volatile organic compound (VOC) emissions.

The integration of AI and smart monitoring in painting automation further enhances efficiency. AI-powered systems can adjust to different surface geometries, optimize paint distribution in real-time, and detect defects early in the process. As industries push for greater sustainability and productivity, automated painting solutions continue to evolve, offering improved precision, cost savings, and enhanced worker safety while meeting stringent environmental regulations. Figure 1.3 shows a painting robot.



Figure 1.3. *Painting robots*

2) **Quality control**

Quality control has long been the guardian at the gates of brand reputation and customer satisfaction. However, as industries evolve and consumer demands intensify, traditional manual methods have started showing their limitations. But throughout this puzzle emerges a solution both elegant and robust that is the automated quality control (AQC).

Automated quality control is the technological regime that employs smart machines, sophisticated sensors, and advanced algorithms to inspect products without human intervention diligently. It holds immense importance as it transcends human speed and precision capabilities, significantly multiplying output while upholding and often enhancing the stringent standards that signify superior quality. The integration of automated quality control across various industries is reshaping the manufacturing landscape by leaps and

bounds. The powerful benefits it brings to the table are undeniable, revolutionizing not only how products are made but also ensuring that they meet high standards of quality with unparalleled consistency. Furthermore, better product inspections mean safer products reaching consumers crucial in fields like automotive or pharmaceuticals where safety isn't just a preference but a stringent requirement.

The infusion of AI into automated quality control marks a significant step towards intelligent manufacturing. This merging heralds a future where systems do more than follow programmed instructions but they learn and adapt. [4] Figure 1.4 shows an automated quality control with sensor.



Figure 1.4. *Quality control with sensor.*

3) **Packaging and conditioning**

The packaging and conditioning of materials and final products (such as food, pharmaceutical, and household appliances) incorporate climatic, physico-chemical, and mechanical protections to ensure product safety during handling, preservation, storage, and transportation (against degradation agents). Robotics is the ideal solution to meet the industry's demands, as it offers great flexibility through a wide range of robots with various morphologies, capable of adapting to all applications.

Packaging robots are typically used around a conveyor-equipped production line with vision systems. Packaging includes various operations such as depalletization,

wrapping, boxing, palletization, casing, and picking. [5]

Automated packaging and conditioning systems have transformed manufacturing and logistics by improving efficiency, precision, and product safety. These systems leverage robots, AI, and smart sensors to handle tasks like wrapping, labeling, sorting, and palletizing, reducing human effort and increasing overall productivity. Figure 1.5 illustrates an automated packaging line.

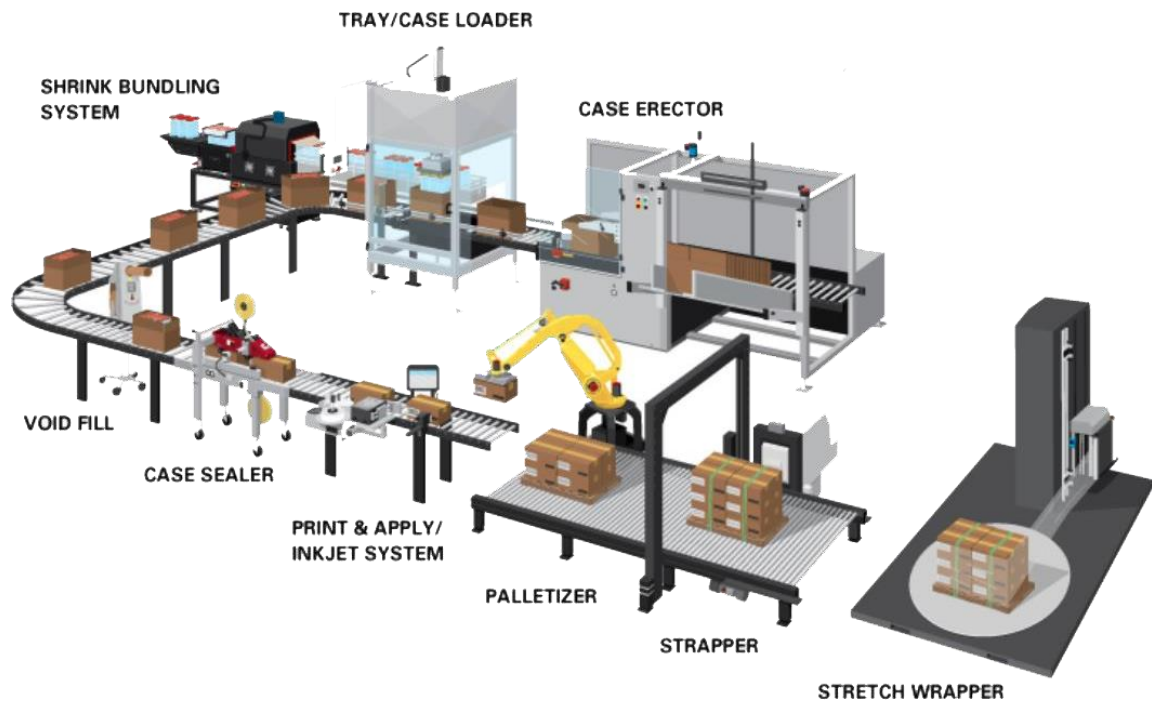


Figure 1.5. *Automated packaging line*

4) **Selective sorting**

Selective sorting is the process of classifying products, raw materials, or components in a production line or logistics center based on specific criteria such as material type, size, shape, or color. This is particularly valuable in manufacturing, recycling and waste management, pharmaceutical industry, and warehouse management.

Automated sorting systems are essential for high-capacity manufacturing and distribution companies, as they help enhance quality and reliability in operations. By leveraging AI, robotics, and sensor technology, these systems ensure fast, accurate, and efficient classification of materials and products. This leads to improved workflow efficiency, reduced errors, and

optimized resource utilization, making them a crucial component in modern industrial processes. Figure 1.6 illustrates a Waste sorting system.



Figure 1.6. *Waste sorting system*

1.5 Work Presentation (Problem Statement)

Our study focuses on automated selective sorting at the middle or at the end of a production line, based on the visible color of a product. The main objective of our system is to classify products into three categories: red, blue, and a third category for green and yellow. Once sorted, each product is directed into a dedicated storage bin, packaging unit, or bag according to its color.

To implement an efficient color-based sorting system, we must address two fundamental questions:

- How can the system accurately detect the product's color?
- What mechanism should be used to sort and direct the product once its color has been identified?

1.6 Discussion

This chapter highlighted the benefits and significance of automation in production lines. Our project involves developing an automated and high-performance selective sorting system. The next chapter will detail the hardware aspects of the system, explaining the functionality of each component.

Chapter 2

Hardware Design

2.1 Introduction

With the advancements in modern technology, a vast array of electronic components is available, allowing for the development of countless machines, devices, and multifunctional robots. This chapter examines the key components, sensors, and actuators used in our project, detailing their functionality and operational principles.

An embedded system is equipped with perceptual capabilities that allow it to sense certain aspects of its environment. It also includes effectors that enable it to interact with and influence that environment. The system's behaviour can typically be described as a function that maps sensory inputs to effector actions. This function may include an internal state, meaning the output can depend not only on current inputs but also on the entire history of past inputs. The system continuously monitors its surroundings and responds with appropriate actions based on this mapping.

2.2 Development Boards

Development boards are electronic boards used for designing, prototyping, and developing embedded applications. They typically integrate a microcontroller or microprocessor along with input/output interfaces, making it easier to connect various sensors, actuators, and communication modules.

Development boards are just printed circuit boards with either a microcontroller or a microprocessor mounted on them and with various metal pins that an experienced developer can use to develop, prototype, and iterate different versions of various smart products. They're also used by hobbyists to help them become more familiar with development concepts and hardware by for example creating a small security camera or other basic IoT device. [6]

The first microcontroller development board to capture the attention of the engineering community was released in 2006. This prototyping platform, which later became known as Arduino (Figure 1), was quickly adopted by a new category of electronic designers that included enthusiasts, hobbyists, and DIY engineers. Arduino laid the foundations for the commercial success of later SBCs

and microcontroller-based platforms and was soon followed in 2008 by Beagle Board, which provided engineers with a low-cost, open-source community-supported development platform. In 2012, the Raspberry Pi the first modern single-board computer was released. [7]

2.3 Arduino Uno Board

The Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, 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 an AC-to-DC adapter or battery to get started. You can tinker with your Uno without worrying too much about doing something wrong; in the worst-case scenario, you can replace the chip for a few dollars and start over again. [8]

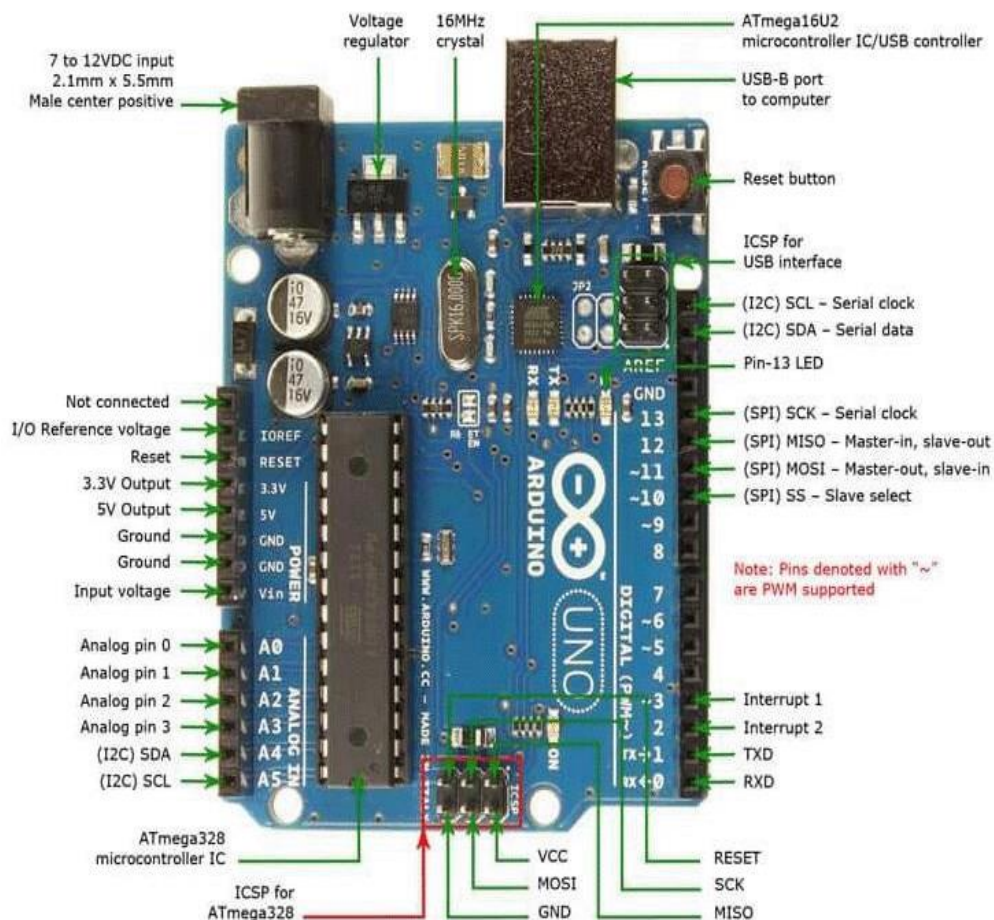


Figure 2.1. Arduino Uno [9]

2.3.1 Key Features

Table 2.3.1. *key features of Arduino uno*

Category	Details
Power Pins	<ul style="list-style-type: none"> - 7 to 12VDC input: 2.1mm x 5.5mm barrel jack (center-positive) - 3.3V & 5V outputs: For powering external components - Ground (GND): Multiple GND pins for easy connectivity
Digital I/O Pins	<ul style="list-style-type: none"> - Total: 14 digital I/O pins - PWM Support: 6 pins (marked with ~) - Common Uses: Motors, sound, LED dimming - Pin 13: Built-in LED for quick testing
SPI Pins	<ul style="list-style-type: none"> - SCK (Serial Clock) - MISO (Master-In, Slave-Out) - MOSI (Master-Out, Slave-In) - SS (Slave Select)
Analog Input Pins	<ul style="list-style-type: none"> - Total: 6 pins (A0–A5) for analog sensors - A4 (SDA) & A5 (SCL): I2C communication
Communication Ports	<ul style="list-style-type: none"> - USB-B Port: Upload code and power the board - TX & RX Pins: Serial communication
Other Notable Components	<ul style="list-style-type: none"> - Reset Button: Restarts the code - ATmega16U2: USB-to-serial conversion - 16MHz Crystal Oscillator: Ensures precise timing - Voltage Regulator: Provides stable power
ICSP Headers	<ul style="list-style-type: none"> - Atmega328: VCC, MOSI, GND, RESET, SCK, MISO - USB Interface: Advanced USB-specific applications

2.4 Arduino Communication Protocols

Arduino supports several communication protocols to interface with sensors, displays, and other peripherals. These protocols enable data exchange between the Arduino board and external devices. Below is a detailed overview of the most common Arduino communication protocols, including UART, I2C, SPI, and 1-Wire, along with their key features, advantages, and disadvantages.

2.4.1 UART Communication Protocol

UART is one of the most used device-to-device (serial) communication protocols. It's the protocol used by Arduino boards to communicate with the computer. It allows asynchronous serial communication in which the data format and transmission speed are configurable. [10]

How UART Works

UART operates by transmitting data as a series of bits, including a start bit, data bits, an optional parity bit, and stop bit(s). Unlike parallel communication, where multiple bits are transmitted

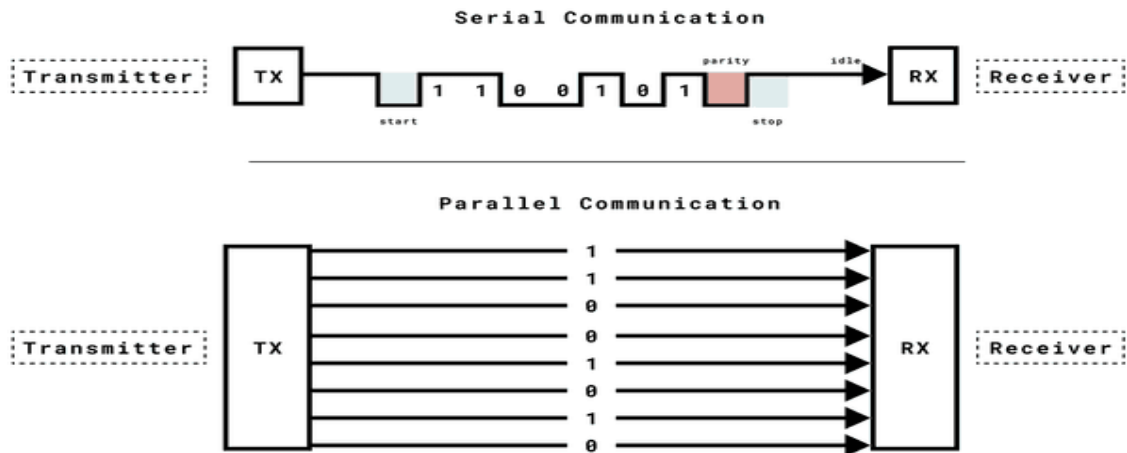
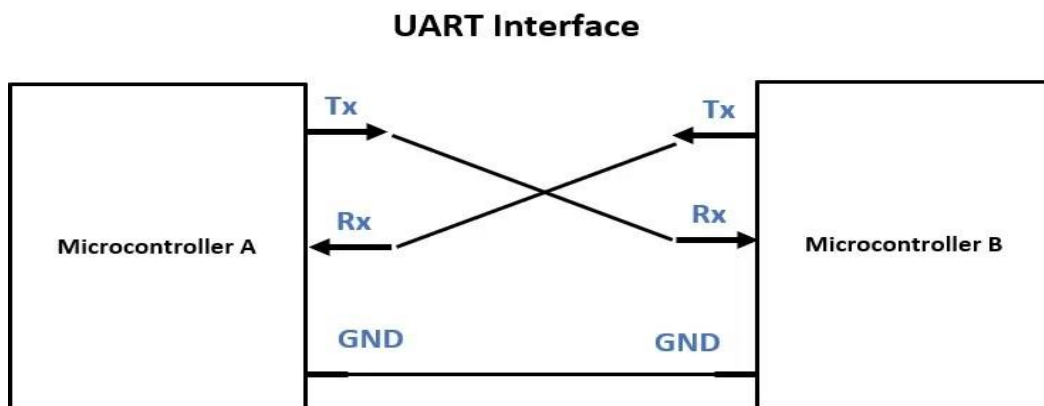


Figure 2.2. UART Communication [10]

simultaneously, UART sends data serially, one bit at a time. As the name reveals, the protocol operates asynchronously, meaning it doesn't rely on a shared clock signal. Instead, it uses predefined baud rates to determine the timing of data bits. [10]

The key components of UART include the transmitter, receiver, and baud rate. The transmitter collects data



www.ucbeginner.com

Figure 2.3. UART Interface [10]

As seen in the image above when using parallel communication an 8-bit message would require eight cables while serial communication only requires one cable for sending messages and one for receiving.

from a source, formats it into serial bits, and sends it via a TX (Transmit) pin. The receiver receives it via a RX (Receive) pin, processes incoming serial data and converts it into parallel data for the host system. The baud rate determines the speed of data transmission. [10]

Advantages of using UART

- Only uses two wires
- No clock signal is necessary
- Has a parity bit to allow for error checking
- The structure of the data packet can be changed as long as both sides are set up for it
- Well documented and widely used method [11]

Disadvantages of using UART

- The size of the data frame is limited to a maximum of 9 bits
- Doesn't support multiple slave or multiple master systems
- The baud rates of each UART must be within 10% of each other [11]

2.4.2 Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. [12] SPI serves as the main full-duplex, synchronous communication method. It is widely used to share data between microcontrollers, sensors, displays, and peripherals. SPI connects a master and multiple slave devices, transferring data sequentially. [13]

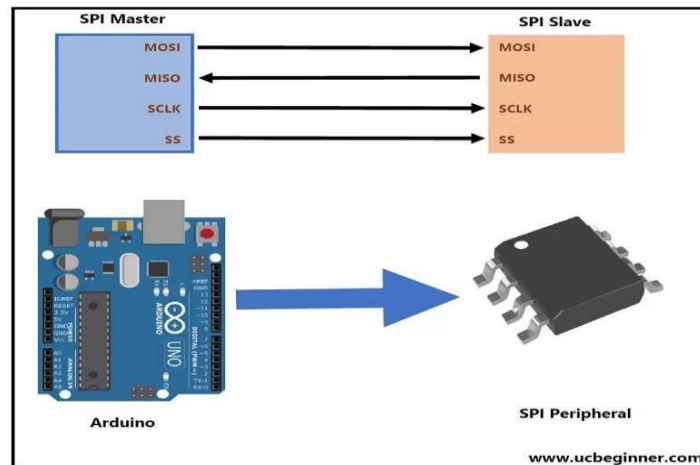


Figure 2.4. Arduino SPI [13]

SPI Protocol and Interface

In SPI communication, there are four key signals:

- **MISO (Master In Slave Out):** Data travels from the slave to the master.
- **MOSI (Master Out Slave In):** Data flows from the master to the slave.
- **SCK (Serial Clock):** The master generates this necessary clock signal.
- **SS/CS (Slave Select/Chip Select):** It isolates the desired slave device in scenarios with multiple SPI-sharing devices.
- Additionally, the SPI interface operates in different modes, characterized by clock phase (CPHA) and polarity (CPOL). [13]
-

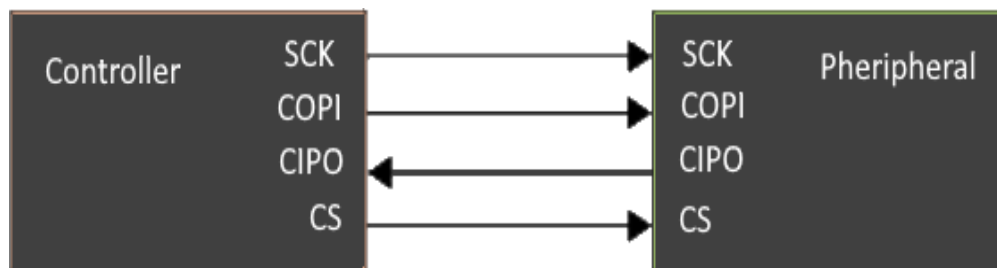


Figure 2.5. SPI Protocol [14]

Understanding SPI Modes and Their Significance

SPI mode selection (CPOL and CPHA) is of great importance, as it controls when data is captured and which clock signal is transmitted. Let's dive deeper into each mode:

- **SPI Mode 0 (CPOL=0, CPHA=0):**
 - The clock stays low during idle.
 - Data is captured on the leading (rising) edge.
- **SPI Mode 1 (CPOL=0, CPHA=1):**
 - The clock stays low during idle.
 - Data is captured on the trailing (falling) edge.
- **SPI Mode 2 (CPOL=1, CPHA=0):**
 - The clock stays high during idle.
 - Data is captured on the leading (rising) edge.
- **SPI Mode 3 (CPOL=1, CPHA=1):**
 - The clock stays high during idle.
 - Data is captured on the trailing (falling) edge. [13]

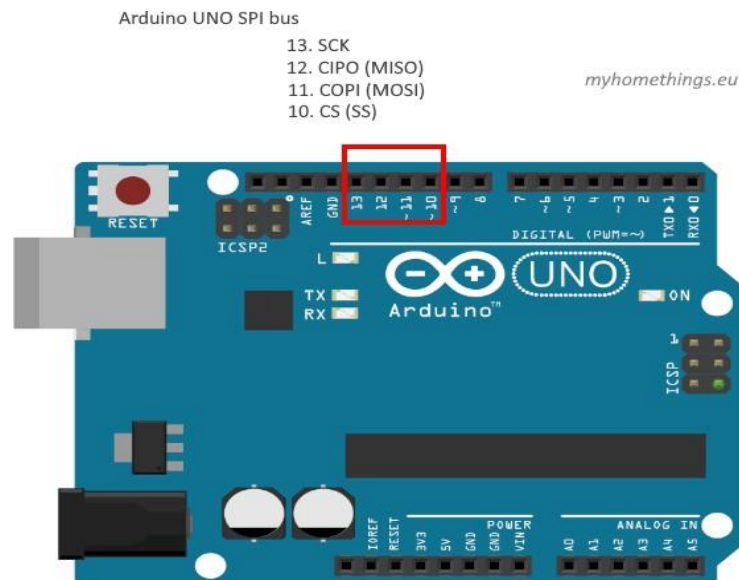


Figure 2.6. SPI bus [14]

Choosing the appropriate SPI mode depends on the unique demands of your project. For example, when connecting to a specific sensor or display, refer to the device data sheet for the recommended SPI mode. A wrong choice can lead to communication errors. [13]

Advantages of using SPI

- It is a very simple protocol, requiring minimal processing overhead. Designers primarily need to understand the read/write timing diagrams of the microcontroller/EPROM to use it effectively.
- It supports full-duplex communication.
- SPI uses a push-pull configuration, enabling higher data rates and support for longer distances.
- It consumes less power compared to the I2C interface.
- Data can be transferred at high speeds, in the tens of MHz range. [15]

Disadvantages of using SPI

- It requires one CS (Chip Select) line per slave device. This increases hardware complexity when multiple slave devices are present in the design.
- Adding a slave device necessitates software changes and the addition of an extra CS line.
- SPI lacks flow control and acknowledgment mechanisms.
- The master and slave relationships are fixed and cannot be altered dynamically, unlike I2C.
- The master may not inherently know whether a slave device is present or absent. Handshaking is typically required using software to address this. [15]

2.4.3 I2C Communications

I2C is a serial protocol used on a low-speed 2-wire interface. It was originally developed by Phillips in 1982 to allow integrated circuits within television receivers to communicate with one another. [16]

I2C bus has two lines, one is a data line (SDA) and another is a clock line (SCL). There are three kinds of signals when communicating, Start signal, Stop signal, and Answer signal.

Start Signal: When SCL is High, SDA change from High to Low, it start to transmit data
Stop Signal: When SCL is High, SDA change from Low to High, it stops transmitting.

Answer Signal: Every time IC sends back a certain Low plus to the sender after it receives 8 bits of data. [17]

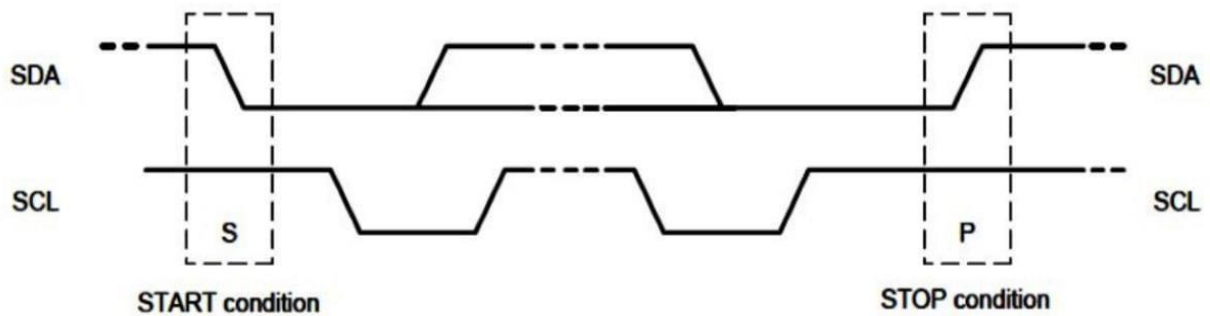


Figure 2.7. SDA–SCL signal graph [17]

When using an Arduino for any project, one of the main areas of concern is the limited inputs and outputs (I/O). For the Arduino used in this application note, there are 13 digital I/O and 6 analog I/O. When connecting one LCD using normal communication methods, the LCD requires 6 digital pins, leaving only 7 for other desired functions. Using I2C, it is possible to use multiple LCDs, digital-to-analog converters, and a multitude of sensors with only two total digital pins. [18]

How I2C Works

An I2C bus has two signals, along with a power and ground connection:

SDA (Serial Data Line): Transmits data between devices.

SCL (Serial Clock Line): Synchronizes data transmission. [16]

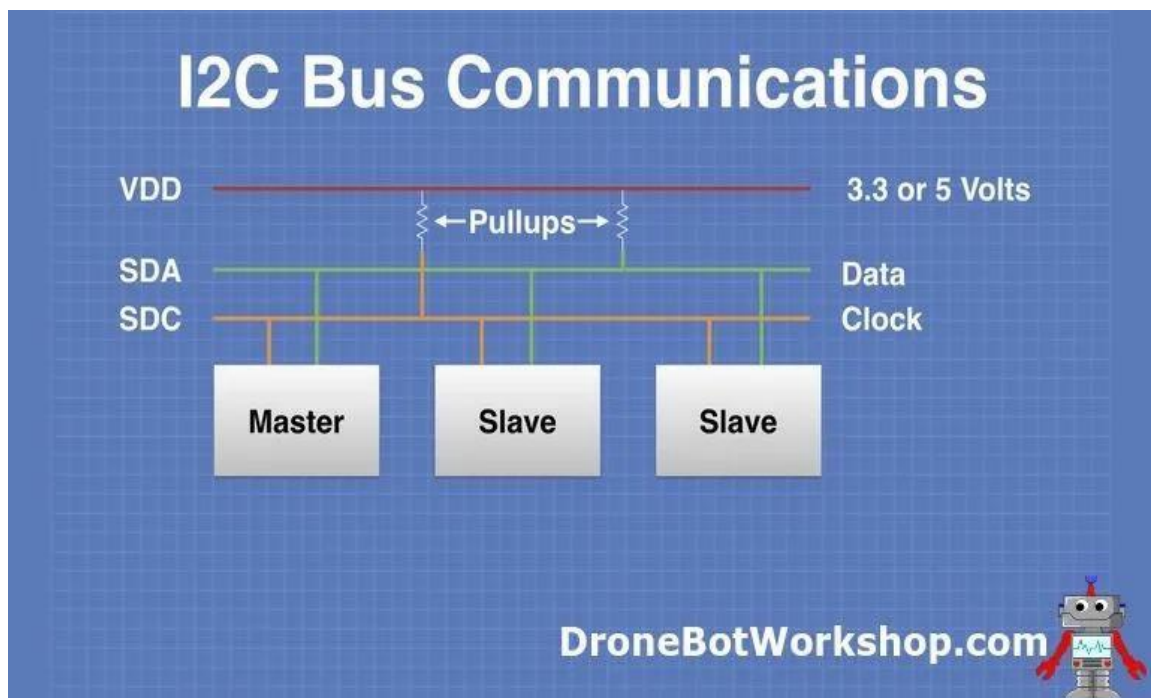


Figure 2.8. *I2C Bus Communications* [16]

Advantages of using I2C

Flexibility – The I2C protocol supports multi-master, multi-slave communication, which implies you can add a lot of functionality to your design. More than one master IC controlling and communicating with the slave ICs can speed things up and add functionalities to the embedded system.

Addressing feature – Yet another advantage of the I2C protocol lies in its inherent ability to use chip addressing. It means that you can easily add components to the bus without any complexity. It eliminates the necessity of CS (chip select) lines.

Simplicity – I2C protocol doesn't complicate the design. It requires only two bidirectional signal lines to establish communication among multiple devices. Further, the pin count is low as well.

Better error handling mechanism – To improve the error detection and correction mechanism, the I2C protocol relies on ACK/NACK feature, which is a robust error correction feature. ACK stands for Acknowledgement whereas NACK means No Acknowledgement.

Adaptable – The I2C protocol is adaptable in the sense that it can work well with both slow ICs and fast ICs. [19]

Disadvantages of using I2C

Conflicts – Due to chip addressing, there's always a possibility of an address conflict. **Slower speeds** – I2C protocol uses pull-up resistors rather than the push-pull ones used by its peers. Due to the open-drain design, the speed is limited.

Requires more space – Now, as an embedded system engineer, you know how valuable PCB real estate is. So, it isn't such a positive attribute that the I2C protocol requires so much space for its pull-up resistors. [19]

2.4.4 Comparison of UART, I2C and SPI

Table 2.4.4: *Protocols Comparison*

Protocol	Complexity	Speed	# of Devices	# of Wires	Duplex	No. of master and slave
UART	Simple	Slowest	Up to 2 devices	1	Full Duplex	Single to Single
I2C	Easy to chain multiple devices	Faster than UART	Up to 127, but gets complex	2	Half Duplex	Multiple slaves and master
SPI	Complex as device increases	Fastest	Many, but gets complex	4	Full Duplex	1 master, multiple slaves

2.4.5 Advantages of Arduino Boards

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. [8]

2.4.6 Applications in Prototyping and Research

An Arduino board (such as the Uno) connected to a breadboard with plugins such as inputs, sensors, lights, and displays can be controlled by a code written in the Arduino development environment. How to achieve this is by prototyping with Arduino. Prototyping with Arduino has grown in popularity with the increased use of the Arduino platform. Prototyping with Arduino, however, is not an easy task for nonprogrammers with interest in the field. With increased public interest in the field will come a need for accessible information. [20]. Arduino Uno-based systems are used to measure air quality and soil moisture, in biomedical engineering, the Arduino Uno has been instrumental in prototyping devices such

as ECG monitors and prosthetics, demonstrating its potential to revolutionize healthcare research [21]. These diverse applications highlight the Arduino Uno's role as a critical tool in bridging the gap between theoretical research and practical implementation.

2.4.7 The features of different types of Arduino boards

The features of different types of Arduino boards are listed in the tabular form. [22]

Table 2.4.7. *Arduino Boards Features [22]*

Arduino Board	Processor	Memory	Digital I/O	Analogue I/O
Arduino Uno	16Mhz Atmega328	2KB SRAM, 32KB flash	14	6 inputs, 0 output
Arduino Due	84MHz AT91SAM3X8E	96KB SRAM, 512KB flash	54	12 inputs, 2 output
Arduino Mega	16MHz Atmega2560	8KB SRAM, 256KB flash	54	16 inputs, 0 output
Arduino Leonardo	16MHz Atmega32u4	2.5KB SRAM, 32KB flash	20	12 inputs, 0 output



Figure 2.9. *different types of Arduino Boards [23]*

2.5 The L298N Motor Driver

L298N is a classic and widely used dual H-bridge motor driver chip that can control two DC motors or one stepper motor at the same time. Its biggest feature is that it supports high voltage input, which is suitable for a variety of motor control scenarios, such as robots, automation equipment, and small industrial systems. [24]

This dual bidirectional motor driver, is based on the very popular L298 Dual H-Bridge Motor Driver Integrated Circuit. The circuit will allow you to easily and independently control two motors of up to 2A each in both directions. It is ideal for robotic applications and well suited for connection to a microcontroller requiring just a couple of control lines per motor. It can also be interfaced with simple manual switches, TTL logic gates, relays, etc. This board equipped with power LED indicators, on-board +5V regulator and protection diodes. [25].

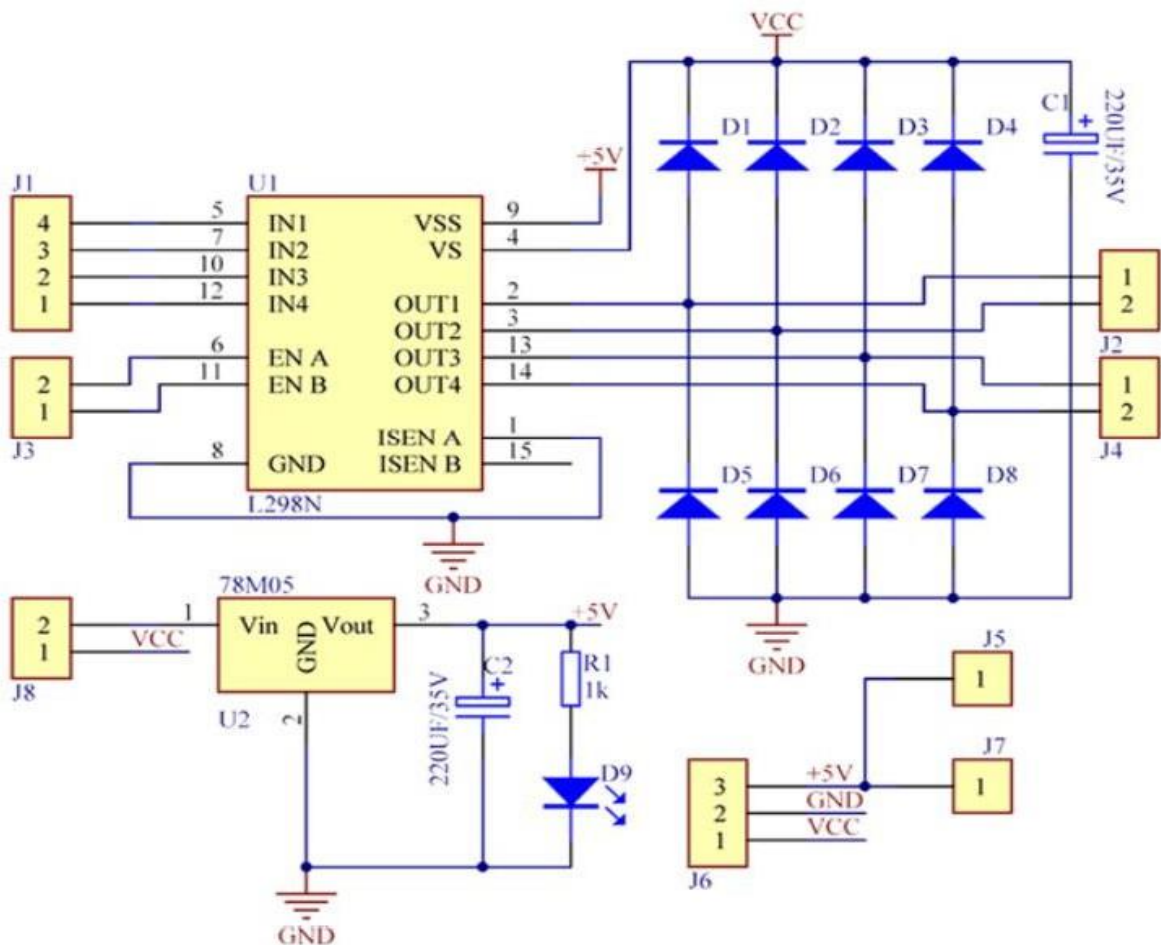


Figure 2.10: L298N Motor Driver Schematic Diagram [25]

Technical Specifications

Input Voltage:

3.2V~40Vdc.

Power Supply:

DC 5 V - 35 V

Peak current:

2 Amp

Operating current range:

0 ~ 36mA Control signal

input voltage range:

- Low: $-0.3V \leq V_{in} \leq 1.5V$.
- High: $2.3V \leq V_{in} \leq V_{ss}$.

Enable signal input voltage range:

- Low: $-0.3 \leq V_{in} \leq 1.5V$ (control signal is invalid).
- High: $2.3V \leq V_{in} \leq V_{ss}$ (control signal active).

Maximum power consumption: 20W (when the temperature $T = 75\text{ }^{\circ}\text{C}$). Storage temperature: $-25\text{ }^{\circ}\text{C} \sim +130\text{ }^{\circ}\text{C}$.

On-board +5V regulated Output supply (supply to controller board i.e. Arduino). Size: 4.3cm x 4.3cm x 2.7cm [25]

2.5.1 Operational Principles

The L298N chip integrates two complete H-bridge circuits inside, which can drive two DC motors independently or control a bipolar stepper motor in series. Its input voltage range is 5V to 46V, and the maximum drive current is 2A (single channel). In addition, L298N supports PWM (pulse width modulation) control, which can achieve motor speed regulation.

L298N is also equipped with a diode for circuit protection to prevent the reverse current generated by the motor when it stops or reverses from damaging the chip. This feature makes it more reliable in motor control applications. [24]

Main features of L298N

High voltage support:

The input voltage range of L298N is 5V to 46V, which enables it to drive motors of various specifications. Compared with some low-voltage drivers, it is suitable for more complex industrial applications.

Dual channel control

Each L298N chip can drive two motors independently, supporting bidirectional rotation and brake functions. When controlling stepper motors, precise direction and step control can be achieved.

Overheat protection:

L298N has built-in overheat protection. When the chip temperature is too high, it automatically limits the output to prevent burning.

Logic voltage and motor voltage separation:

L298N supports independent logic voltage (5V) and motor drive voltage, which makes it easy to integrate into different control systems without worrying about voltage incompatibility.

External heat sink support:

In order to meet the needs of high-power motors, the L298N chip has a metal heat sink on the top and supports the installation of an external heat sink to enhance the heat dissipation effect. [24]

2.5.2 Pulse Width Modulation (PWM)

One of the standout features of the L298N module is its ability to control motor speed using Pulse Width Modulation (PWM). By adjusting the duty cycle of the PWM signal connected to the enable pins (ENA and ENB), you can effectively regulate the average voltage delivered to the motor. This allows for smooth and precise speed control without the need to change the motor's power supply voltage. [26]

width modulation is a technique which allows us to adjust the average value of

the voltage that's going to the electronic device by turning on and off the power at a fast rate. The average voltage depends on the duty cycle, or the amount of time the signal is ON versus the amount of time the signal is OFF in a single period of time. [27]

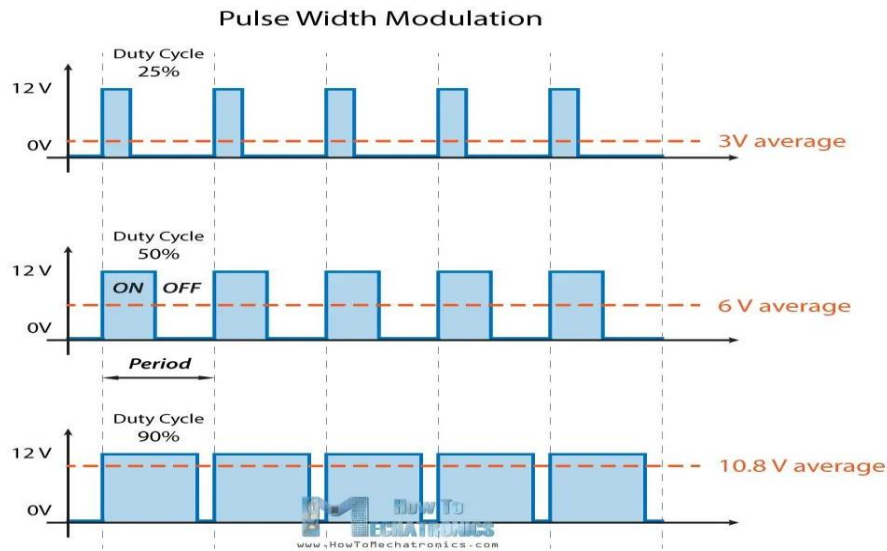


Figure 2.11. 25%, 50%, and 90% Duty Cycle Examples [27]

If a digital signal spends half of the time on and the other half off, we would say the digital signal has a duty cycle of 50% and resembles an ideal square wave.

If the percentage is higher than 50%, the digital signal spends more time in the high state than the low state and vice versa if the duty cycle is less than 50%.

Here is a graph that illustrates these three scenarios:

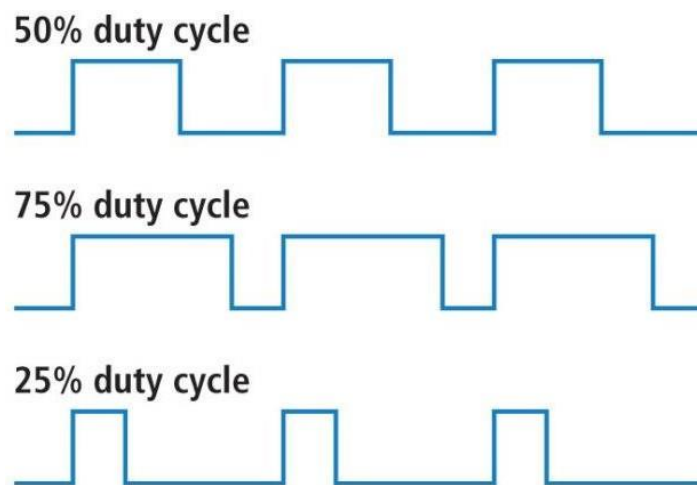


Figure 2.12. 25%, 50%, and 75% Duty Cycle Examples [28]

100% duty cycle would be the same as setting the voltage to 5 Volts (high). 0% duty cycle would be the same as grounding the signal. [28]

2.5.3 H-bridge Configuration

An **H-bridge** is an electronic circuit that switches the polarity of a voltage applied to a load. These circuits are often used in robotics and other applications to allow DC motors to run forwards or backwards. The name is derived from its common schematic diagram representation, with four switching elements configured as the branches of a letter “H” and the load connected as the cross-bar.

The term *H-bridge* is derived from the typical graphical representation of such a circuit. An H- bridge is built with four switches (solid-state or mechanical). When the switches S1 and S4 (according to the first figure) are closed (and S2 and S3 are open) a positive voltage is applied across the motor. By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

Using the nomenclature above, the switches S1 and S2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches S3 and S4. This condition is known as shoot-through. [29]

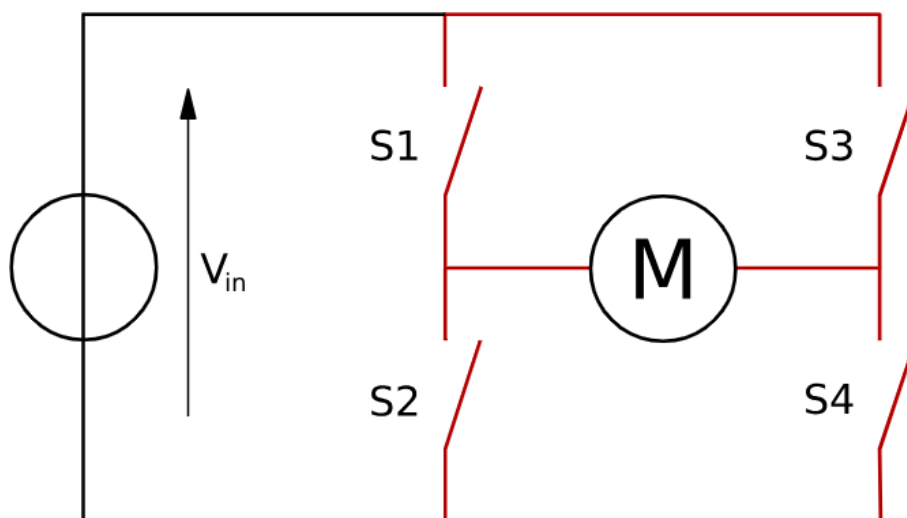


Figure 2.13. Schematic of an H-Bridge [29]

2.5.4 Applications

The versatility of the L298N motor driver has led to its widespread adoption in various fields:

Robot control

L298N is often used to control wheel motors in robots, supporting forward, backward and turning operations, and can adjust speed according to PWM signals.

Stepper motor drive

In equipment such as 3D printers and CNC machine tools, L298N can be used as a stepper motor driver to ensure precise control of the motor.

Industrial automation

In automation equipment, L298N is suitable for driving high-voltage, high-current motors, such as conveyor belt systems and industrial robotic arms.

Education and experiment

L298N has become the first choice for students and enthusiasts to learn motor control because of its ease of use and high-cost performance. [24]

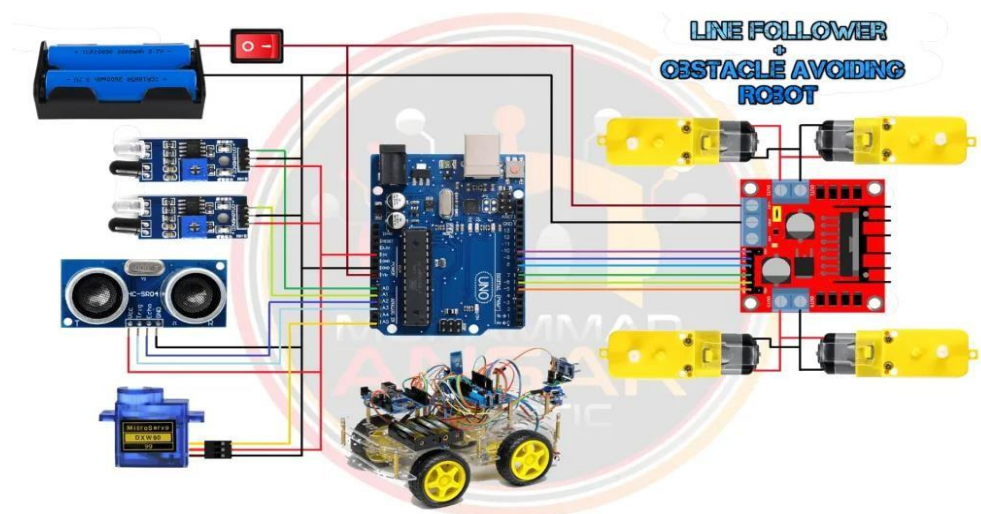


Figure 2.14. *L298N in a Line-Following Robot [30]*

2.5.5 L298N Pinout

The L298N module has 11 pins that allow it to communicate with the outside world. The pinout is as follows:

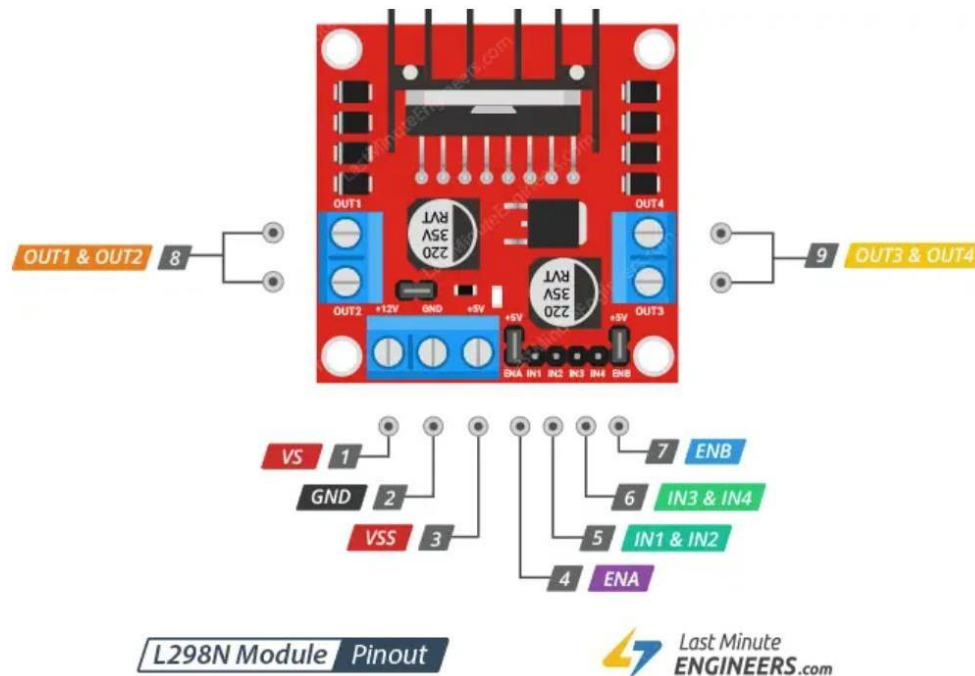


Figure 2.15. L298N Pinout [30]

Let's get acquainted with each pin one by one:

Power Pins:

The L298N motor driver module receives power from a 3-pin, 3.5mm-pitch screw terminal and it has two input power pins: VS and VSS.

VS pin powers the IC's internal H-Bridge, which drives the motors. This pin accepts input voltages ranging from 5 to 12V.

VSS is used to power the logic circuitry within the L298N IC, and can range between 5V and 7V. GND is the common ground pin.

Output Pins:

The output channels of the L298N motor driver, **OUT1** and **OUT2** for motor A and **OUT3** and **OUT4** for motor B, are broken out to the edge of the module with two 3.5mm-pitch screw terminals. You can connect two 5-12V DC motors to

these terminals. Each channel on the module can supply up to 2A to the DC motor. The amount of current supplied to the motor, however, depends on the capacity of the motor power supply.

Direction Control Pins:

The direction control pins allow you to control whether the motor rotates forward or backward. These pins actually control the switches of the H-Bridge circuit within the L298N chip. The module has two direction control pins. The IN1 and IN2 pins control the spinning direction of motor A; While IN3 and IN4 control the spinning direction of motor B. The spinning direction of the motor can be controlled by applying logic HIGH (5V) or logic LOW (Ground) to these inputs. The figure below shows various combinations and their outcomes.

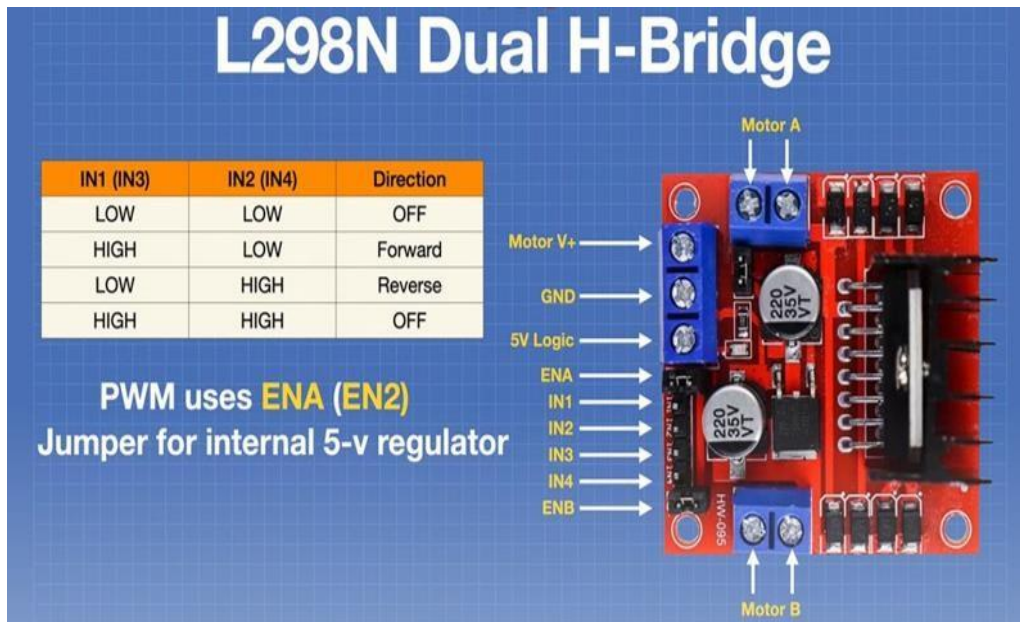


Figure 2.16. L298N Direction [31]

L298N Arduino Hookup:

You can hook up an L298N to an Arduino (or any other microcontroller) in several different ways, the only restriction is the *ENA* and *ENB* need to be attached to PWM-enabled pins.

Here is an example:

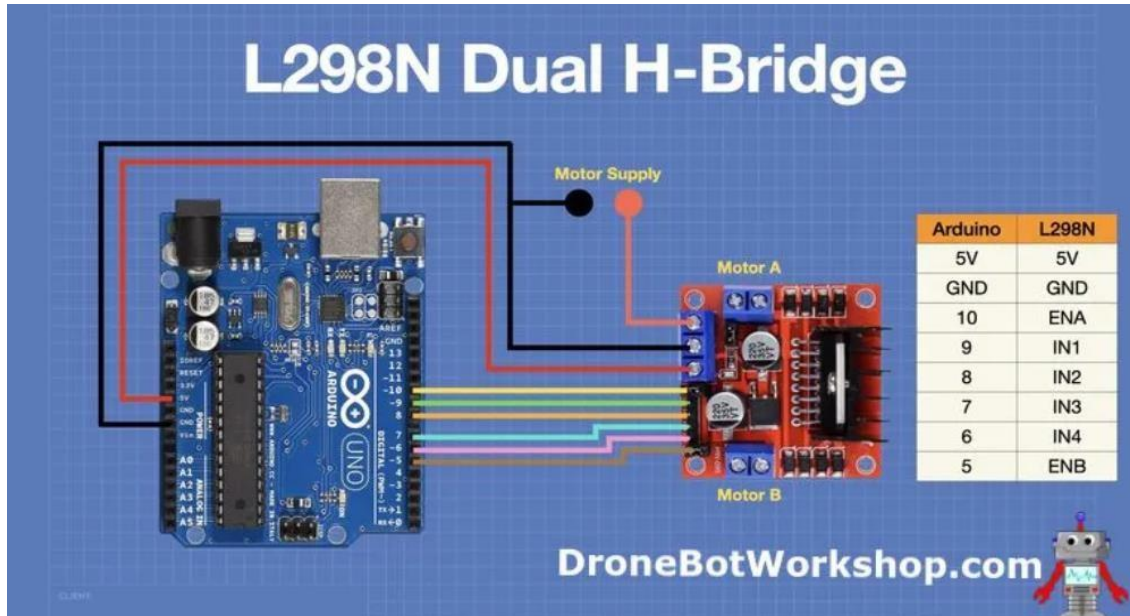


Figure 2.17. *L298N Arduino Hookup* [31]

Speed Control Pins:

The speed control pins ENA and ENB are used to turn on/off the motors and control their speed.

Pulling these pins HIGH will cause the motors to spin, while pulling them LOW will stop them. However, with Pulse Width Modulation (PWM), the speed of the motors can be controlled.

The module usually comes with a jumper on these pins. When this jumper is in place, the motor spins at full speed. If you want to control the speed of the motors programmatically, remove the jumpers and connect them to the Arduino's PWM-enabled pins.

Voltage Drop of L298N:

The L298N has a voltage drop of approximately 2V. This is due to the fact that internal switching transistors have a voltage drop of approximately 1V when forward biased, and because an H- Bridge requires the current to pass through two transistors, the total voltage drop is 2V.

So, if you connect 12V to the motor power supply terminal, the motors will receive approximately 10V. This means that a 12V DC motor will never spin at full speed. [32]

Understanding the pin configuration of the L298N is essential for designing motor control systems. For example:

In **robotics**, the L298N's direction control pins (IN1, IN2, IN3, IN4) are used to navigate robots by controlling wheel rotation.

In **industrial automation**, the enable pins (ENA, ENB) are used to adjust conveyor belt speeds using PWM signals.

In **educational projects**, the L298N's straightforward pin layout makes it an ideal tool for teaching motor control principles.

2.5.6 Advantages and Limitations

The L298N motor driver stands out for its affordability, ease of use, and reliable operation. It supports the simultaneous control of multiple motors and is compatible with a wide variety of microcontrollers, making it an excellent choice for prototyping and development in embedded systems.

Despite its strengths, the L298N does come with a few drawbacks. It has a limited current handling capacity compared to more modern motor drivers, which can be a constraint in high- power applications. Moreover, when used at higher currents, it tends to generate significant heat, necessitating the use of heat sinks or other cooling methods to maintain safe operating temperatures.

2.6 Actuators

In a machine or a remote-control system, whether semi-automatic or automatic, an actuator is the component of the operative part that, upon receiving a command from the control unit via a possible pre-actuator, converts the supplied energy into useful work for performing tasks, which may be pre-programmed, in an automated system.

In other words, an actuator is the device that provides the necessary force to execute a task commanded by a remote-control unit. Actuators enable the transformation of received energy into a physical phenomenon (movement, heat generation, light emission, etc.). [33]

2.6.1 Gearmotor 12V 120RPM

A gear motor is a motor designed with an integrated gearbox. Gear motors function as torque multipliers and speed reducers thus requiring less power to

move a given load. The design of the gearbox structure, type of gears, lubrication and type of coupling affects its performance.

The concept of the gear motor is inspired by the patent of the German designer and entrepreneur *Albert Obermoser*, who in 1928 invented the so-called *Vorlegemotor* (gear motor). Since then, the basic model has been progressively improved and new models of gear motors have been created.



Figure 2.18. DC Gear-motors [34]

Depending on the number and type of gears, different combinations of output RPM and torque can be achieved. With fewer gears, the result is higher RPM and lower torque and vice versa. It can be mounted in any position.

The gear motor structure regulates whether the gear motor is suitable for light, medium or heavy loads and short or long operating periods. Depending on the internal gear structure and the reduction stages, the gear motor varies the speed on the output shaft.

The reduction ratio is the ratio between the input speed and the output speed; therefore, it is one of the most important characteristic values of the gearbox. The power and load capacity of a gearbox depends on the maximum torque it is capable of transmitting and it is measured in the physical unit Newton meter [Nm]. [34]

1) Technical Specifications

- **Voltage Rating:** 12V DC.
- **Speed:** 120 RPM (Revolutions Per Minute).
- **Horsepower:** 0.08 Hp
- **Torque:** 4.75 Nm
- **Shaft Type:** D-shaped shaft.
- **Dimensions:** Diameter: 37 mm,
Length: 21 mm (excluding the shaft), Shaft
diameter: 6 mm

2) Operational Principles

A DC gear motor is a combination of a DC motor and a gearbox. It works on the principle of electromagnetic induction, where a current-carrying conductor placed in magnetic fields, experiences a force and causes the rotation of the motor. The gearbox is used to increase or decrease the speed of the motor depending on application requirements.

The DC motor consists of two main components: stator and rotor. The stator consists of permanent magnets or electromagnets that create a stationary magnetic field. At the same time, the rotor consists of an armature coil that carries electricity. The gearbox is connected to the shaft of the DC motor. It consists of a series of gears that transmit the rotational motion from the motor to its output shaft.

When the power is supplied to the DC motor, the armature coil energizes and creates a magnetic field. The stationary magnetic of the stator interacts with the magnetic field of the armature coil and causes the rotor to rotate. Then the rotational motion of the motor is transmitted to the gearbox.

The gears in the gearbox either increase or decrease the speed and torque of the DC motor. Then the output shaft of the gearbox rotates at the desired speed and the torque as per the application's requirement. [35]

3) Applications

Electric Gear motors are used in applications that require high output torque and lower output shaft rotational speed, especially where space and available power

are limited. This describes a wide range of common equipment applications across multiple industries. From automotive wiper and window motors to hospital beds and patient lifts, robots to conveyor systems, automatic door operators to frozen beverage mixers, gear motors are critical to the overall performance of equipment that we depend on every day. [36]

2.6.2 Servomotor MG5010

A servo motor (servomotor) is a highly specialized motor designed for precise control of rotary or linear motion. It's a rotational or translational motor that employs a feedback mechanism to ensure exact positioning, typically using a control signal that dictates the motor's movement to a desired position. This mechanism allows for precise control of various components, making servo motors crucial in applications where precise positioning and smooth motion are required. [37]

In the 1960s, Yaskawa developed a succession of innovative DC servo motors, and in 1983, AC servo motor was launched, creating a new wave in the motor industry. And now, nearly 30 years later, almost 100% of factory automation (FA) uses AC servo motor. While a general motor is designed to work by continuously rotating the load, a servo motor is designed not only to rotate the load but also to respond accurately and quickly to the target. [38]



Figure 2.19. Servomotor [39]

1) Technical Specifications

The main technical characteristics of servomotors are as follows:

- **Operating voltage.**
- **The torque** indicates the rate of acceleration of the output shaft and its ability to overcome the resistance to the rotation of the load. The ability to realize the full potential of the servomotor is directly proportional to the torque.
- **The rotation speed** of the servomotor indicates the time it takes for the output shaft to turn by 60°. For example, a rotation speed of 0.07 s means that the servomotor shaft will turn by 60° in 0.07 s. The working voltage of the servomotor power supply affects both the rotation speed and the torque.
- **The maximum rotation angle** indicates the angle to which the output shaft of the servomotor can turn. In modern production, servomotors with continuous rotation are used, meaning that the maximum rotation angle is 360°. However, in some mechanisms, motors with smaller rotation angles, such as 120°, 180°, 270°, etc., are used.
- **The dimensions of the servomotor** affect the choice of the motor used to produce the mechanical structures in which they will be installed. This parameter is important for devices where speed, lightness, and compactness are crucial, such as drone models.

And for the one that we used in our project (MG5010 Servo Motor), Here is a table summarizing the main technical characteristics:

Table 2.6.2.1. *MG5010 technical characteristics*

Specification	Value
Operating Voltage	4.8V to 6.0V
Stall Torque	5.2 kg*cm (4.8V), 6.5 kg*cm (6.0V)
Rotation Speed	0.20 sec/60° (4.8V), 0.16 sec/60° (6.0V)
Dimensions	41 x 20 x 38 mm
Maximum Rotation Angle	180°

2) Operational Principles

A servomotor consists of a DTC motor, gearbox with shaft, and controller with necessary sensors (encoder, position sensor, etc.). A draft of the construction of a servomotor is presented in Figure1 [40]

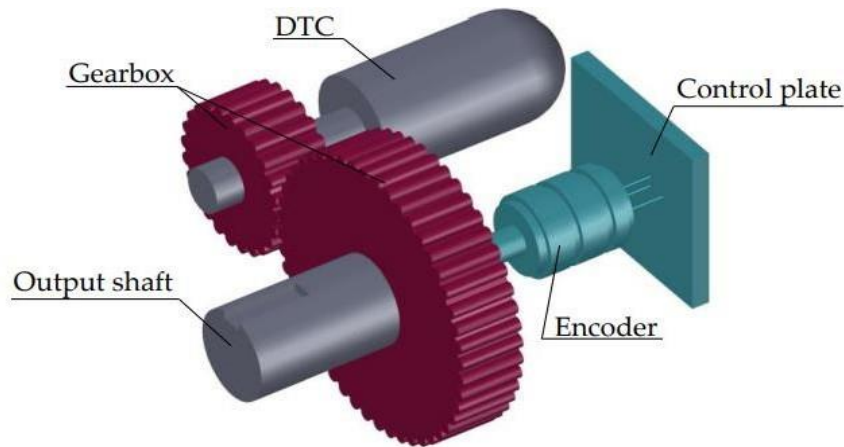


Figure 2.20. *The draft of the servomotor [40]*

If you open up a standard hobby servo motor, you will almost always find three core components: a DC motor, a controller circuit, and a potentiometer or similar feedback mechanism. The DC motor is attached to a gearbox and output/drive shaft to increase the speed and torque of the motor. The DC motor drives the output shaft. The controller circuit interprets signals sent by the controller, and the potentiometer acts as the feedback for the controller circuit to monitor the position of the output shaft. [39] thus creating a closed-loop control system with feed.

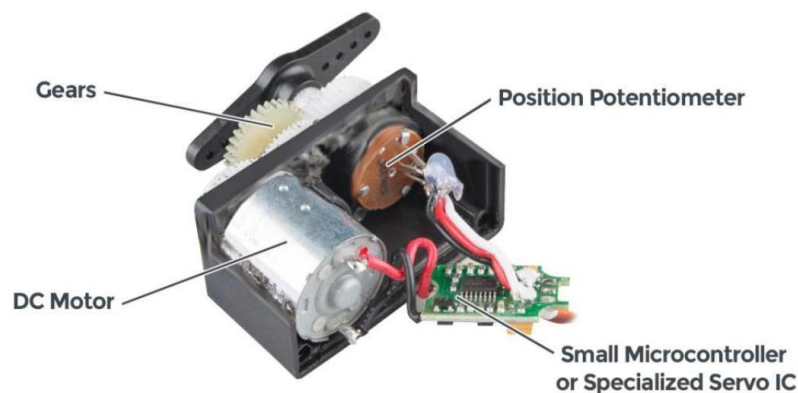


Figure 2.21. *The inside of the servomotor [39]*

3) **Controlling servomotors: The pulse width modulation technique**

A popular method of controlling a servomotor is pulse-width modulation (PWM). This method is based on determining the angle of rotation or speed of the output shaft based on the pulse length at a given frequency.

1. PWM generates pulses of varying width (duration) with different periods.
2. To control the speed of the servomotor, the width of the pulses is changed, allowing the regulation of the power supplied to the motor.
3. Control of the position of the servomotor is possible using feedback. To adjust the position, the voltage applied to the motor, after the PWM signal is converted, is compared to the desired voltage, resulting in a control signal.
4. PWM is also used to regulate a smooth trajectory of movement from one point to another for the servomotor.
5. In addition to PWM, PID controllers and microcontrollers are applied to enhance the efficiency of regulation and control.

This method has gained popularity due to its simplicity of implementation and low cost. PWM also provides high efficiency in speed and positioning control, making it applicable in applications requiring a high response to control input and precise control. Examples of such applications may include non-dynamic systems such as fans, pumps, or conveyors. [40]

4) **Applications**

Servomotors are used in applications that require precise control of position, speed, and torque. They have a wide range of applications in various fields and industries. Some of the common applications are:

- **Robotics:** Servo motors are used to provide precise motion and force for robotic arms, legs, joints, grippers, etc. They enable robots to perform tasks such as picking, placing, welding, assembling, etc.
- **CNC machinery:** Servo motors are used to drive the axes of CNC machines such as lathes, mills, routers, etc. They enable CNC machines to perform accurate and complex machining operations such as cutting, drilling, engraving, etc.

- Automated manufacturing: Servo motors are used to control the movement and position of various components and devices in automated manufacturing systems, such as conveyors, feeders, loaders, unloaders, etc. They enable automated manufacturing systems to achieve high productivity and quality.
- Medical equipment: Servo motors are used to operate various medical devices and instruments such as surgical robots, scanners, pumps, ventilators, etc. They enable medical equipment to perform precise and safe operations and treatments. [41]

2.7 Sensors

2.7.1 Color Sensor TCS34725

TCS34725 is an I2C bus-based color light-to-digital converter with an IR filter, providing a digital return of red, green, blue (RGB), and clear light sensing values. The high sensitivity, wide dynamic range, and IR blocking filter make the TCS34725 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials. [17]



Figure 2.22. TCS34725 Color Sensor [43]

Features:

- Separate Red/Green/Blue light intensity sensors
- Ambient (clear) light intensity sensor

- IR light blocking filter
- Integrated logic controlled white LED light source
- 3,800,000 :1 dynamic range
- Adjustable gain and integration time
- Interrupt output available based on predefined lighting thresholds
- I2C interface
- 3.3 or 5V operation [44]

1) Module Connections

The module brings out the following connections.

1 x 7 Header

- **LED** = LED control input. Leave floating for LED always on. Pull LOW to turn LED off
- **INT** = Interrupt output active LOW. Output is open-collector so requires pull-up resistor. Connects to an interrupt input on the MCU if used.
- **SDA** = I2C SDA line. Connects to SDA on MCU
- **SCL** = I2C SCL line. Connects to SCL on MCU
- **3V3** = 3.3V output from the on-board 3.3V regulator
- **GND** = Ground connects to ground on the MCU
- **VIN** = Power input. Connects to 3.3V or 5V on the MCU [44]

Figure.01: RGB Color Sensor TCS34725 Pinout Diagram

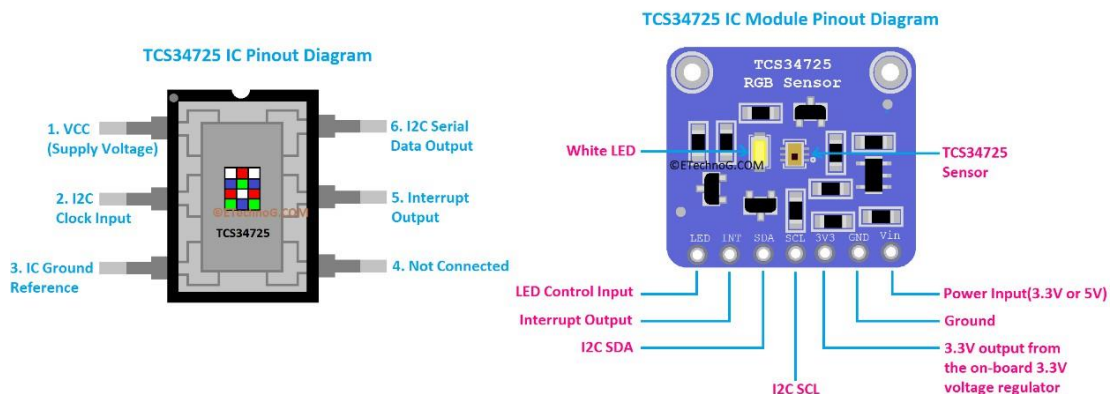


Figure 2.23. TCS34725 Pinout Diagram [45]

2) Working Principles

The TCS3472 device provides a digital return of red, green, blue (RGB), and clear light sensing values. An IR blocking filter, integrated on-chip and localized to the color sensing photodiodes, minimizes the IR spectral component of the incoming light and allows color measurements to be made accurately. The high sensitivity, wide dynamic range, and IR blocking filter make the TCS3472 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials. [44]

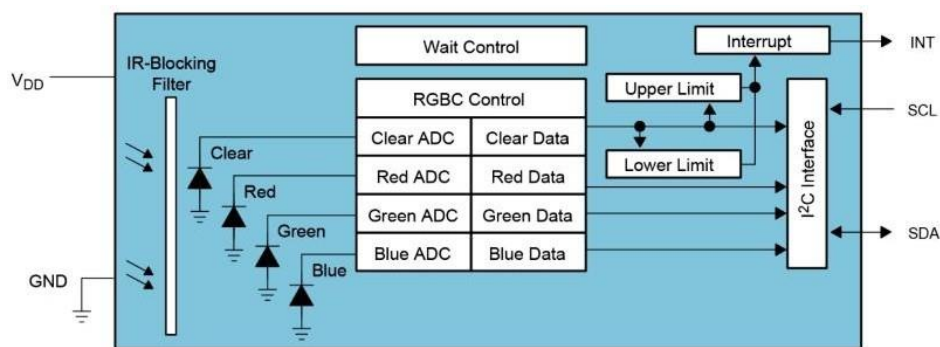


Figure 2.24. Functional Block Diagram of TCS34725 [44]

The measurements are taken using a 3×4 matrix of sensors that have red/green/blue color filters in front of them. It also measures the overall intensity of the reflected light using a clear filter over three of the sensors. The entire sensor array has an IR filter over it that minimizes the effect of IR light on the readings.

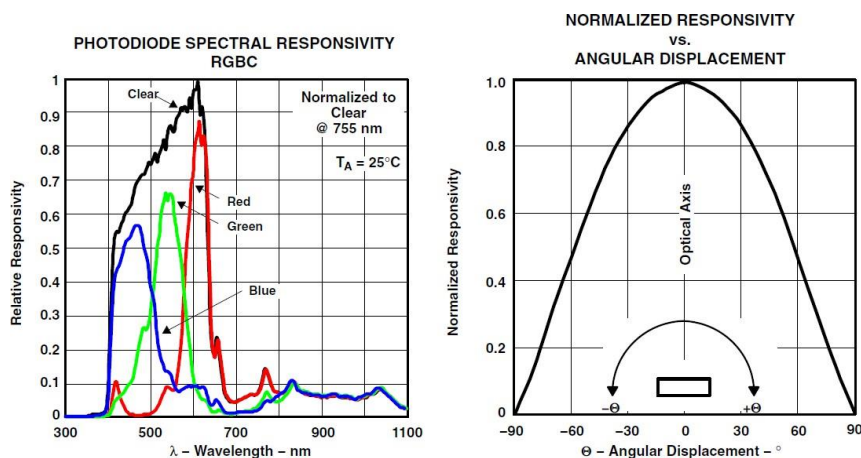


Figure 2.25. Color recognition sensor (CMOS technology) [46]

The 3×4 matrix is formed by photodiodes filtered for red, green, blue, and unfiltered (clear). All sensors are filtered for infrared.

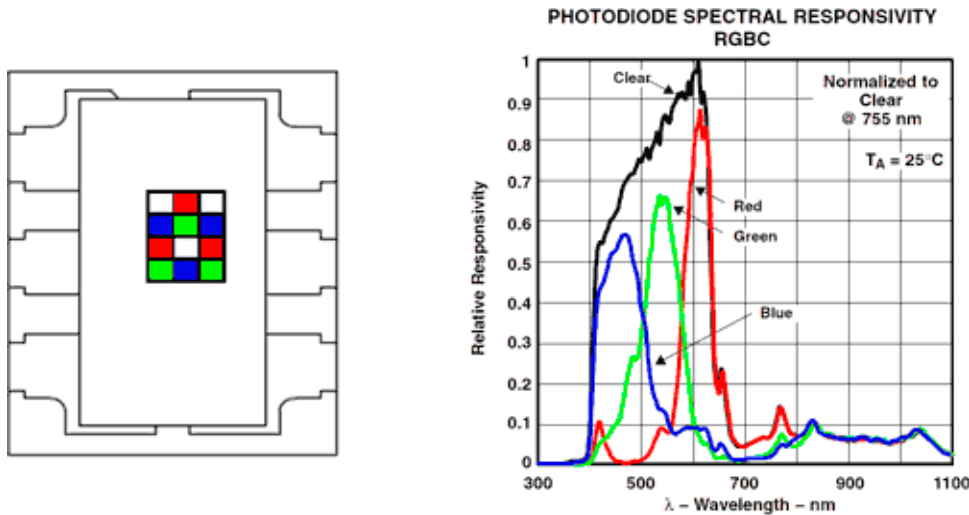


Figure 2.26. *TCS34725 Photodiode Spectral Responsivity RGBC [47]*

The analog sensor outputs are converted to 16-bit digital values using 4 integrating ADCs. These sensor digital values are then available to the MCU via the I2C bus. [44]

The module communicates via a standard I2C interface. The I2C address is fixed at the address of 0x29.

The module includes MOSFETs and pull-up resistors to provide level shifting of the I2C SDA and SCL lines so that they will work correctly with both 3.3 and 5V MCUs. [48]

There are two main settings that can be configured to affect the readings and optimize them for a particular application:

Gain Setting: The gain setting can be set to 1X, 4X, 16X, or 60X. Higher gain settings may help read the color correctly under lower light conditions but may also increase the noise level of the reading.

Integration Time: The integration time is the length of time used for taking the sample. Longer integration times may provide better accuracy in some applications. The integration time can be set to 2.4ms, 24ms, 50ms, 101ms, 154ms, or 700ms. [44]

3) Applications

The **TCS34725 Color Sensor RGB Sensor Board** is widely used in a variety of applications that require accurate color sensing. In industrial automation, the sensor can be used for product quality control, where it ensures that items meet specific color standards. It is also commonly used in robotics, where it can help robots recognize and respond to different colors, enabling tasks such as line-following or color-based sorting.

In consumer electronics, the **TCS34725** is used in devices that require ambient light sensing and color calibration, such as smartphones and monitors. The sensor's ability to provide accurate color data makes it an essential component in any application where color differentiation is important. [49]

2.8 LCD I2C Module

The I2C display interface is a common communication protocol for character, graphic, and segment LCDs. An I2C display consists of an LCD controller and a display module. The interface is communicated over two wires which include a data line and a clock line.[50], making it easier to connect to microcontrollers like the Arduino Uno. This section explores its features, working principles, and applications in the sorting system.

2.8.1 Overview

Character LCDs are specially designed to display letters, numbers, and symbols. A 16×2 character LCD, for example, can show 16 characters across each line, with two lines total.

If you look very closely at the screen, you'll notice small rectangular boxes for each character position. Inside each of these rectangles is a grid of 5×8 tiny dots or pixels. These pixels light up in different patterns to form different letters, numbers, or symbols. [51]

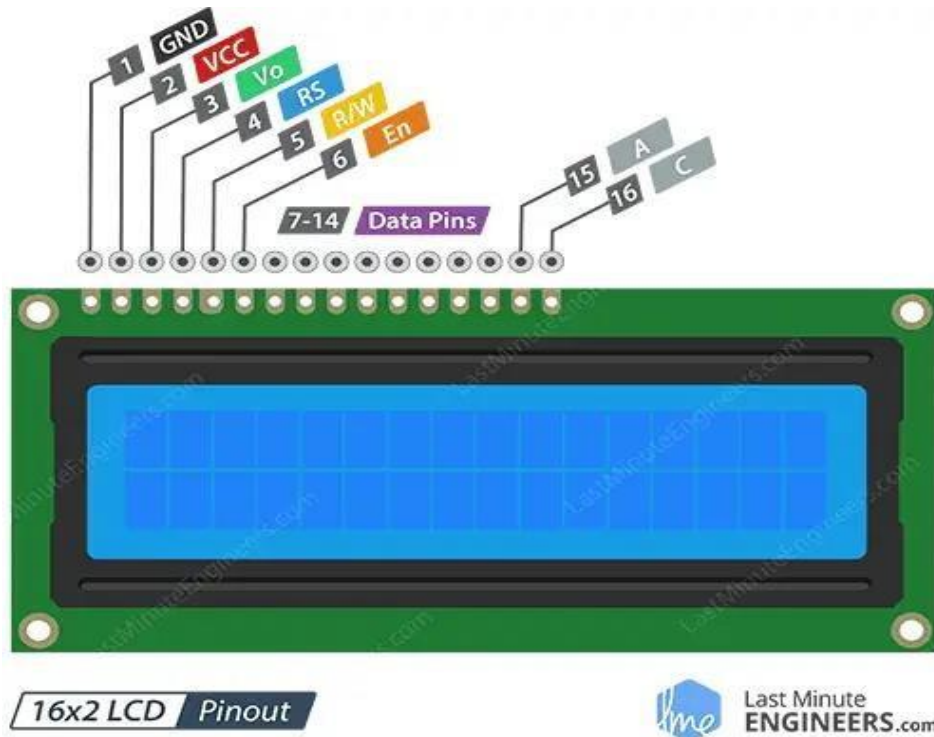


Figure 2.27. 16x2-Character-LCD-5x8-Pixel-Matrix-Grid [51]

2.8.2 Technical Specifications

- **Display Type:** 16x2 character LCD.
- **Interface:** I2C (Two-wire communication).
- **Operating Voltage:** 5V DC.
- **Backlight:** green with white text.
- **I2C Address:** Configurable (default is often 0x27 or 0x3F).
- **Dimensions:**
 - Width: 80 mm (including the PCB).
 - Height: 31 mm (including the PCB).
 - Thickness: 7 mm (Just the LCD panel portion)

2.8.3 Working Principles

A Liquid Crystal Display (LCD) works by carefully controlling light using liquid crystals to create images or text on your screen.

Inside every LCD, there's a backlight that provides a steady source of light. The special liquid crystals are sandwiched between two layers of polarized glass.

[51]

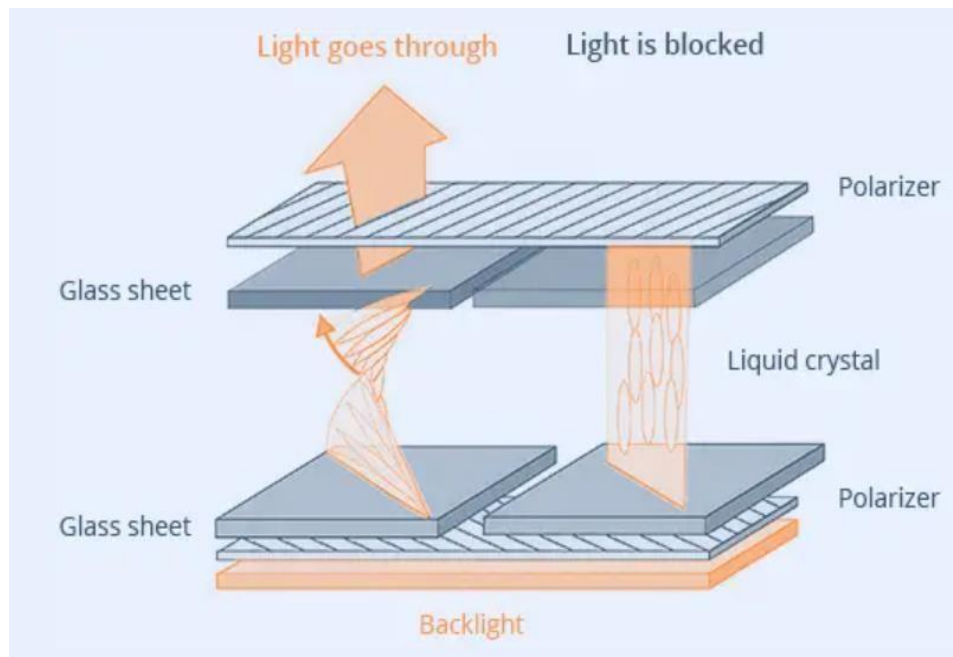


Figure 2.28. *Liquid Crystal technology* [51]

The LCD I2C module uses the I2C communication protocol to receive data from the microcontroller. The I2C backpack converts the I2C signals into parallel data for the LCD. Key features include:

- **I2C Communication:** The module communicates with the microcontroller using the SDA (data line) and SCL (clock line) pins.
- **Backlight Control:** The backlight can be turned on/off or adjusted using the I2C interface.
- **Custom Characters:** The LCD can display custom characters by programming the character generator RAM (CGRAM). [51]

2.8.4 Pin Configuration

The LCD I2C module typically has the following pins:

VCC: Connects to the 5V power supply.

GND: Connects to the ground.

SDA: Connects to the SDA pin on the microcontroller (e.g., A4 on Arduino Uno).

SCL: Connects to the SCL pin on the microcontroller (e.g., A5 on Arduino Uno).

2.8.5 Applications in the Sorting System

In the sorting system, the LCD I2C module can be used for:

Status Display: Showing the current state of the system (e.g., "Sorting in Progress," "Object Detected").

Debugging Information: Displaying sensor readings, motor status, or error messages.

User Interface: Providing a simple interface for users to interact with the system (e.g., start/stop buttons).

2.8.6 I2C Serial Interface Adapter Module for LCD

Due to limited pin resources in a microcontroller/microprocessor, controlling an LCD panel could be tedious. Serial to Parallel adapters such as the I2C serial interface adapter module with PCF8574 chip makes the work easy with just two pins. The serial interface adapter can be connected to a 16x2 LCD and provides two signal output pins (SDA and SCL) which can be used to communicate with an MCU/MPU. [52]

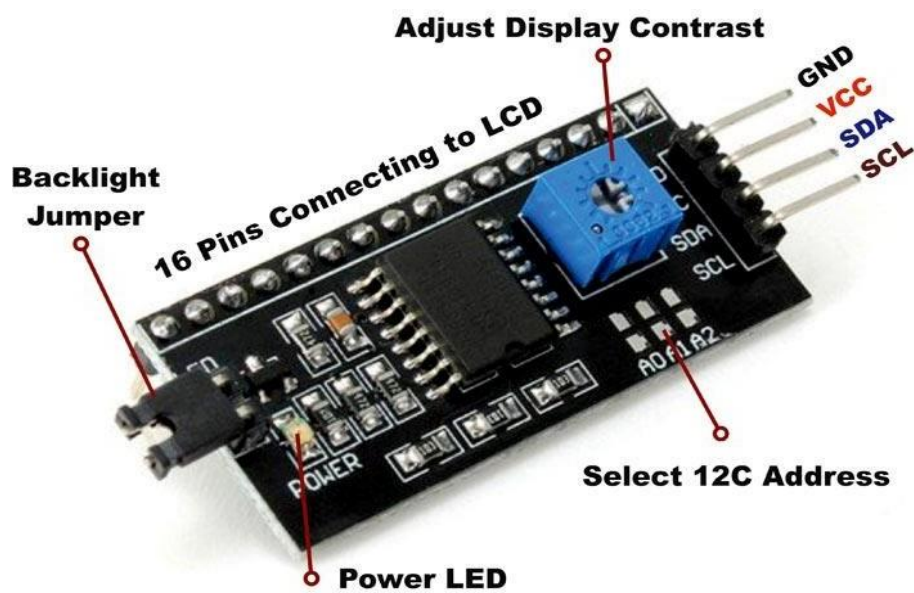


Figure 2.29. PCF8574 I2C LCD adapter [52]

Features and Specifications:

- Operating Voltage: 5V DC
- I2C control using PCF8574
- Can have 8 modules on a single I2C bus

- I2C Address: 0X20~0X27 (the original address is 0X20, you can change it yourself via the onboard jumper pins) [52]

Pin Configuration:

Table 2.8.6. *PCF8574 Pin Configuration*

Pin Name	Pin Type	Pin Description
GND	Power	Ground
VCC	Power	Voltage Input
SDA	I2C Data	Serial Data
SCL	I2C Clock	Serial Clock
A0	Jumper	I2C Address Selection 1
A1	Jumper	I2C Address Selection 2
A2	Jumper	I2C Address Selection 3
Backlight	Jumper	Control Backlight of panel

Chapter 3

Control Logic and Circuit Simulation

3.1 Introduction

In this chapter, we present the software tools and simulation environments used to develop, test, and validate the proposed sorting system. It includes the configuration and use of the Arduino IDE for programming and Tinkercad Circuits for virtual prototyping and simulation. Additionally, detailed wiring diagrams created using Fritzing and simulation screenshots are included to illustrate the project's software and virtual implementation.

3.2 Software Tools Overview

3.2.1 Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.[53]

Key Features:

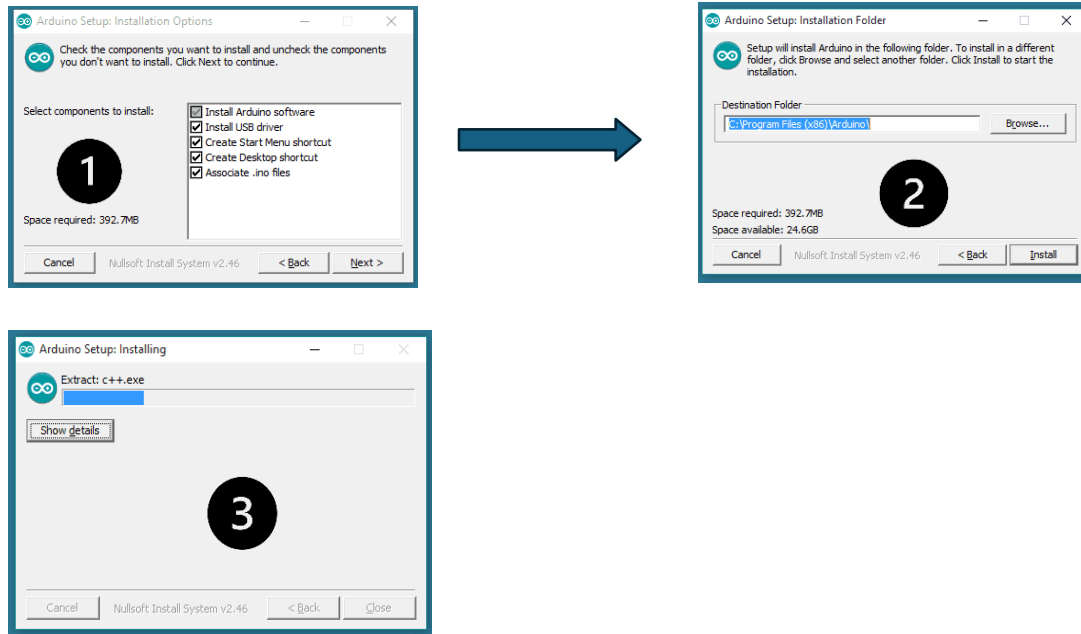
Simple C/C++ based language.

Integration with required libraries (Adafruit_TCS34725, Wire, Servo, LiquidCrystal_I2C). Real-time serial monitoring for debugging sensor data and system behavior.

1) Arduino IDE Installation (Windows)

Get the latest version from the download page. You can choose between the Installer (exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package you need to install the drivers manually. The Zip file is also useful if you want to create a portable installation.

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.[53]



The process will extract and install all the required files to execute properly the Arduino Software (IDE)

2) General program structure

Similar to other programming languages, Arduino features an adaptable and intuitive development environment that is compatible with any operating system capable of executing C/C++ code.



Figure 3.1. Arduino IDE interface [54]

Verify / Upload - compile and upload your code to your Arduino Board.

Select Board & Port - detected Arduino boards automatically show up here, along with the port number.

Sketchbook - here you will find all of your sketches locally stored on your computer. Additionally, you can sync with the Arduino Cloud, and also obtain your sketches from the online environment.

Boards Manager - browse through Arduino & third-party packages that can be installed. For example, using a MKR WIFI 1010 board requires the Arduino SAMD Boards package installed.

Library Manager - browse through thousands of Arduino libraries, made by Arduino & its community.

Debugger - test and debug programs in real time.

Search - search for keywords in your code.

Open Serial Monitor - opens the Serial Monitor tool, as a new tab in the console.

3) Program Implementation

Before uploading code to the board, ensure the correct board type (e.g., Arduino UNO) and USB port (e.g., COM3) are selected, as illustrated in the figure below.

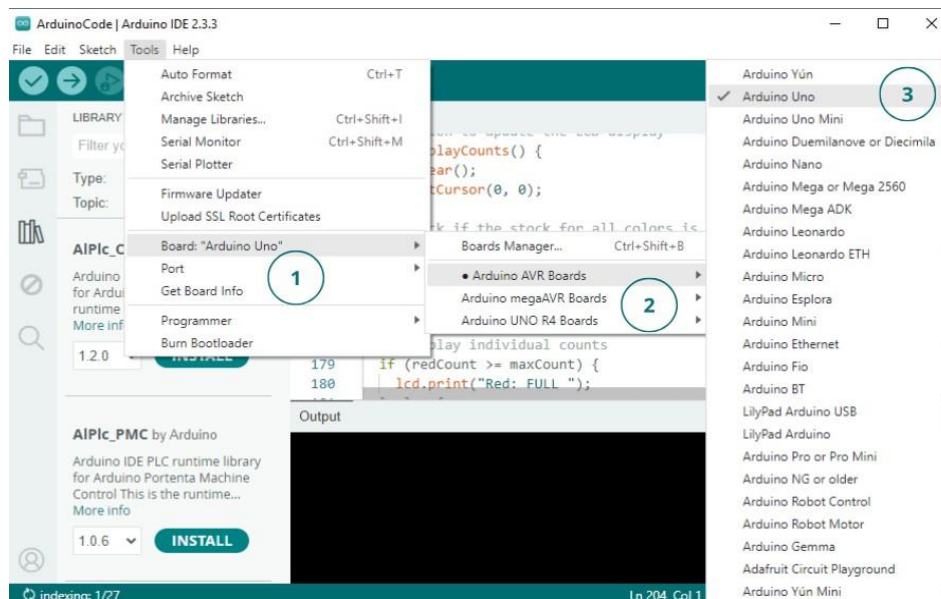


Figure 3.2. selecting Arduino Board (STEP1)

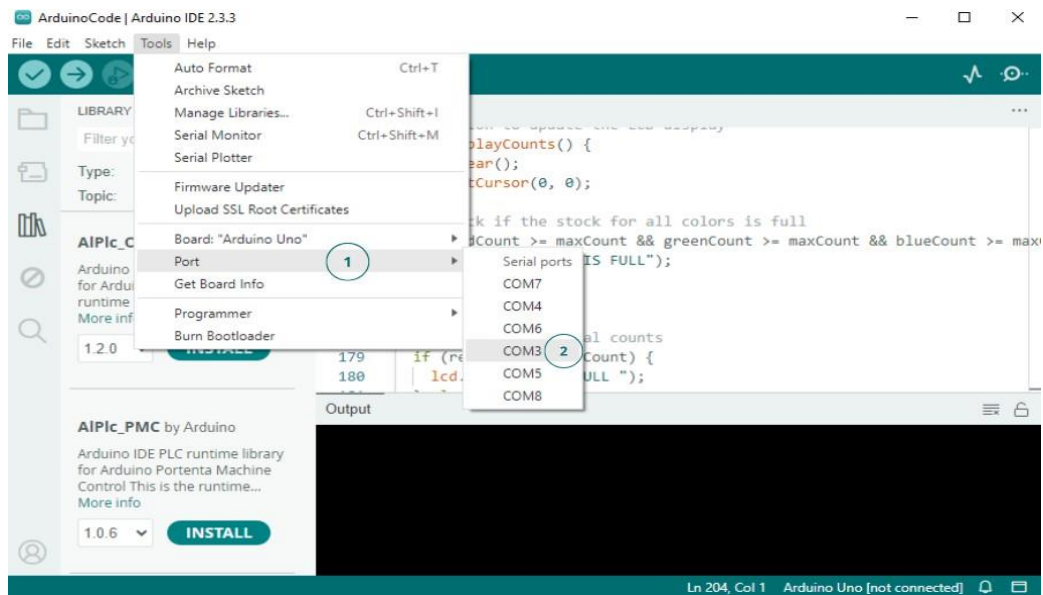


Figure 3.3. selecting Port (STEP2)

4) Program Description

The Arduino API can be divided into three main parts: functions, variables and structure:

- **Functions:** for controlling the Arduino board and performing computations. For example, to read or write a state to a digital pin, map a value or use serial communication.
- **Variables:** the Arduino constants, data types and conversions. E.g. int, boolean, array.
- **Structure:** the elements of the Arduino (C++) code, such as sketch (loop (), setup ()), control structure (if, else, while, for), arithmetic operators (multiplication, addition, subtraction), comparison operators, such as == (equal to), != (not equal to), > (greater than).

The Arduino API can be described as a simplification of the C++ programming language, with a lot of additions for controlling the Arduino hardware.[55]

Program Structure

The absolute minimum requirement of an Arduino program is the use of two functions:

void setup() and **void loop()**

. The "void" indicates that nothing is returned on execution.

void setup()

- this function executes only once, when the Arduino is powered on. Here we define things such as the mode of a pin (input or output), the baud rate of serial communication or the initialization of a library.

void loop()

- this is where we write the code that we want to execute over and over again, such as turning on/off a lamp based on an input, or to conduct a sensor reading every X second.

The above functions are **always** required in an Arduino sketch, but you are of course able to add several more functions, which is useful for longer programs.[55]

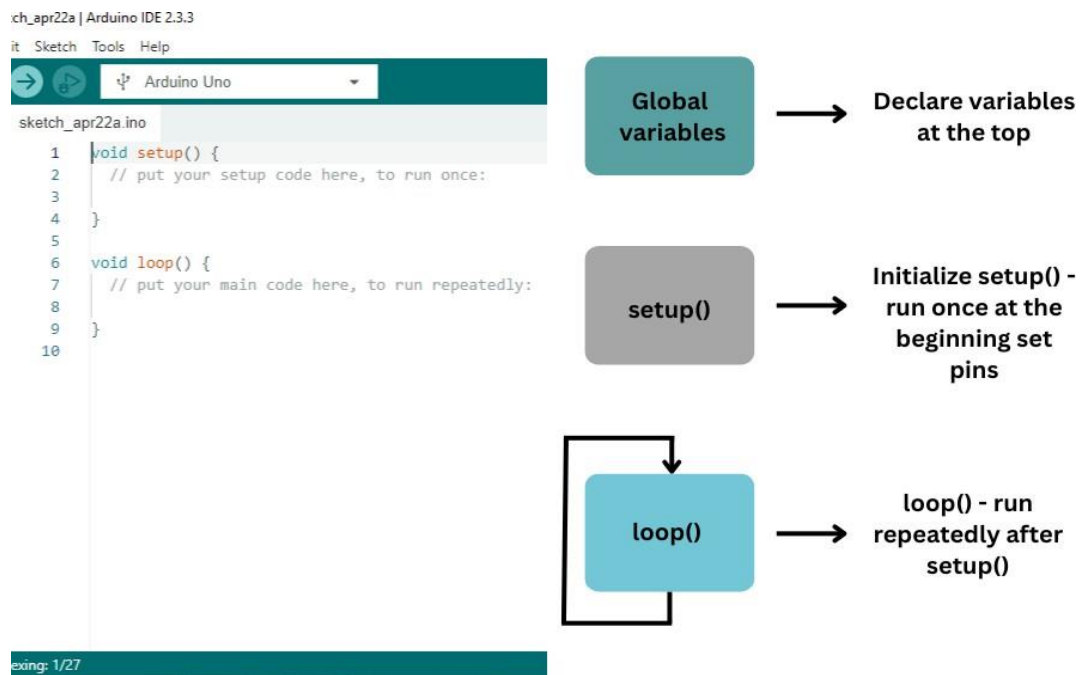


Figure 3.4. Program Structure

Variable declaration

A variable is a place to store a piece of data. It has a name, a value, and a type. For example, this statement (called a *declaration*):

```
int pin = 13;
```

creates a variable whose name is pin, whose value is 13, and whose type is int. Later on in the program, you can refer to this variable by its name, at which point its value

will be looked up and used. For example, in this statement:

```
pinMode(pin, OUTPUT);
```

it is the value of pin (13) that will be passed to the pinMode() function. In this case, you don't actually need to use a variable, this statement would work just as well:

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

The advantage of a variable in this case is that you only need to specify the actual number of the pin once, but you can use it lots of times. So if you later decide to change from pin 13 to pin 12, you only need to change one spot in the code. Also, you can use a descriptive name to make the significance of the variable clear (e.g. a program controlling an RGB LED might have variables called redPin, greenPin, and bluePin).[62]

Here is a simple code example demonstrating variable declaration and modification:

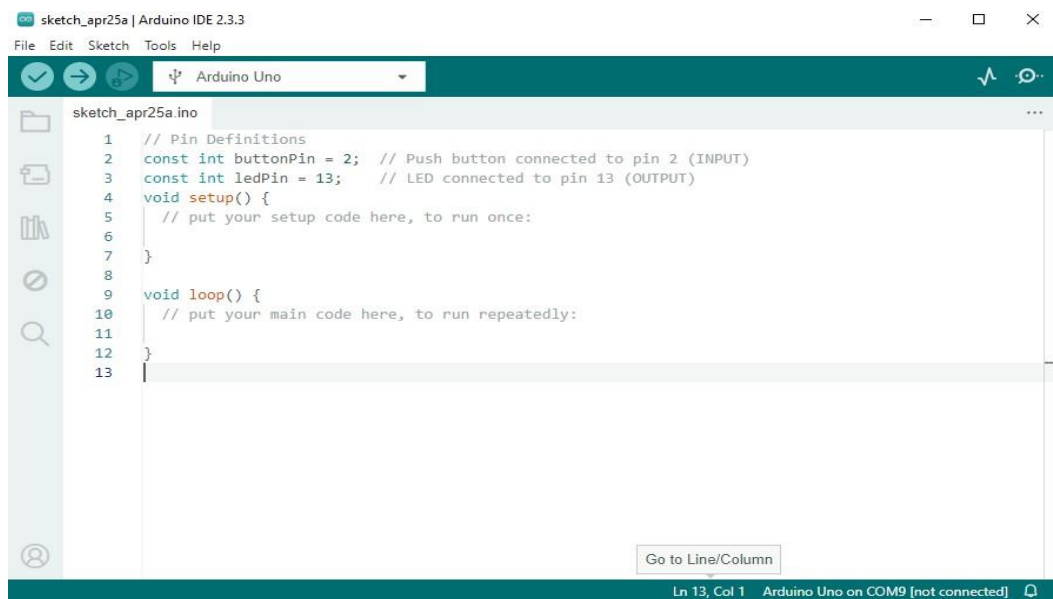


Figure 3.5. declaring variables, button (pin 2), LED (pin 13)

Code demonstration:

const int ensures these values **cannot be**

changed later. Use descriptive names

(buttonPin, ledPin) for readability.

Pins are declared globally (accessible in setup() and loop()).

The "setup()" function

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup() function will only run once, after each powerup or reset of the Arduino board.[56]

As the void setup function is called only once at the very beginning of the program, this will be the place to:

Initialize variables' values.

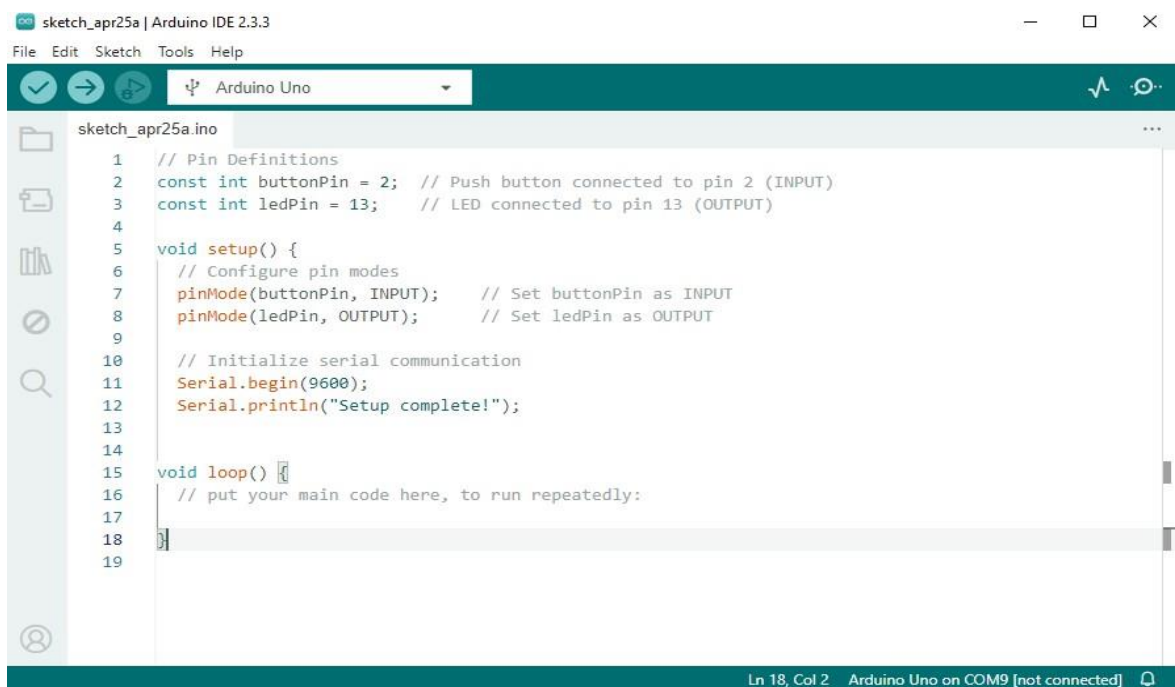
Setup communications (ex: Serial).

Setup modes for digital pins (input/output).

Initialize any hardware component (sensor/actuator) plugged to the Arduino.Etc....

The void setup, as its name suggest, is made for you to do any setup required at the beginning of the program. Don't write the core functionalities here, just the initialization code.[57]

Example: Turning an LED off and on using a push button



```
sketch_apr25a | Arduino IDE 2.3.3
File Edit Sketch Tools Help
Arduino Uno
sketch_apr25a.ino
1 // Pin Definitions
2 const int buttonPin = 2; // Push button connected to pin 2 (INPUT)
3 const int ledPin = 13; // LED connected to pin 13 (OUTPUT)
4
5 void setup() {
6 // Configure pin modes
7 pinMode(buttonPin, INPUT); // Set buttonPin as INPUT
8 pinMode(ledPin, OUTPUT); // Set ledPin as OUTPUT
9
10 // Initialize serial communication
11 Serial.begin(9600);
12 Serial.println("Setup complete!");
13
14
15 void loop() {
16 // put your main code here, to run repeatedly:
17
18
19 }
```

Ln 18, Col 2 Arduino Uno on COM9 [not connected]

Figure 3.6. void setup()

Code explanation:

pinMode() configures pins as:

INPUT: For reading signals (e.g., buttons, sensors). OUTPUT: For controlling components (e.g., LEDs, motors).

Serial.begin(9600): Starts serial communication at **9600 baud rate** (needed for printing messages).

Serial.println("..."): Prints text to the **Serial Monitor** (Tools > Serial Monitor).

Note: This section runs **only once** when the Arduino powers up.

The "loop()" function

After creating a `setup()` function, which initializes and sets the initial values, the `loop()` function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.[58]

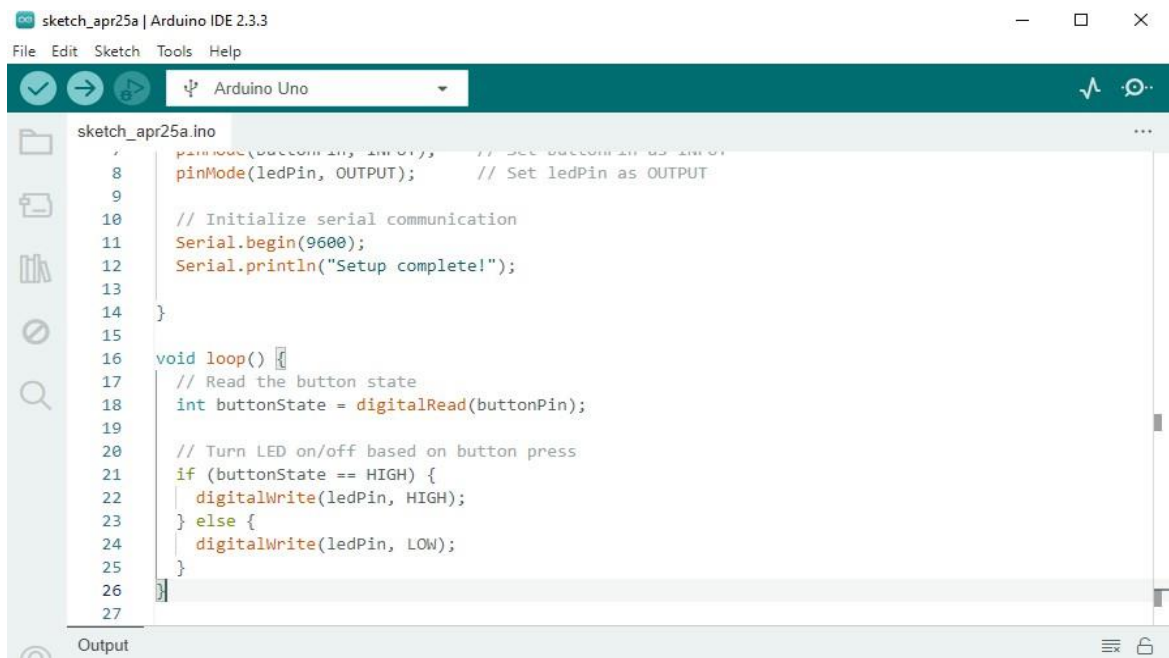


Figure 3.7: `void loop()`

Key Points:

- **digitalRead()** checks the state of `buttonPin` (HIGH/LOW).

- **digitalWrite()** controls the LED:
 - HIGH: 5V (ON).
 - LOW: 0V (OFF).
- The code loops **forever**, creating real-time interactions.

5) Installing required libraries

Arduino libraries are typically small packages of code that somebody else wrote for a particular purpose. One example is a library that adds support for a specific sensor, such as the BME280. Utilizing libraries means you don't have to know a sensor or module inside-out to connect it to an Arduino and try it. Furthermore, you don't have to reinvent the wheel if a library exists that already implements a commonly used feature (e.g., mathematical functions such as vector and matrix calculations).

The Arduino IDE comes with a few built-in libraries right away, which you can find a list of here. Besides these, you can typically find new libraries that add support for breakout modules on the vendor's or manufacturer's website.

As everyone is free to write their own

Arduino libraries, you can also download new libraries on many personal websites and blogs. When you manually download a library for the Arduino IDE, you typically end up with a single zip file that contains all the code files. There is, however, also an easier way of installing Arduino libraries with the library manager.[59]

Here's an example of how to install a library using the Arduino IDE's built-in library manager by using main menu bar:

Installation via Library Manager:

- Select **Tools** in menu bar → choose **Manage Libraries**
- Search for wanted Library → Select the Correct Library & Version
- Click "**Install**"
- Verify Installation: The library should now appear under **Sketch** → **Include Library**

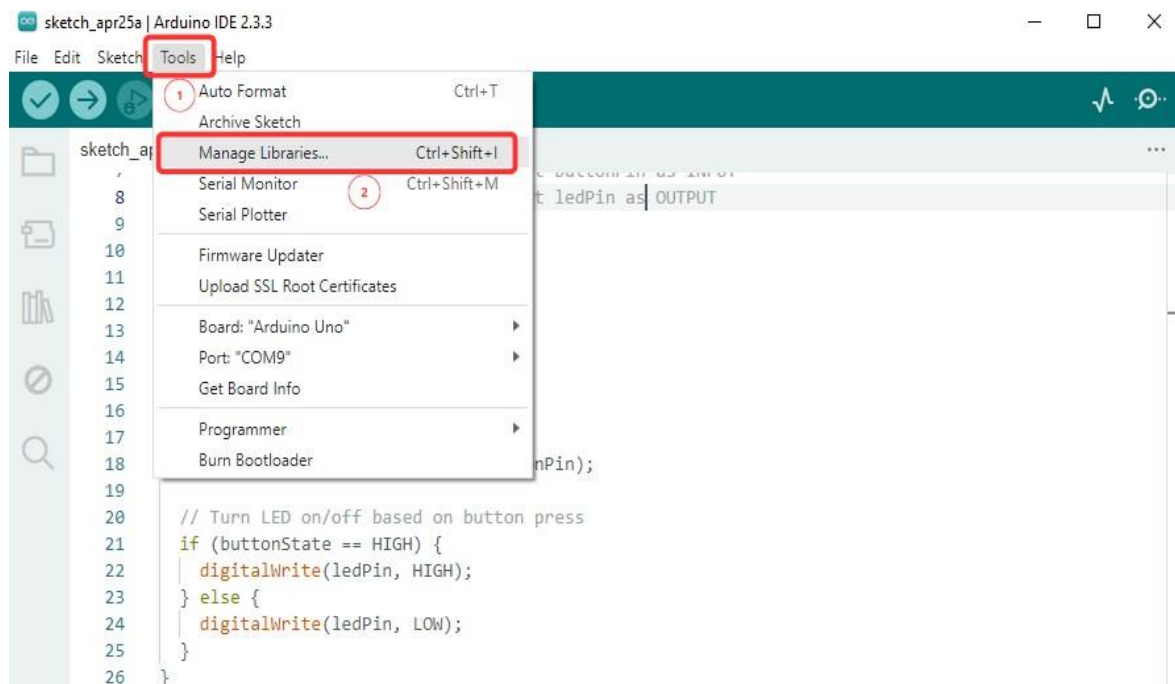


Figure 3.8: Selecting Library manager

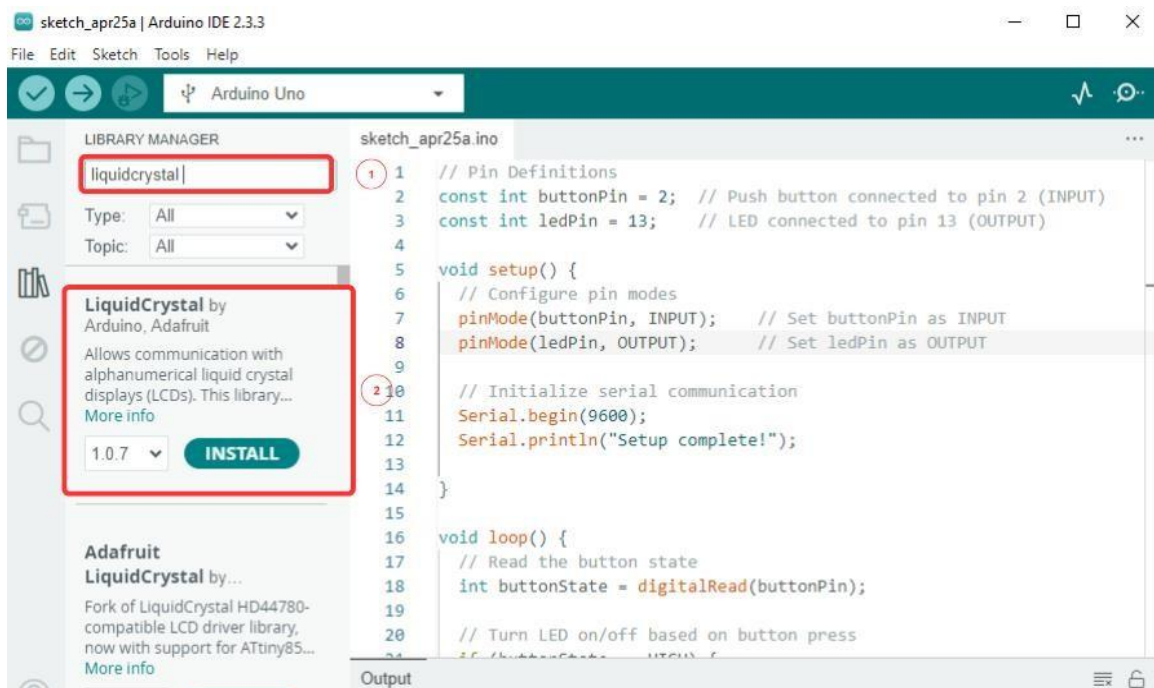


Figure 3.9. Installing wanted Library

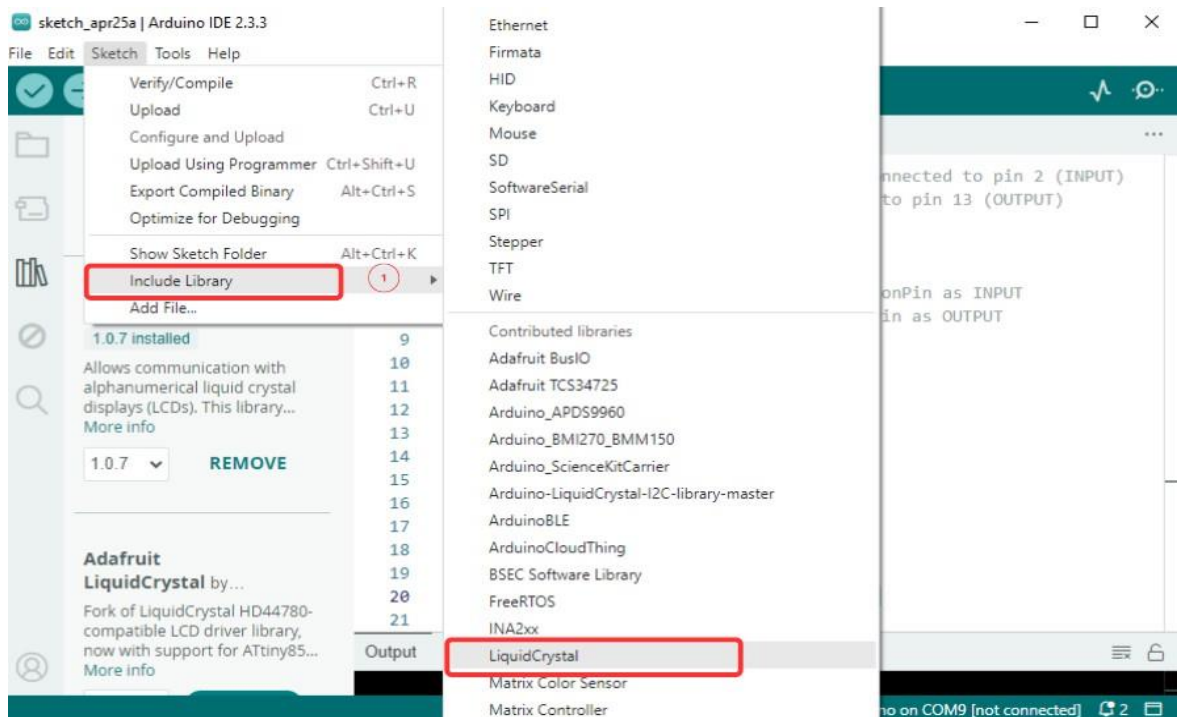


Figure 3.10. Verification of Library

6) Compilation and Upload Process

Once you have selected the correct Arduino board and port, you can proceed to upload the sketch:

Click the Verify Button (✓)

Check the Output Console:

✓ Success → "Done compiling."

✗ Failure → Error messages appear.

Fix Errors (if any) – Review error messages, correct the code, and verify again.

Click the "Upload" button to program the board with the sketch.

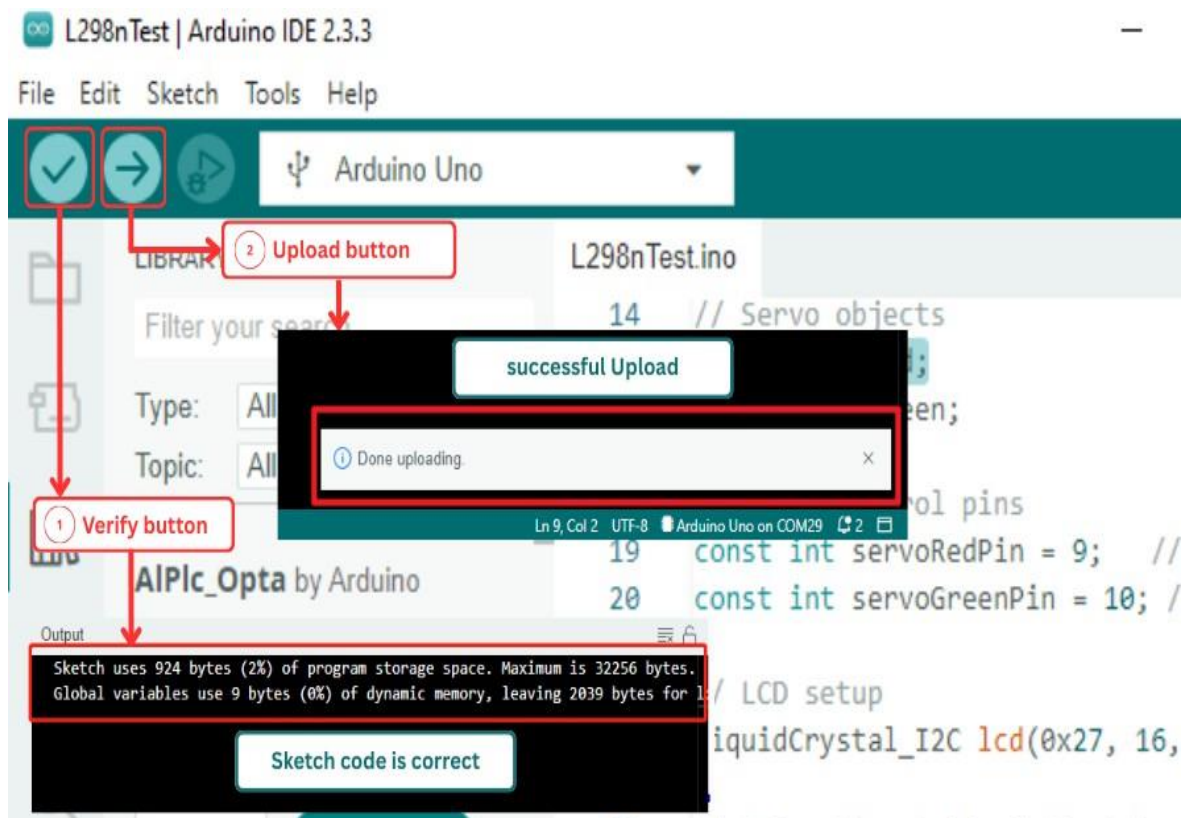


Figure 3.11. Verification & Uploading sketch

After uploading, the Arduino automatically runs the sketch when powered on, with the TX/RX LEDs blinking during serial communication, and the program can be replaced by uploading new code.

3.3 Tinkercad

3.3.1 Overview

Tinkercad is a free-of-charge, online 3D modeling program that runs in a web browser.^[1] Since it became available in 2011 it has become a popular platform for creating models for 3D printing as well as an entry-level introduction to constructive solid geometry in schools.[60] One of the most exciting things about Arduino and Tinkercad is that they can be used together. Tinkercad has a range of electronic components that can be used to build circuits and control devices, and it also has a built-in Arduino simulator that allows users to program and test their circuits in a virtual environment. This makes it easy to prototype and test new ideas before building them in the real world.[61]

3.3.2 Interface

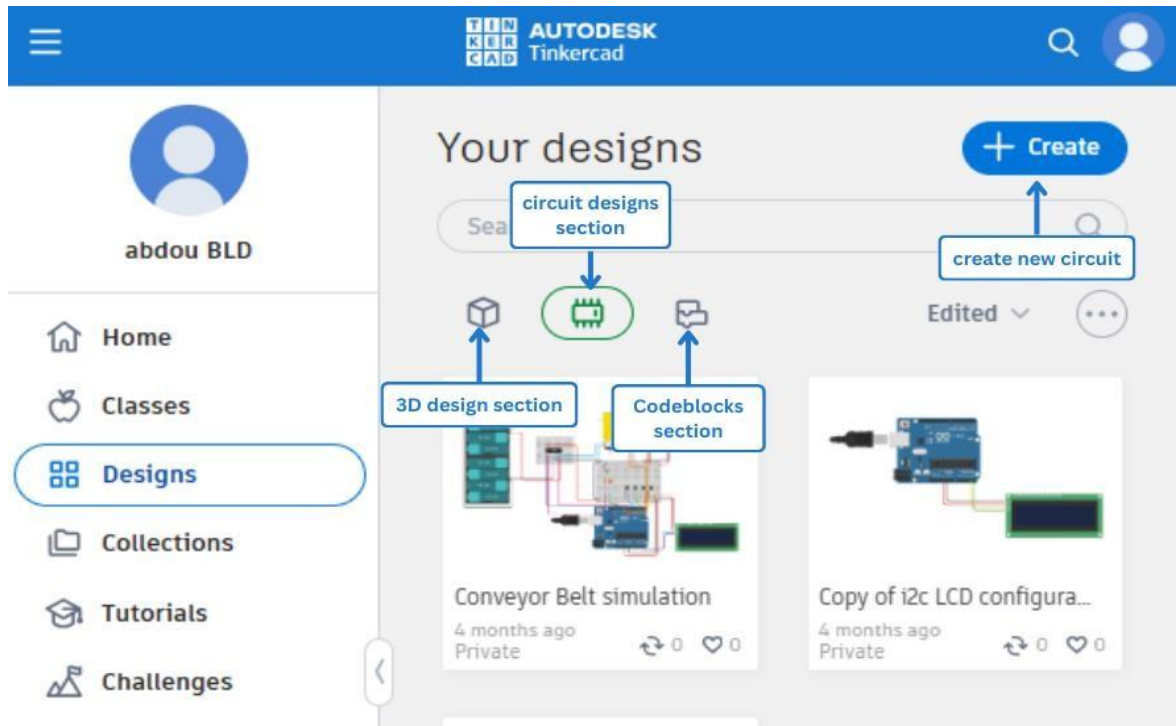


Figure 3.12. Tinkercad overall interface

Main Sections (Left Sidebar)

- **Home:** Takes you to the homepage.
- **Classes:** Manage or view class-related content.
- **Designs:** View all your projects (selected in the image).
- **Collections:** Organize your designs into groups.
- **Tutorials:** Access learning resources and guided lessons.
- **Challenges:** Participate in creative tasks and challenges.

Design Categories (Center Tabs)

- **3D Design Section:** Icon looks like a cube. For creating and viewing 3D models.
- **Circuit Designs Section:** Highlighted with a green chip icon. Used to build and simulate electronic circuits.

- **Codeblocks Section:** Icon looks like stacked blocks. Allows programming using visual blocks (ideal for logic-based learning).
- **Creating New Design**

Create Button: Located at the top-right. Click this to start a new project (circuit, 3D designs, etc.).

3.3.3 Circuit Design Workspace

In Tinkercad, the main focus is on circuit design, which is supported by a user-friendly interface consisting of a top toolbar with various essential functions, a right-hand sidebar containing all the electronic components, and a central workspace where the actual circuit is built, simulated, and observed in action.

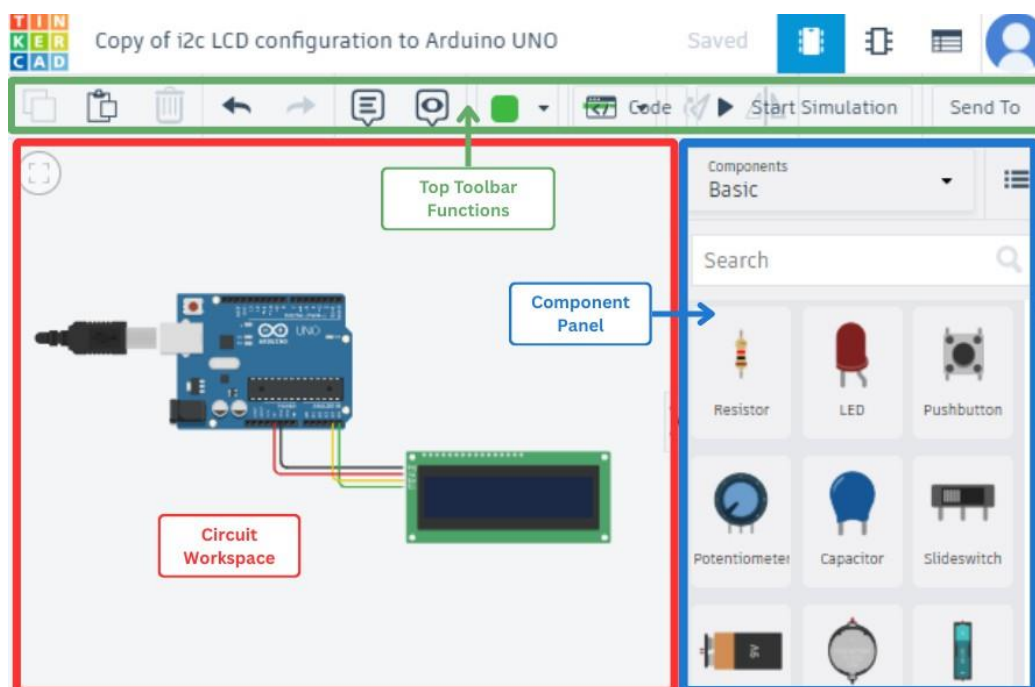


Figure 3.13: Tinkercad Circuit design interface

Top Toolbar Functions (Green area)

- **Delete (Trash icon):** Remove selected components.
- **Undo / Redo Arrows:** Revert or repeat recent actions.
- **Comment Tool (Speech bubble):** Add comments to specific areas.

- **Zoom Controls (Magnifying Glass icons):** Adjust view of the workspace.
- **Code Button:** Opens the code editor (Block-based, Text, or both).
- **Start Simulation:** Run the circuit simulation.
- **Send To:** Share or export the design.

Circuit Workspace (Main Area in Red)

- **Arduino Uno:** Main microcontroller in this circuit.
- **LCD Display (I2C):** Connected to the Arduino via four wires (GND, VCC, SDA, SCL).
- **Wiring:** Colored lines indicate electrical connections between components.

Component Panel (Right Sidebar in blue)

- **Components Category Dropdown:** Choose between Basic, All, or custom components.
- **Search Bar:** Find specific electronic parts quickly.
- **Component Icons:** (Resistor, LED, Pushbutton, Potentiometer.... etc)

Drag and Drop: Add components to the circuit by dragging from the panel into the workspace.

3.3.4 Simulation Steps in Tinkercad

Here are steps for conducting a simulation of an LCD i2c connected to an Arduino board:

Step 1: Design Your Circuit

Drag and drop components from the right-hand sidebar into the workspace. Connect components properly using virtual wires:

- LCD I2C GND --> GND Arduino
- LCD I2C VCC --> 5V Arduino
- LCD I2C SDA --> A4 Arduino
- LCD I2C SCL --> A5 Arduino

Step 2: Write or Import Code

- Click on the **"Code"** button.
- Use either **Blocks** or **Text** (C/C++) mode to write your Arduino code.
- Include the required Libraries for the used components (LiquidCrystal I2C library for LCD i2c).
- Make sure your code matches the circuit design.

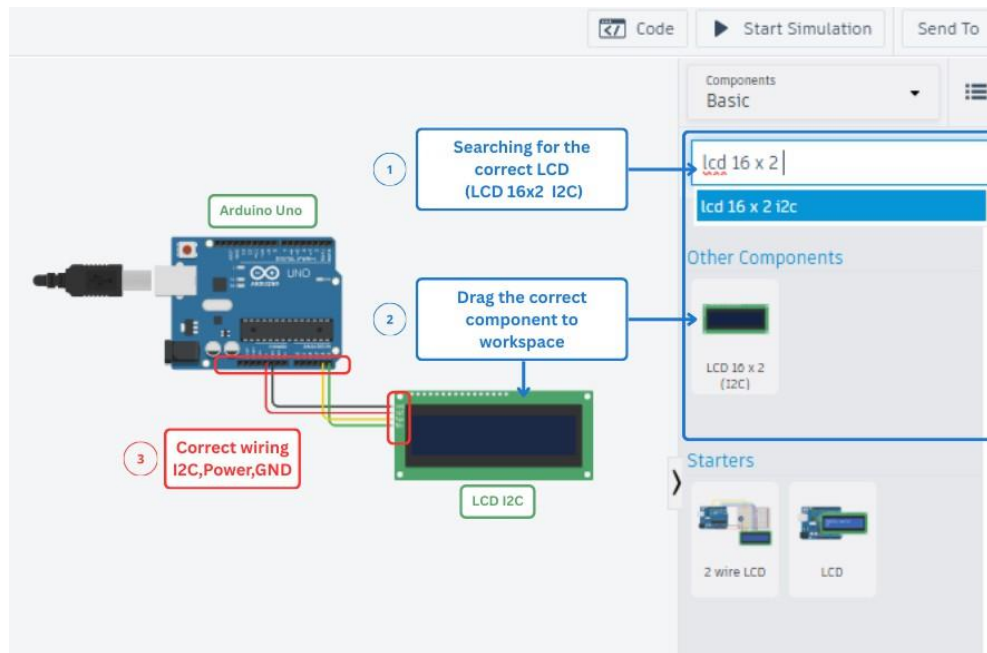


Figure 3.14. STEP1 Circuit design

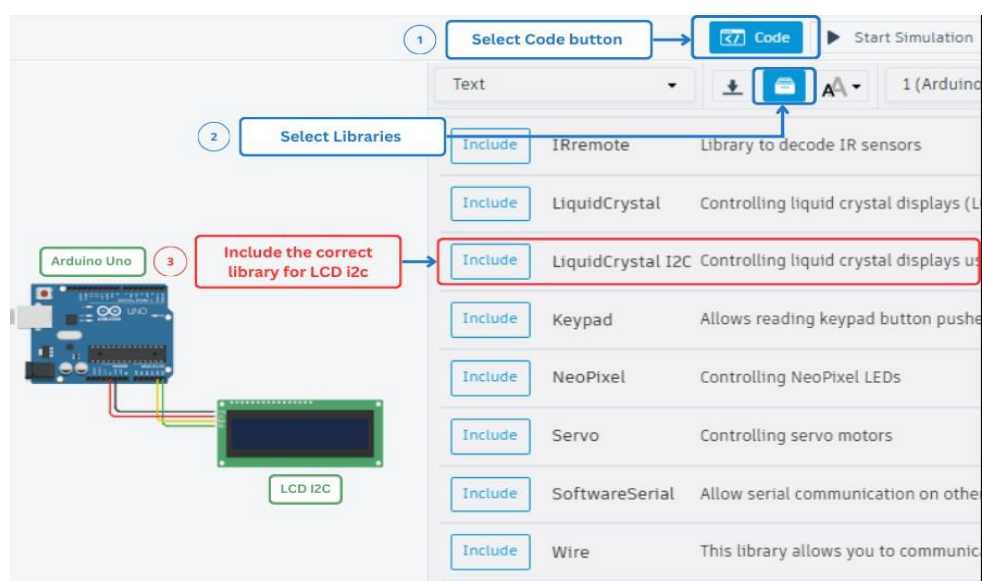


Figure 3.15. STEP2 Write or Import Code

Step 3: Start the Simulation & check for Wiring/Code errors

- Click the "Start Simulation" button at the top of the screen to run the project.
- Open the Serial Monitor to check for printed messages or debugging information.
- Look out for any code errors or output confirmation.

Observe Circuit Behavior

- Watch how components behave (LCD displays the correct message).
- Verify that everything functions as expected based on your code logic.

Stop the Simulation

- Click "Stop Simulation" to end the process and make any necessary changes.

Debug and Modify

- If something doesn't work correctly, revise your wiring or code.
- Repeat the simulation to test improvements.

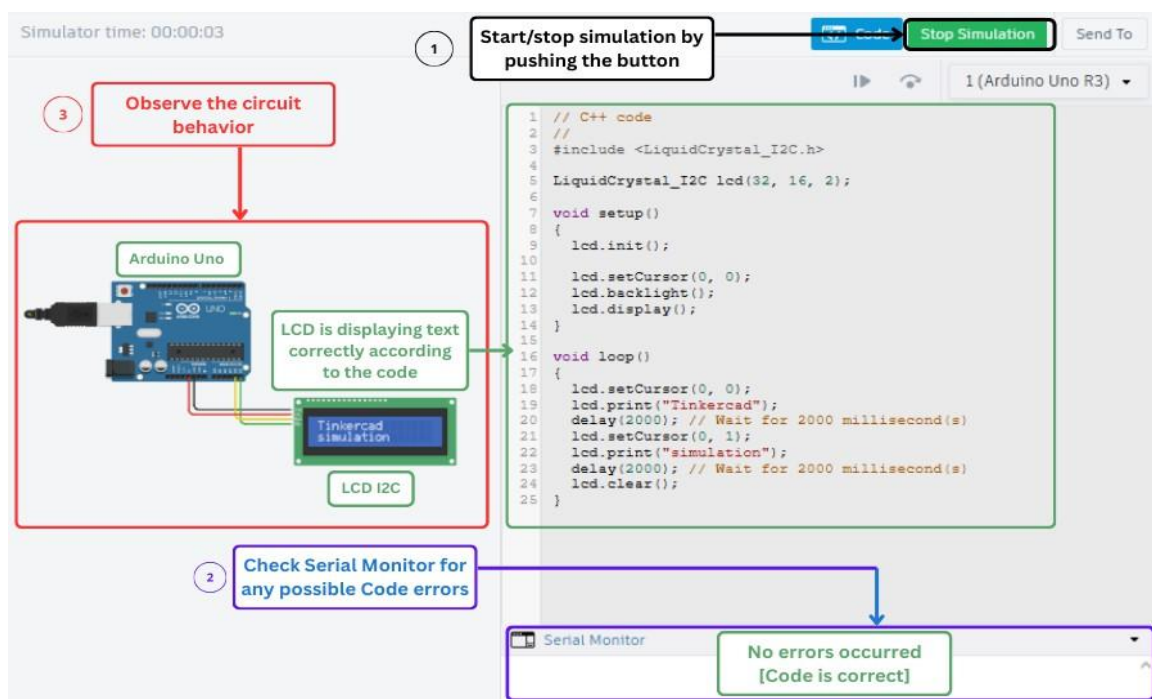


Figure 3.16. Start the Simulation & check for Wiring/Code errors

Chapter 4

Implementation & Testing

4.1 Introduction

In this chapter, we present the realization of our Arduino-based color sorting system. We begin with a Tinkercad simulation to test the logic and component interactions virtually. After that, we describe the wiring and schematics of the main components. Finally, we integrate the hardware and software and carry out tests to validate the system's functionality.

4.2 Project environment

We chose to develop a conveyor belt with a color sorting mechanism as our project because it represents a practical application of automation and embedded systems, commonly used in industrial sorting and quality control processes (Recycling, Production lines). The project allowed us to combine both hardware and software skills while addressing real-world challenges such as object detection, classification, and motion control.

To implement this system with improved performance and functionality, we selected the following components:

- **Sensors:**
 - TCS34725 color sensor – for precise color detection.
- **Actuators:**
 - 12V gear motor – to drive the conveyor belt.
 - 2x MG5010S servo motors – to handle the sorting mechanism.
- **Control and Interface:**
 - Arduino Uno – as the main control unit.
 - 16x2 LCD with I²C interface – to display system information.
 - L298N motor driver – to control the DC motor.
 - Breadboard and jumper wires – for wiring and prototyping connections.

4.2.1 Sorting logic:

The sorting process begins when the TCS34725 color sensor detects the RGB values of an object on the conveyor belt and sends the data to the Arduino. The Arduino processes the values to classify the object as red, green, or unclassified color. If the object is red or green and the respective stock has not yet reached its maximum capacity (3 pieces), the Arduino activates the corresponding servo motor (Servo 1 for red, Servo 2 for green) while the

conveyor belt continues moving, redirecting the object to its designated storage zone. The LCD displays updates in real time to show the current count of stored objects. If the detected color is neither the chosen ones, the servos remain inactive, and the object proceeds to the general stockpile. Once a storage bin reaches full capacity (3 objects), any additional objects of that color are treated as unclassified and automatically diverted to the general stockpile, ensuring continuous operation without interruptions.

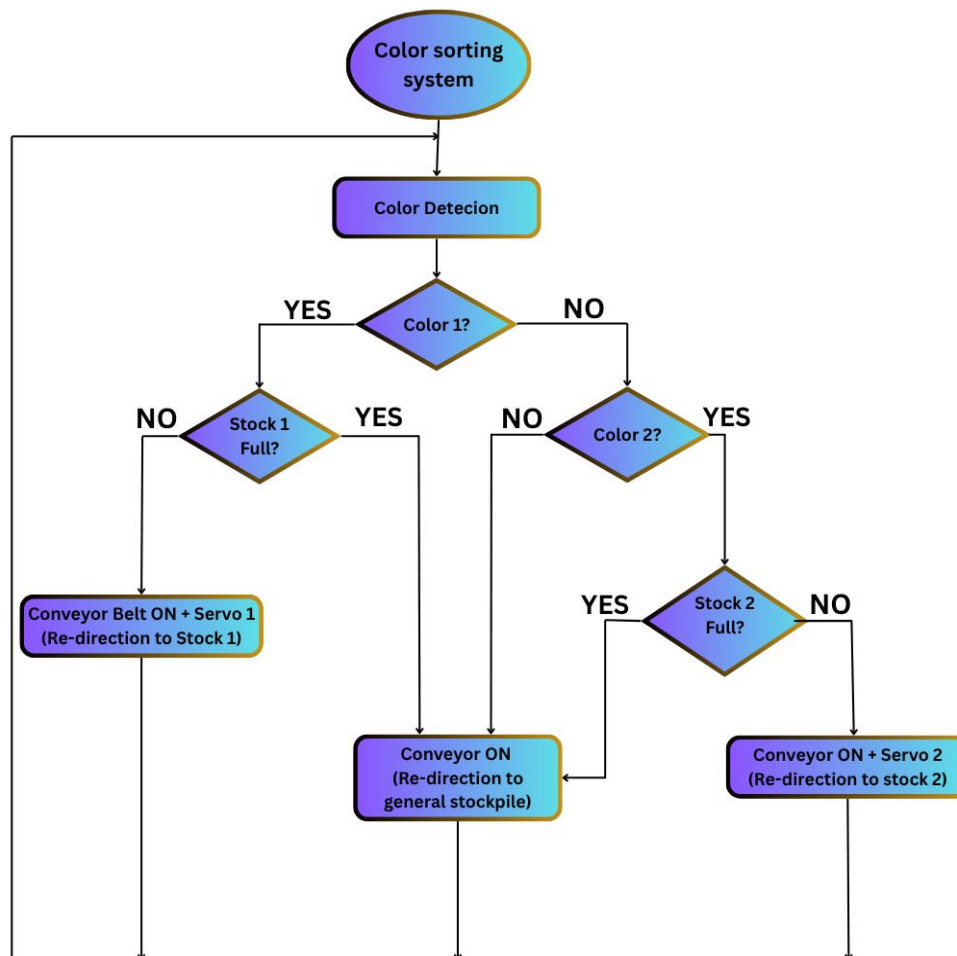


Figure 4.1. System's Logigram

4.3 Simulation

The development of our color sorting system began with a simulation in Tinkercad, starting with the circuit design. One of the main objectives of this simulation was to detect wiring errors and ensure correct system's mechanism and circuit safety, particularly to prevent short circuits and to avoid overload.

4.3.1 Power Supply and Wiring

In the simulation, we carefully tested the power supply configuration to avoid any overloads or short circuits in real implementation. The servomotors were powered with a 6V battery, the gear motor was powered through a 12V transformer connected to the motor driver, and the rest of the components were powered using the 5V output from the Arduino. All parts of the circuit were connected to a common ground to ensure stable operation.

4.3.2 Code and Library Integration

We uploaded and tested the Arduino code directly in Tinkercad. This included the use of essential libraries such as the “LiquidCrystal_I2C” library for controlling the LCD i2c display, as well as libraries for servomotors “Servo (built-in)” and special functions are used for motor control. The simulation helped verify that the logic executed correctly and that the code behaved as intended.

4.3.3 Color Detection Emulation

Since Tinkercad does not support the TCS34725 color sensor or any photodiode-based alternatives, we emulated its behavior using three pushbuttons. Each button represented an input corresponding to a specific color value: red, green, or blue. This allowed us to simulate the detection of different colored objects and test the system’s response.

4.3.4 Motor Control: Conveyor Mechanism

The conveyor system in the simulation was modeled using a gear motor controlled via an L293D motor driver, which replaced the unavailable L298N module. The gear motor simulated the conveyor’s motion by running when a color was detected.

4.3.5 Sorting Mechanism

The sorting action was controlled using two servomotors. When a red color input was triggered, Servo1 rotated to redirect the object to the red stockpile while the gear motor activated to move the object. A similar process occurred for green objects. In cases where the object was unclassified or a specific stockpile was full, only the gear motor was activated to carry the item to the general stockpile at the end of the conveyor. Furthermore, the LCD displays real time changes and the state of each stockpile.

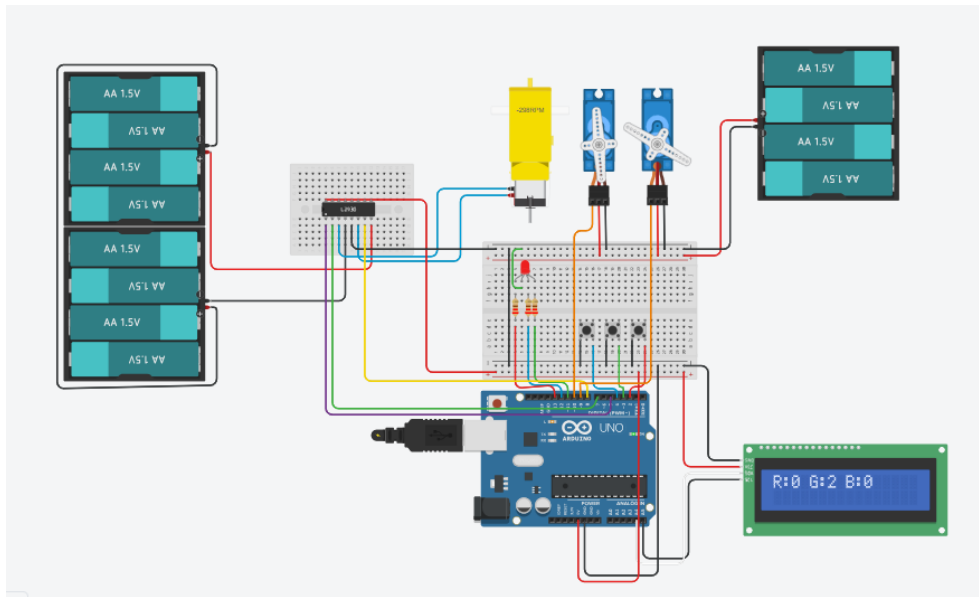


Figure 4.2. Tinkercad simulation

4.4 Electronics components wiring & programming

4.4.1 Servomotors

Power Supply Configuration:

The two servomotors used for sorting are powered by a 6V battery pack composed of four AAA batteries. The positive terminal of the battery pack is connected to the positive rail of the breadboard, while the negative terminal is connected to the ground rail. In order to establish a common ground reference between the power source and the Arduino Uno, the breadboard's ground rail is also connected to one of the Arduino's GND pins. This setup helps ensure proper voltage levels and stable communication between the Arduino and the servos.

Wiring with Arduino:

we have connected servomotors as it shown below:

- Servomotor 1 :
 - GND → Ground rail (connected to battery GND and Arduino GND)
 - VCC → 6V rail (from the battery)
 - Signal (yellow wire) → Arduino digital pin 9
- Servomotor 2 :
 - GND → Ground rail
 - VCC → 6V rail
 - Signal (yellow wire) → Arduino digital pin 10

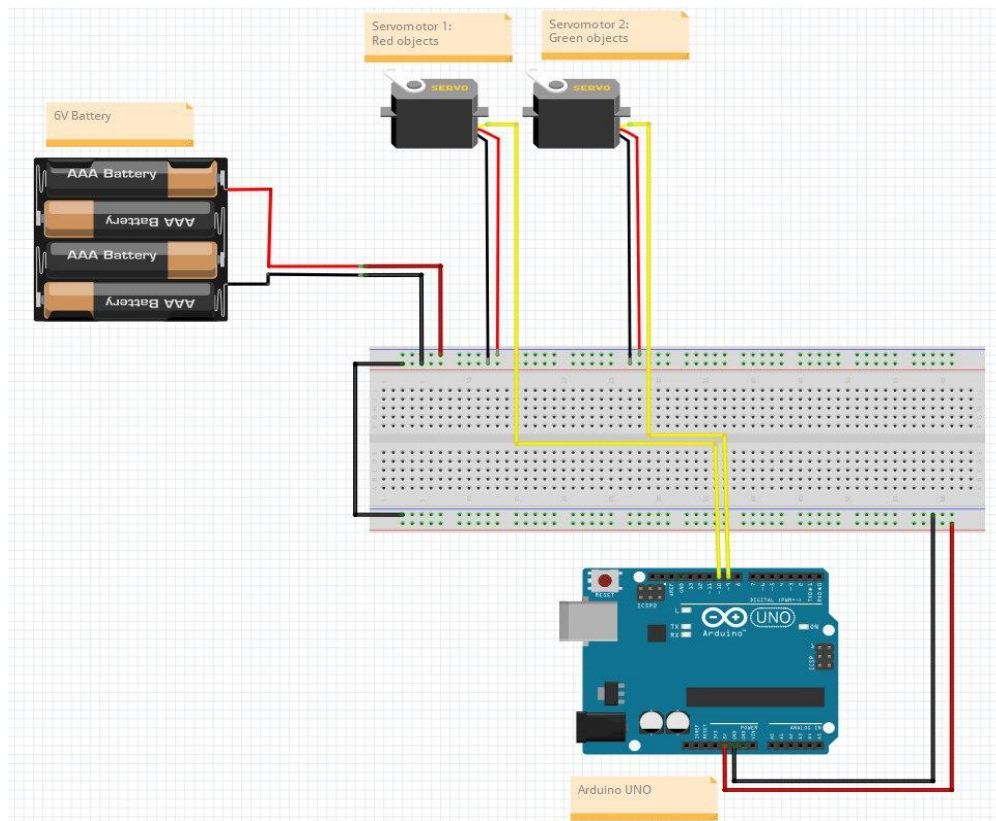


Figure 4.3. *Servomotors wiring*

Signal wires are connected directly to the Arduino’s PWM-capable digital pins, allowing independent control of each servo.

Testing and observations:

During testing, both servos responded accurately to the commands from the Arduino, indicating that the power and signal wiring was correctly implemented. The 6V power supply provided sufficient current to drive the servos without causing jitter or unexpected resets. Ensuring a common ground between the Arduino and the battery pack was essential for stable PWM signal interpretation.

4.4.2 Gear motor with L298N

Power Supply:

In this setup, a 12V transformer is used to drive the DC gear motor through the L298N motor driver. The power source is built from a 12V transform. The positive terminal of this supply is connected to the 12V input (VCC) on the L298N, and the negative terminal is connected to the GND pin. The Arduino Uno is powered separately and shares a common ground with the L298N to ensure proper signal transmission and avoid floating reference errors.

Wiring with Arduino:

The full wiring for the components is shown in the figure below:

- The DC gear motor is connected to the OUT1 and OUT2 terminals on the L298N motor driver. (These outputs control the direction and movement of the motor.)
- The IN1 and IN2 control pins on the L298N are connected to:
 - IN1 → Arduino digital pin 7
 - IN2 → Arduino digital pin 8
 These pins control the rotation direction of the motor.
- The EN1 (Enable A) pin is used for PWM-based speed control and is connected to:
 - EN1 → Arduino digital pin 5 (PWM-capable)
- The 5V jumper on the L298N module is removed, because:
 - The motor is powered by an external 12V power source.
 - The onboard 5V regulator is not needed to power the logic or Arduino.

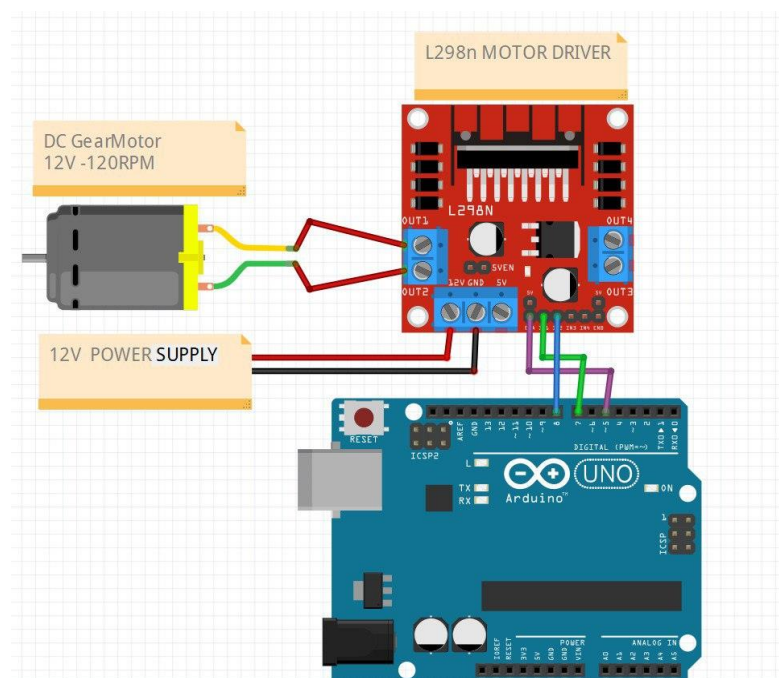


Figure 4.4. Gearmotor+L298N wiring

To control the motor from the Arduino, a simple sketch was uploaded to test both direction and speed. The code sets the enable pin to PWM (via “analogWrite”) and controls IN1 and IN2 to rotate the motor in either direction. For example, setting IN1 HIGH and IN2 LOW moves the motor forward, while reversing them changes the direction. The enable pin can be modulated to adjust speed dynamically.

Testing and Observations:

During testing, the motor responded correctly to the direction and speed control commands. The gear motor operated smoothly with the 12V supply, delivering sufficient torque for conveyor operation. It was crucial to ensure a common ground between the L298N and the Arduino Uno to maintain reliable communication.

4.4.3 LCD I2C

Power Supply Configuration:

The I2C LCD module is powered directly from the Arduino Uno's 5V and GND pins. The VCC pin of the LCD's I2C backpack is connected to the Arduino's 5V output, while the GND pin is connected to the Arduino's GND. This ensures the LCD receives a regulated 5V power supply suitable for both the display and the I2C communication module. Since the current draw of the LCD is relatively low, it can be safely powered from the Arduino without requiring an external power source.

Wiring with Arduino:

The LCD module is connected to the Arduino via the I2C interface using four pins as shown:

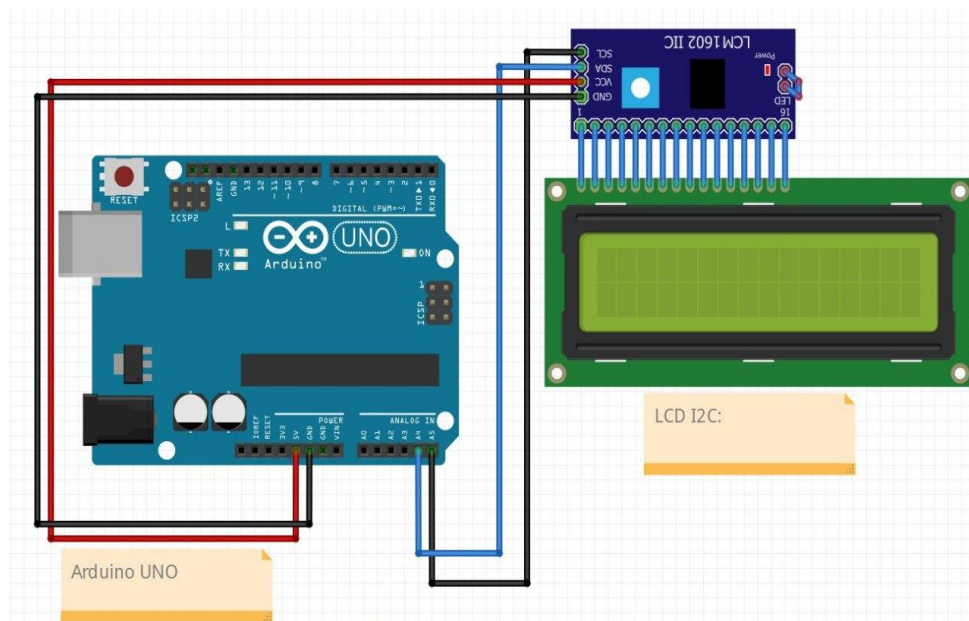


Figure 4.5. LCD I2C wiring

- **GND** → Arduino **GND** (Provides ground reference).
- **VCC** → Arduino **5V** (Powers the I2C backpack and the LCD module).
- **SDA** → Arduino **A4** (Serial Data line for I2C communication).
- **SCL** → Arduino **A5** (Serial Clock line for I2C communication).

The LCD module shown is a 1602 (16x2) character display with an I²C backpack, which simplifies connections compared to using 16 pins in parallel mode. The contrast and backlight are controlled via the potentiometer and jumper on the I²C board.

Address scanner code:

In order to extract the exact i2c address we used this i2c scanner code:

This scanner code is really important to identify any devices that are connected to the i2c bus on Arduino board and report any devices that respond by systematically checking each possible I2C address.

The code initializes the I2C communication with "Wire.begin()", then iterates through all addresses "Wire.beginTransmission(address)", check if device responds by

"Wire.endTransmission()", its hexadecimal address is printed to the Serial Monitor.

This helped us to verify correct wiring, detect address conflicts, and confirm whether sensors or modules are working properly.

```
#include <Wire.h>
void setup() {
  Wire.begin();
  Serial.begin(9600);
  Serial.println("I2C Scanner");
}
void loop() {
  byte error, address;
  for (address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0) {
      Serial.print("Found device at 0x");
      Serial.println(address, HEX);
    }
  }
  delay(5000);
}
```

Figure 4.6. I2C Address Scanner code

Testing and Observations:

The LCD I2C module was successfully initialized. I2C communication was confirmed at address 0x27 through scanner verification code.

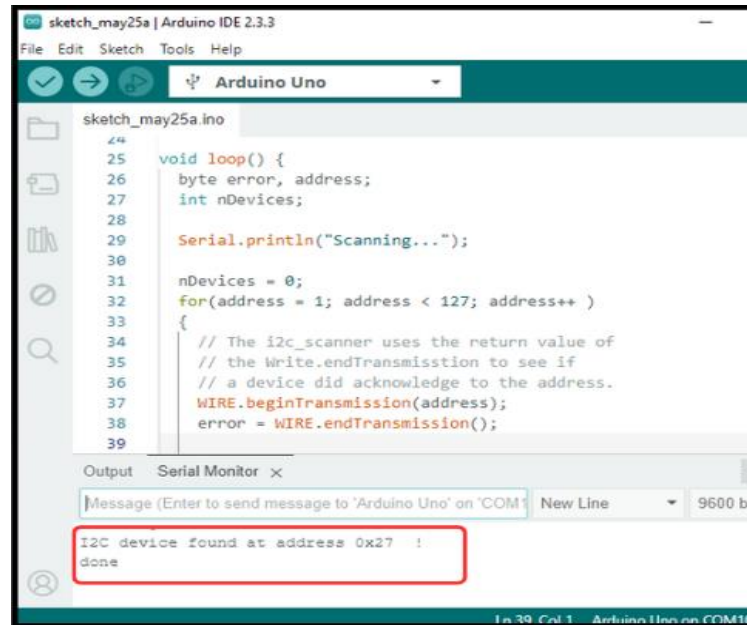


Figure 4.7. LCD I2C Adress

The display handled text alignment, backlight control, and custom characters (like a heart symbol) as expected, though with an 8-character storage limit, with stable operation under normal voltage conditions ($5V \pm 0.5V$). These tests confirm the module's reliability for typical embedded applications while identifying key operational constraints.



Figure 4.8. LCD I2C display test

4.4.4 TCS34725 Color Sensor

Power Supply Validation

The TCS34725 color sensor was powered directly from the Arduino's 5V output pin, with a common ground connection established between the sensor and Arduino board. This configuration provided stable power for initial testing and operation of the color detection functions. The sensor's VIN pin was connected to 5V, while its GND pin was tied to the Arduino's ground to complete the circuit. All communication and power connections were verified with a multimeter before testing to ensure proper voltage levels and continuity. This basic power arrangement allowed the sensor to function for preliminary color measurement applications.

Wiring with Arduino Board:

The TCS34725 color sensor was connected to the Arduino Uno board using the following wiring configuration:

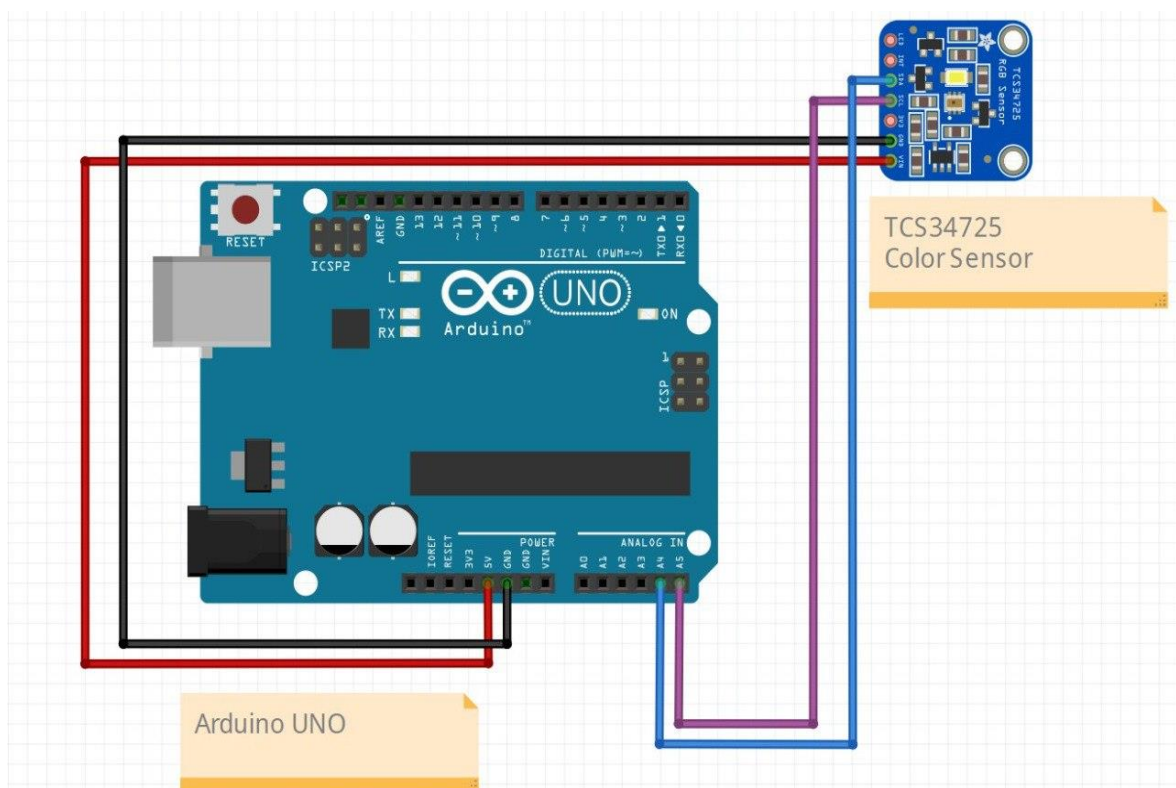


Figure 4.9. TCS34725 wiring

Pins connections:

- VIN (Sensor) → 5V (Arduino)
- GND (Sensor) → GND (Arduino)
- SDA (Sensor) → A4 (Arduino)
- SCL (Sensor) → A5 (Arduino)

Address scanner code:

The TCS34725 color sensor functioned properly when powered by the Arduino's 5V pin with a common ground connection. Furthermore, we have extracted the correct address of the color sensor via Address scanner code as shown in the figure 4.10 below:

The sensor was correctly detected at I2C address 0x29 (figure), confirming successful communication.

The integrated LED illuminated objects effectively when enabled, allowing reliable color detection of test samples (red, green, blue).

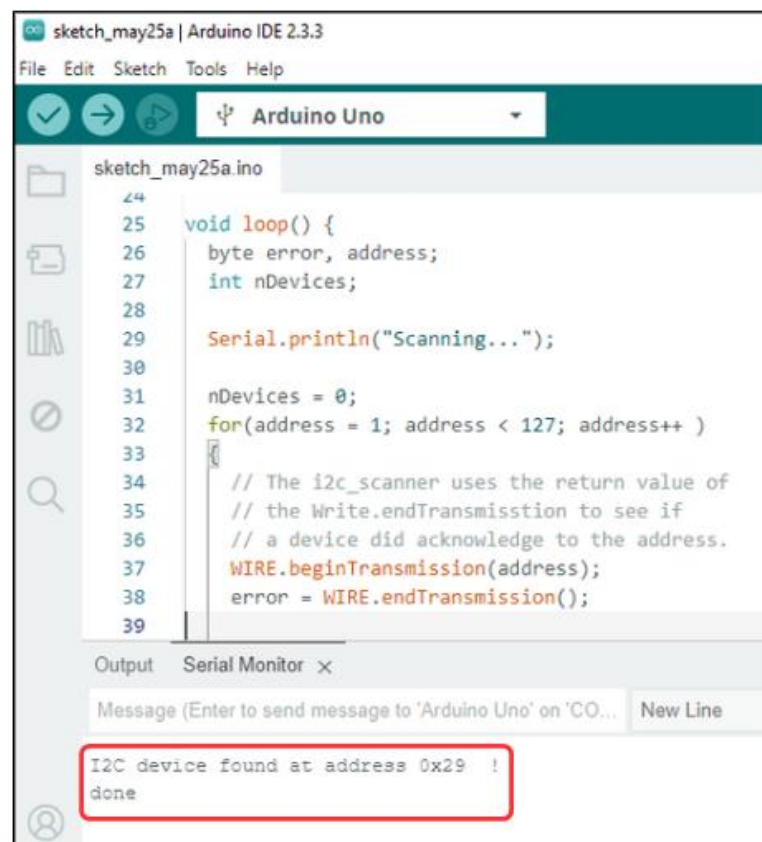


Figure 4.10. TCS34725 I2C Address

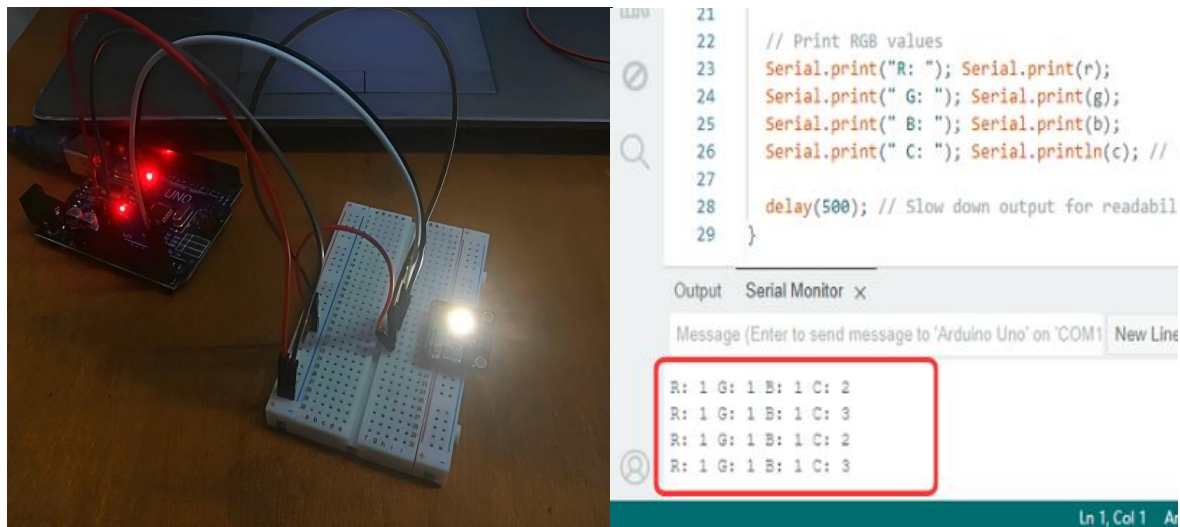


Figure 4.11. *TCS34725 test*

RGB values showed clear differentiation between primary colors, verifying the sensor's basic functionality in this configuration.

4.5 Color Calibration Methodology for an Automated Sorting System:

During preliminary testing, we evaluated the system using standardized color samples (red, green, blue, white, yellow, and orange) and with a simple code we observed significant challenges in color discrimination. The primary issues included:

1. **Excessive Detection Range for Red**

The sensor's broad response to red wavelengths caused misclassification of analogous hues, particularly yellow and orange, which were erroneously identified as red due to spectral overlap.

2. **White-Green Confusion**

White samples ($RGB \approx 255, 255, 255$) were frequently misidentified as green ($RGB \approx 0, 255, 0$), suggesting inadequate normalization for ambient illumination or insufficient separation in the green channel's dynamic range.

3. **Blue-Black Ambiguity**

Low-light conditions exacerbated the sensor's difficulty in distinguishing dark blue ($RGB \approx 0, 0, 128$) from true black ($RGB \approx 0, 0, 0$), highlighting limitations in the sensor's signal-to-noise ratio for short-wavelength detection.

The hardware implementation phase required: an Arduino Uno development board served as the central processing unit, interfaced with both a TCS34725 color sensor and I2C LCD module through carefully configured electrical connections as shown in the figure below:

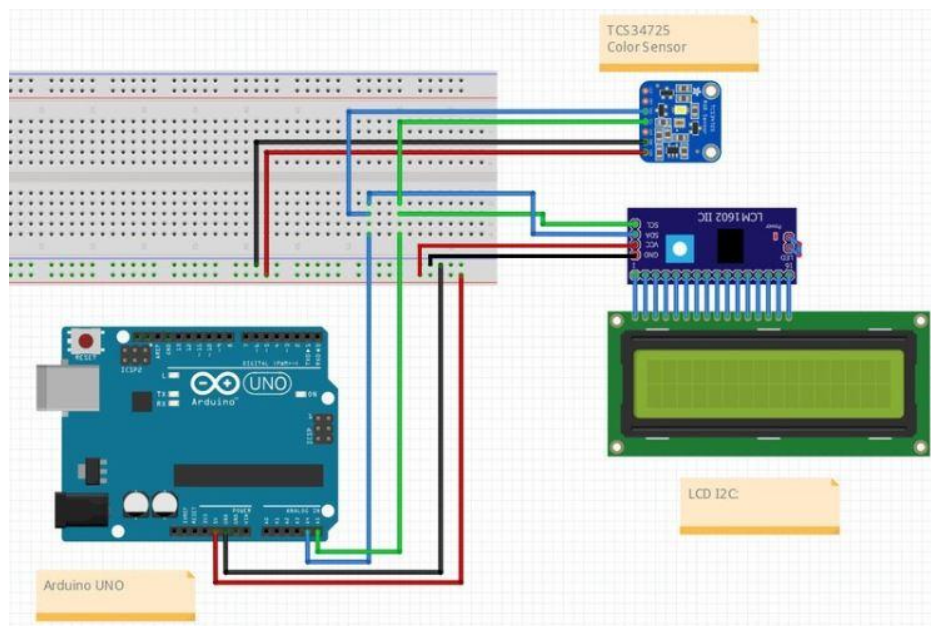


Figure 4.12. Calibration circuit design

Pin mapping:

TCS34725 with Arduino :

- VIN → Arduino 5V
- GND → Common ground
- SDA → Analog pin A4 (I²C data)
- SCL → Analog pin A5 (I²C clock)

LCD I2C with Arduino :

- VCC → Arduino 5V
- GND → Common ground
- SDA → Analog pin A4 (I²C data)
- SCL → Analog pin A5 (I²C clock)

We insured a common ground for all components by uploading required libraries.

4.5.1 Primary Color Testing Using Raw RGB Thresholds

The initial phase of this study involved testing the color detection system using conventional RGB value thresholds. The TCS34725 color sensor was interfaced with an Arduino Uno microcontroller, with detection results displayed on a 16×2 I2C LCD i2c module.

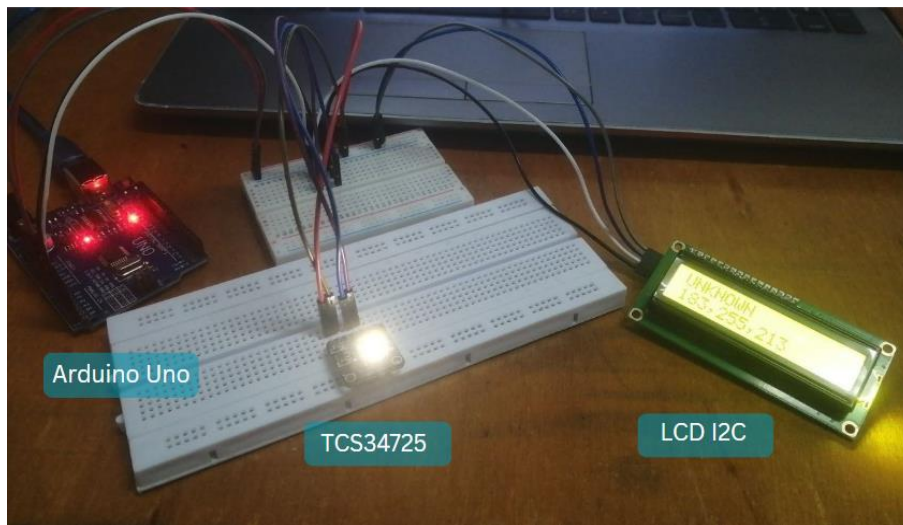


Figure 4.13. *hardware implementation*

Basic conditional statements were implemented to classify colors based on the relative intensities of red, green, and blue components.

4.5.2 Enhanced Calibration Using Hex Values in Processing IDE

To address the limitations of basic RGB thresholding, a secondary calibration phase was implemented using Processing 4.0. This software environment provided real-time visualization of color detection and enabled precise hexadecimal value analysis. The system was modified to output both RGB and hexadecimal color codes (e.g., #FF0000 for red) through serial communication.



Figure 4.14. *Calibration via Processing interface*

The experimental methodology involved:

1. Systematic sampling of reference colors under controlled illumination
2. Recording of hexadecimal values for each color category
3. Statistical analysis to determine optimal detection ranges
4. Iterative threshold adjustment to minimize classification errors

The final phase translated the hexadecimal calibration data into optimized Arduino code.

Color basic principles:

In color theory and digital color processing, hue, saturation, and brightness are three essential components used to describe and manipulate colors in a more intuitive way than RGB values.

Hue defines the type of color, **saturation** controls how vivid that color appears, and **brightness** adjusts how light or dark the final color looks together, they create a complete and flexible way to describe any visible color.

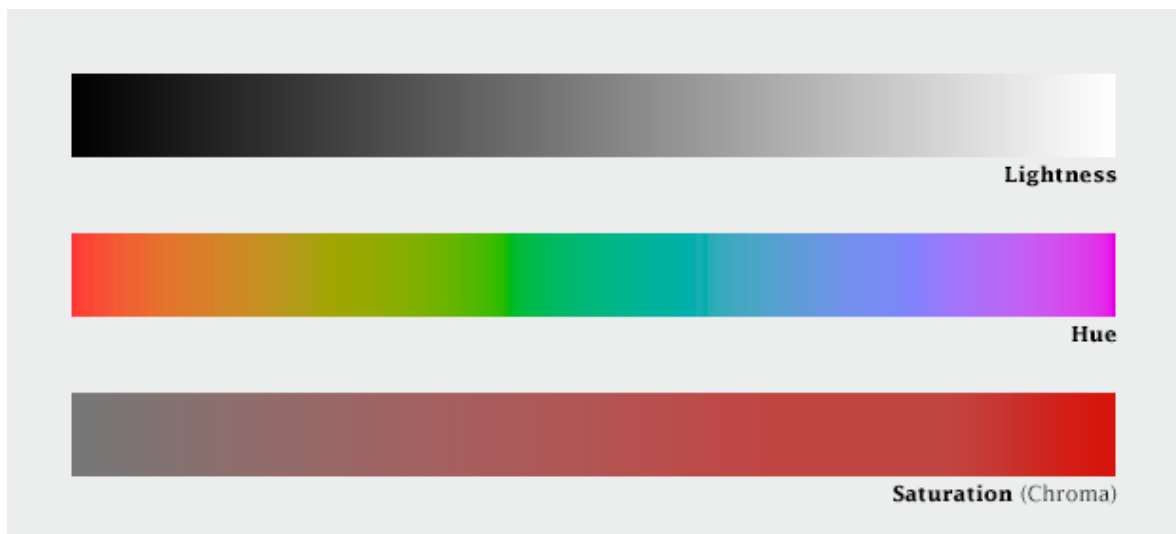


Figure 4.15. *Lightness, Hue, Saturation*

The hue value ranges used to classify the detected colors are outlined below:

Red: $\text{Hue} \leq 10^\circ$ or $\text{Hue} \geq 350^\circ$

Green: $\text{Hue} \geq 70^\circ$ and $\text{Hue} \leq 160^\circ$

Yellow: $\text{Hue} \geq 40^\circ$ and $\text{Hue} \leq 70^\circ$

And for the blue detection we used **RGB-based** directly, not HSV-based like hue. It focuses on intensity and color balance rather than angle values like hue.

4.6 Full System's circuit design

After testing each component separately, we achieved this circuit design insuring compability and sufficient power supply and insuring common ground and high performance with an Arduino code.

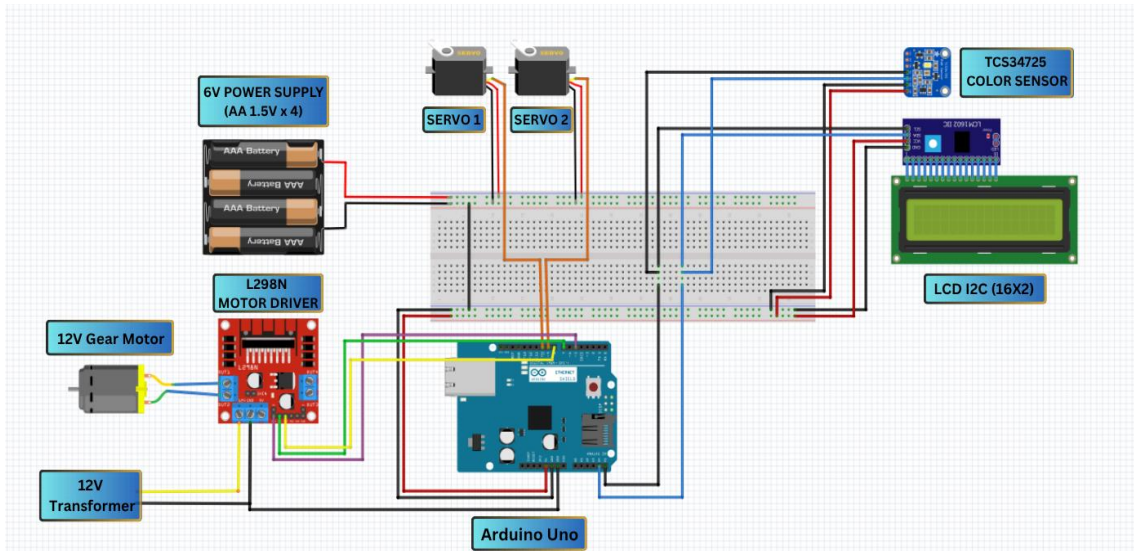


Figure 4.16. Final circuit design

Table 4.6. Pin Configuration and Wiring Connections of the Components.

Component	Pin	Connects To
TCS34725 Color Sensor	VCC	Arduino 3.3V or 5V
	GND	Arduino GND
	SDA	Arduino A4
	SCL	Arduino A5
Servo Motor 1 (SG-5010)	Signal (Orange)	Arduino D9
	Power (Red)	6V Battery Positive Terminal
	Ground (Brown)	6V Battery Negative Terminal + Arduino GND
Servo Motor 2 (SG-5010)	Signal (Orange)	Arduino D10
	Power (Red)	6V Battery Positive Terminal
	Ground (Brown)	6V Battery Negative Terminal + Arduino GND

L298N Motor Driver	ENA	Arduino D5 (PWM)
	IN1	Arduino D7
	IN2	Arduino D8
	OUT1 & OUT2	12V Gear Motor Terminals
	12V+	12V Battery Positive Terminal
	GND	12V Battery Negative Terminal + Arduino GND
	5V (optional)	Arduino 5V (only if onboard regulator used)
LCD I2C Module	VCC	Arduino 5V
	GND	Arduino GND
	SDA	Arduino A4
	SCL	Arduino A5
6V Battery	Positive Terminal	Servo Motors' Power (Red)
	Negative Terminal	Servo Motors' Ground (Brown) + Arduino GND
12V Battery	Positive Terminal	L298N 12V+
	Negative Terminal	L298N GND + Arduino GND
Arduino Power		Via USB or Barrel Jack

4.7 Mechanical structure and material choices

Our color sorting machine prototype uses simple, cost-effective materials suitable for a school project:

- **Wooden frame**
- **PVC cylinders on steel shafts**
- **Polyurethane (PU) belt between cylinders**
- **Toothed belt and pulleys**

This design ensures reliable operation while keeping construction simple and affordable.

Key Components and Design Choices

1. Wooden Frame Construction

The frame provides stable support for all components, and we used it because it is easy to work with and modifiable and for its low cost.

2. PVC Cylinders with Steel Shafts

- **PVC pipes** offer lightweight, affordable rotating surfaces
- **Steel shafts** provide necessary strength and straight rotation

Attachment method: Glue and screw + nut was used.



Figure 4.17. Conveyor body structure

4. Object Transport System

PU belt connects two main cylinders:

- Provides grip for moving objects
- Flexible enough for smooth operation



Figure 4.18. *PU belt Material*

5. Toothed Belt and Pulleys Drive System

The toothed belt connecting motor to shaft offers:

- **Precise power transmission**
- **Better speed synchronization**
- **Higher torque capacity**

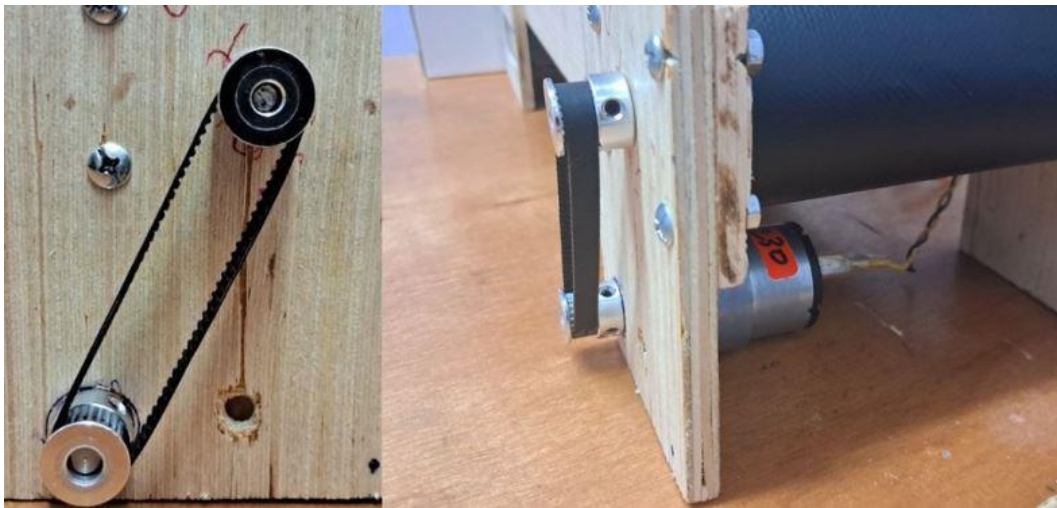


Figure 4.19. *Toothed Belt and Pulleys Drive System*

4.8 Mechanical Validation

4.8.1 Conveyor Belt Functionality

The conveyor belt was evaluated for alignment consistency and proper tensioning. Minor lateral drift was corrected using side guides. The belt tension was adjusted to prevent slippage without overloading the motor.

4.8.2 Drive Mechanism Testing

Efficiency tests on the pulley and toothed belt system revealed minimal mechanical losses, maintaining over 90% torque transmission.

The gearmotor's PWM input was correlated to varying loads. Speed stability was maintained.

4.9 Electronic Component Calibration

4.9.1 Servomotor Precision

Sorting accuracy was optimized by adjusting servo angles from the default 90° to 144°, which provided clearer bin separation.

Response times were estimated based on motor speed and object redirection.

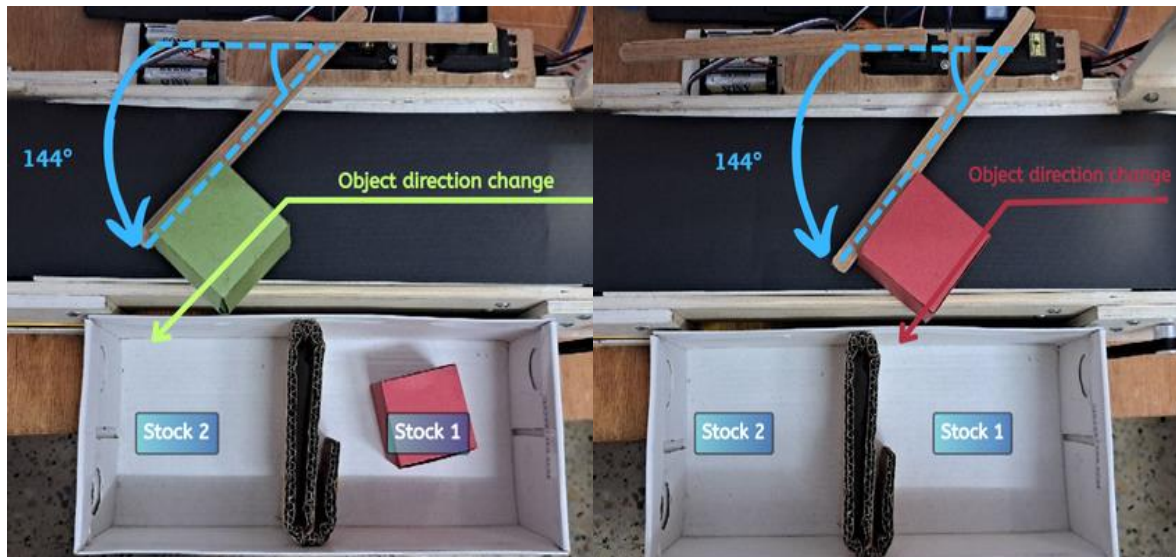


Figure 4.20. *Servos angle calibration*

In a test of 4 different colored objects, the system achieved 93% classification accuracy. Misclassifications primarily occurred under low contrast or edge-shadowed conditions.

4.10 Software and Control Systems

4.10.1 Web Application implementation

The integration of a web-based control application significantly enhanced the system's flexibility and adaptability, addressing key limitations of the initial prototype. By developing an interactive dashboard hosted on a local server, the system transitioned from a static, hardcoded operation to a dynamic, user-configurable platform.

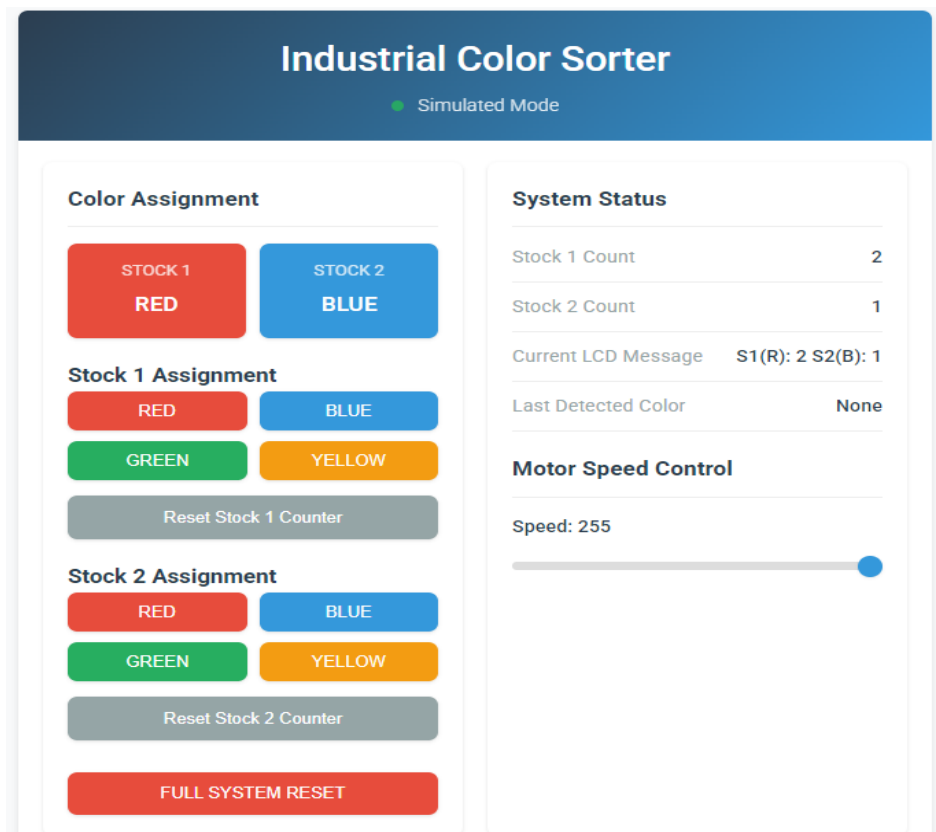


Figure 4.21. Web application User Interface

The web app allows real-time adjustments to color-to-bin assignments, servo angles, and motor speed without requiring physical access to the Arduino or manual code modifications, in order to avoid repetitive reflashing and hardware resets. This diagram illustrates the communication architecture of a color sorting system that utilizes an Arduino and a color sensor.

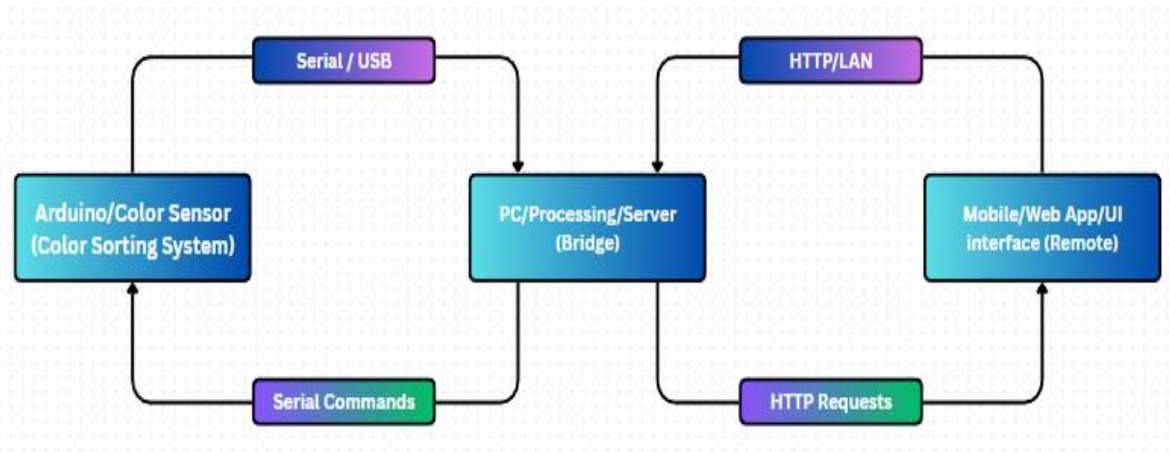


Figure 4.22. Color Sorting System Communication Architecture

The Arduino, which handles the color detection and sorting operations, connects to a PC or server via a Serial/USB interface. The PC acts as a bridge between the Arduino and a remote user interface. It sends and receives Serial Commands to/from the Arduino, facilitating real-time control and feedback. Simultaneously, the PC communicates with a Mobile or Web App through HTTP/LAN, enabling users to send HTTP Requests from the remote interface. This setup allows for remote monitoring and control of the color sorting system, making it accessible and manageable from various devices over a local network.

4.10.2 LCD Feedback Logic

The LCD displays real-time status of the color sorting system, showing active stock assignments (S1/S2) with their mapped colors and current item counts. It dynamically updates to reflect sorting operations, mode changes, and system alerts, providing immediate visual feedback for operator monitoring.



Figure 4.23. LCD Feedback screen

4.10.3 Manual Mode

We also added a manual mode that enabled direct control, reducing debugging time, Servos and motors were tested individually, which improved setup accuracy.

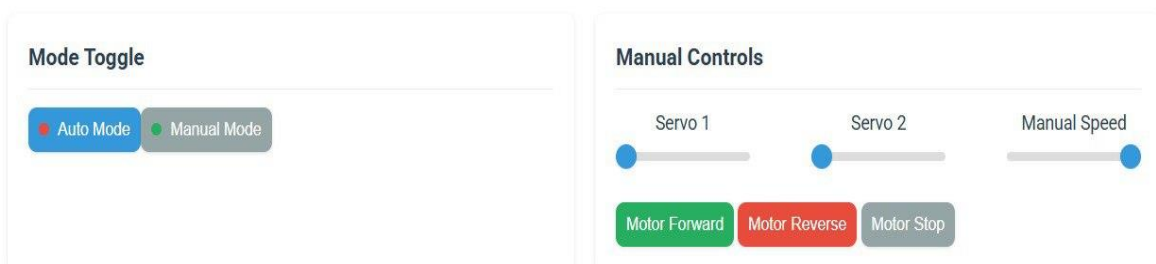


Figure 4.24. Manual mode

4.10.4 Remote Control Advantages

Color assignment was performed in <1s via the web interface, significantly faster than the 30s needed for code modification and reflash.

This eliminated the need for physical resets, improving usability.

4.11 Future Improvements

Our prototype works well, but there's plenty of room to take it further. Some potential improvements are :

- IOT communication (MQTT).
- Cloud-based monitoring.
- Database logging.
- Advanced alarm & Safety features.
- Predictive maintenance.
- Replace static servos with robotic arm.
- Replace the color sensor with a Camera module that works with AI (Image processing algorithms).

Conclusion

Conclusion

After thorough study and development, our project resulted in the creation of a color-based sorting system designed to classify objects in the middle or at the end of a production line.

This color sorting system serves as a flexible and functional prototype, suitable for educational purposes or adaptation to more complex industrial setups. It's hardware and software structure can be easily modified to support more advanced detection such as identifying different shades, materials, or shapes by incorporating additional sensors or vision systems.

The system also serves as a practical application of industrial automatism, combining real-time sensing, actuation, and autonomous decision-making. The inclusion of both automatic and manual operating modes allows efficient control, improved debugging, and adaptability to different production scenarios. Alarm functionalities have also been considered to enhance safety and reliability; for instance, alerts can be triggered when a bin is full, when a sensor malfunctions, or when a fault in the process is detected. These alarms can help prevent damage, reduce downtime, and maintain system efficiency.

While our system currently uses a local server for real-time monitoring and control, it can be further improved by integrating IoT technologies and MQTT communication protocols. This would allow remote monitoring, cloud-based data logging, and even predictive maintenance, making the system smarter and more scalable for industrial applications.

We hope this humble project will inspire future students and contribute as a practical foundation for developing more advanced automated sorting systems and embedded solutions.

References

References

- [1] https://en.wikipedia.org/wiki/Production_line/2025
- [2] <https://en.wikipedia.org/wiki/Automation/2025>
- [3] <https://en.wikipedia.org/wiki/Coating/2024>
- [4] <https://www.qarmainspect.com/blog/automated-quality-control/2025>
- [5] MASSINISSA IAMRACHE, Étude et réalisation d'un bras robotique, mémoire de fin d'études, Master ELN, Université Mouloud Mammeri de Tizi Ouzou (UMMTO), 2017.
- [6] <https://www.nabto.com/development-board-guide/2020>
- [7] <https://www.elektormagazine.com/news/development-boards-past-present-and-posterity>, 7 July 2022.
- [8] <https://docs.arduino.cc/hardware/uno-rev3/2024>
- [9] <https://www.jameco.com/Jameco/workshop/CircuitNotes/CN-Arduino-uno.html/2018>
- [10] <https://docs.arduino.cc/learn/communication/uart/2024>
- [11] <https://www.circuitbasics.com/basics-uart-communication/2019>
- [12] <https://docs.arduino.cc/learn/communication/spi/2024>
- [13] <https://ucbeginner.com/arduino-spi-guide/2022>
- [14] <https://myhomethings.eu/en/arduino-spi-communication-serial-peripheral-interface/2023>
- [15] <https://www.rfwireless-world.com/terminology/other-wireless/spi-advantages-and-disadvantages/2021>
- [16] <https://dronebotworkshop.com/i2c-arduino-arduino/2020>
- [17] https://www.waveshare.com/wiki/TCS34725_Color_Sensor/2023
- [18] https://www.egr.msu.edu/classes/ece480/capstone/fall15/group03/document/ECE48_ApplicationNote_Alex.pdf/2015
- [19] <https://www.totalphase.com/blog/2016/08/advantages-limitations-i2c-communication/2016>

- [20] M. A. Khan et al., A Comprehensive Review on I2C Communication Protocol, *Procedia Computer Science*, vol. 172, pp. 123–130, 2020.
- [21] A. R. M. Zain et al., Prototype Low-Cost Portable Electrocardiogram (ECG) Based on Arduino-Uno with Bluetooth Feature, *ResearchGate*, 2019.
- [22] <https://www.elprocus.com/different-types-of-arduino-boards/2021>
- [23] <https://circuitdigest.com/article/different-types-of-arduino-boards/2022>
- [24] <https://www.icdrex.com/l298n-a-motor-driver-that-accepts-high-voltages/2021>
- [25] <https://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf/2018>
- [26] https://en.wikipedia.org/wiki/Pulse-width_modulation/2024
- [27] <https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/2018>
- [28] <https://learn.sparkfun.com/tutorials/pulse-width-modulation/duty-cycle/2020>
- [29] <https://en.wikipedia.org/wiki/H-bridge/2024>
- [30] <https://marobotic.com/2023/10/22/line-follower-obstacle-avoiding-robot-using-arduino-and-l298-motor-driver/2023>
- [31] <https://dronebotworkshop.com/dc-motor-drivers/2021>
- [32] <https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/2021>
- [33] H. Abdelatif, *Cours Capteurs et Actionneurs Industriels, Département de Maintenance en Instrumentation, Master 1 MAI, IMSI*, 2024.
- [34] <https://www.micromotors.eu/en/what-is-a-gear-motor/2021>
- [35] <https://mechtex.com/blog/what-is-a-dc-gear-motor/2022>
- [36] <https://www.powerelectric.com/motor-blog/what-are-gear-motors-used-for/2021>
- [37] <https://www.a-m-c.com/servomotor/2022>
- [38] <https://www.yaskawa-global.com/product/servomotor/history/2023>
- [39] <https://www.sparkfun.com/servos/2024>
- [40] Autsou, S. et al., Principles and Methods of Servomotor Control: Comparative Analysis and Applications, *Applied Sciences*, vol. 14, no. 6, 2579, 2024,

- [41] <https://www.electrical4u.com/what-is-servo-motor/2023>
- [43] <https://www.amazon.co.uk/Adafruit-TCS34725-Color-Sensor-Filter/dp/B00OKCRU5M/2017>
- [44] <https://protosupplies.com/product/tcs34725-rgb-color-sensor-module/2021>
- [45] <https://www.etechnog.com/2025/01/color-sensor-tcs34725-pinout-diagram.html/2025>
- [46] <https://www.mdpi.com/1424-8220/19/8/1750/2019>
- [47] <https://www.luisllamas.es/en/arduino-rgb-color-sensor-tcs34725/2023>
- [48] <https://www.otronic.nl/en/rgb-sensor-module-tcs34725.html/2022>
- [49] <https://electroxbd.com/product/tcs34725-color-sensor-rgb-sensor-board/2024>
- [50] <https://focuslcds.com/lcd-resources/i2c-display-communication/2022>
- [51] <https://lastminuteengineers.com/arduino-1602-character-lcd-tutorial/2020>
- [52] <https://components101.com/modules/i2c-serial-interface-adapter-module/2020>
- [53] <https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics/2023>
- [54] <https://www.opta.findernet.com/en/tutorial/getting-started-ide-v2/2021>
- [55] <https://docs.arduino.cc/learn/starting-guide/getting-started-arduino/2024>
- [56] <https://docs.arduino.cc/language-reference/en/structure/sketch/setup/2024>
- [57] <https://roboticsbackend.com/arduino-setup-loop-functions-explained/2021>
- [58] <https://docs.arduino.cc/language-reference/en/structure/sketch/loop/2024>
- [59] <https://www.digikey.com/en/maker/tutorials/2021/an-introduction-to-arduino-libraries/2021>
- [60] <https://en.wikipedia.org/wiki/Tinkercad/2024>
- [61] <https://www.tinkercad.com/projects/Basics-of-Arduino-TINKERCAD/2023>
- [62] <https://docs.arduino.cc/learn/programming/variable/2023>

Annexes

Annex A: Color thresholds code

```
// Check if the color is YELLOW
bool isYellow(RGB c) {
    float hue = c.hue(), brightness = c.brightness();
    return (
        // Yellow hue, high R & G, low B
        (hue >= 40 && hue <= 70 && c.r > 120 && c.g > 120 && c.b < max(c.r, c.g) * 0.6 && abs(c.r - c.g) < 60) ||

        // Bright yellow variant
        (brightness > 200 && c.r > 180 && c.g > 180 && c.b < 150 && hue >= 30 && hue <= 80)
    );
}

// Check if the color is GREEN
bool isGreen(RGB c) {
    float hue = c.hue(), brightness = c.brightness();
    return (
        // Green hue + strong G
        (hue >= 70 && hue <= 160 && c.g > c.r * 1.3 && c.g > c.b * 1.3 && c.g > 80 && abs(c.r - c.b) < 100) ||

        // Bright green
        (brightness > 180 && c.g > 150 && c.g > c.r * 1.2 && c.g > c.b * 1.2 && hue >= 60 && hue <= 170)
    );
}

// Check if the color is BLACK
bool isBlack(RGB c) {
    return c.r < 50 && c.g < 50 && c.b < 50; // All RGB values low
}

// Check if the color is WHITE
bool isWhite(RGB c) {
    return c.r > 200 && c.g > 200 && c.b > 200; // All RGB values high
}

// Check if the color is RED
bool isRed(RGB c) {
    float hue = c.hue(), brightness = c.brightness(); // Get hue and brightness
    return (
        // Red hue + strong red value
        (hue >= 350 || hue <= 10) && c.r > c.g * 1.5 && c.r > c.b * 1.5 && c.r > 80 ||

        // Bright red with slightly relaxed rules
        (brightness > 180 && c.r > 150 && c.r > c.g * 1.3 && c.r > c.b * 1.3 && (hue >= 340 || hue <= 20))
    );
}

// Check if the color is BLUE
bool isBlue(RGB c) {
    float r = c.r, g = c.g, b = c.b;
    float total = r + g + b;
    float bRatio = b / total, rgBalance = abs(r - g);
    return (
        // Low light + strong blue
        (total < 150 && b > r * 2 && b > g * 2) ||
        // Balanced R & G, strong blue
        ((bRatio > 0.35) && b > r * 0.9 && b > g * 0.9 && rgBalance < 100 && b > 50) ||
        // Blue clearly higher than R & G
        (b > 150 && b > r + 30 && b > g + 30)
    );
}
```

Annex B: Code initialization

```
#include <Wire.h>           // For I2C communication
#include <Adafruit_TCS34725.h> // For TCS34725 color sensor
#include <LiquidCrystal_I2C.h> // For I2C LCD display
#include <Servo.h>          // For controlling servos

// Initialize TCS34725 sensor with integration time and gain
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_154MS, TCS34725_GAIN_60X);
|

// Initialize 16x2 LCD at I2C address 0x27
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Create two servo objects for sorting
Servo servo1;
Servo servo2;

// Motor driver pins (L298N)
const int ENA = 5; // PWM speed control pin
const int IN1 = 7; // Motor direction pin 1
const int IN2 = 8; // Motor direction pin 2

// Timing for status update
unsigned long lastStatusUpdate = 0;
const unsigned long STATUS_UPDATE_INTERVAL = 500; // Update every 500 ms

// Servo motor pins
const int SERVO1_PIN = 9;
const int SERVO2_PIN = 10;

// Motor speed and servo angles
int motorSpeed = 255; // Default full speed
const int SERVO_REST = 90; // Servo neutral position
const int SERVO_ACTIVE = 144; // Servo activation angle

// Bin counters
int stock1Count = 0;
int stock2Count = 0;
const int MAX_STOCK = 3; // Max items per bin

// Motor control state
int lastProcessedSpeed = -1;
unsigned long motorStartTime = 0;
bool motorRunning = false;
const unsigned long SORTING_DURATION = 3000; // 3 seconds of motor movement per sort

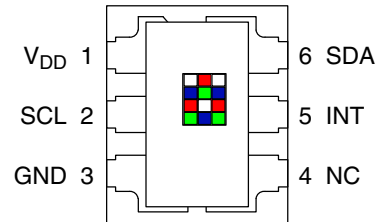
// Color detection state
String currentColor = ""; // Last detected color
String stock1MappedTo = "RED"; // Default assigned color for bin 1
String stock2MappedTo = "BLUE"; // Default assigned color for bin 2

// Mode flag (manual or auto)
bool manualMode = false; // Default is auto mode
```

Features

- Red, Green, Blue (RGB), and Clear Light Sensing with IR Blocking Filter
 - Programmable Analog Gain and Integration Time
 - 3,800,000:1 Dynamic Range
 - Very High Sensitivity — Ideally Suited for Operation Behind Dark Glass
- Maskable Interrupt
 - Programmable Upper and Lower Thresholds with Persistence Filter
- Power Management
 - Low Power — 2.5- μ A Sleep State
 - 65- μ A Wait State with Programmable Wait State Time from 2.4 ms to > 7 Seconds
- I²C Fast Mode Compatible Interface
 - Data Rates up to 400 kbit/s
 - Input Voltage Levels Compatible with V_{DD} or 1.8 V Bus
- Register Set and Pin Compatible with the TCS3x71 Series
- Small 2 mm $\times$ 2.4 mm Dual Flat No-Lead (FN) Package

PACKAGE FN
DUAL FLAT NO-LEAD
(TOP VIEW)



Package Drawing Not to Scale

Applications

- RGB LED Backlight Control
- Light Color Temperature Measurement
- Ambient Light Sensing for Display Backlight Control
- Fluid and Gas Analysis
- Product Color Verification and Sorting

End Products and Market Segments

- TVs, Mobile Handsets, Tablets, Computers, and Monitors
- Consumer and Commercial Printing
- Medical and Health Fitness
- Solid State Lighting (SSL) and Digital Signage
- Industrial Automation

Description

The TCS3472 device provides a digital return of red, green, blue (RGB), and clear light sensing values. An IR blocking filter, integrated on-chip and localized to the color sensing photodiodes, minimizes the IR spectral component of the incoming light and allows color measurements to be made accurately. The high sensitivity, wide dynamic range, and IR blocking filter make the TCS3472 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials.

The TCS3472 color sensor has a wide range of applications including RGB LED backlight control, solid-state lighting, health/fitness products, industrial process controls and medical diagnostic equipment. In addition, the IR blocking filter enables the TCS3472 to perform ambient light sensing (ALS). Ambient light sensing is widely used in display-based products such as cell phones, notebooks, and TVs to sense the lighting environment and enable automatic display brightness for optimal viewing and power savings. The TCS3472, itself, can enter a lower-power wait state between light sensing measurements to further reduce the average power consumption.

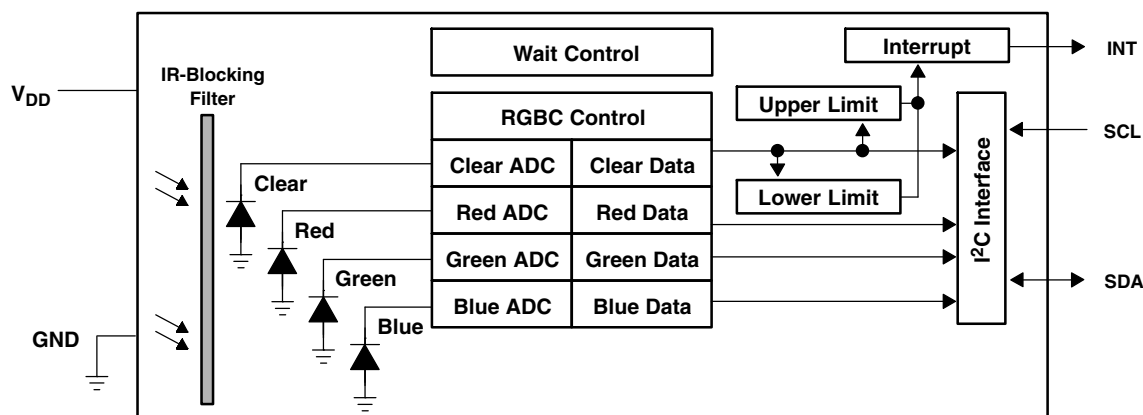
TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

TAOS135 – AUGUST 2012

Functional Block Diagram



Detailed Description

The TCS3472 light-to-digital converter contains a 3×4 photodiode array, four analog-to-digital converters (ADC) that integrate the photodiode current, data registers, a state machine, and an I²C interface. The 3×4 photodiode array is composed of red-filtered, green-filtered, blue-filtered, and clear (unfiltered) photodiodes. In addition, the photodiodes are coated with an IR-blocking filter. The four integrating ADCs simultaneously convert the amplified photodiode currents to a 16-bit digital value. Upon completion of a conversion cycle, the results are transferred to the data registers, which are double-buffered to ensure the integrity of the data. All of the internal timing, as well as the low-power wait state, is controlled by the state machine.

Communication of the TCS3472 data is accomplished over a fast, up to 400 kHz, two-wire I²C serial bus. The industry standard I²C bus facilitates easy, direct connection to microcontrollers and embedded processors.

In addition to the I²C bus, the TCS3472 provides a separate interrupt signal output. When interrupts are enabled, and user-defined thresholds are exceeded, the active-low interrupt is asserted and remains asserted until it is cleared by the controller. This interrupt feature simplifies and improves the efficiency of the system software by eliminating the need to poll the TCS3472. The user can define the upper and lower interrupt thresholds and apply an interrupt persistence filter. The interrupt persistence filter allows the user to define the number of consecutive out-of-threshold events necessary before generating an interrupt. The interrupt output is open-drain, so it can be wire-ORed with other devices.



Terminal Functions

TERMINAL NAME	NO.	TYPE	DESCRIPTION
GND	3		Power supply ground. All voltages are referenced to GND.
INT	5	O	Interrupt — open drain (active low).
NC	4	O	No connect — do not connect.
SCL	2	I	I ² C serial clock input terminal — clock signal for I ² C serial data.
SDA	6	I/O	I ² C serial data I/O terminal — serial data I/O for I ² C.
V _{DD}	1		Supply voltage.

Available Options

DEVICE	ADDRESS	PACKAGE – LEADS	INTERFACE DESCRIPTION	ORDERING NUMBER
TCS34721†	0x39	FN–6	I ² C V _{bus} = V _{DD} Interface	TCS34721FN
TCS34723†	0x39	FN–6	I ² C V _{bus} = 1.8 V Interface	TCS34723FN
TCS34725	0x29	FN–6	I ² C V _{bus} = V _{DD} Interface	TCS34725FN
TCS34727	0x29	FN–6	I ² C V _{bus} = 1.8 V Interface	TCS34727FN

† Contact TAOS for availability.

Absolute Maximum Ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V _{DD} (Note 1)	3.8 V
Input terminal voltage	–0.5 V to 3.8 V
Output terminal voltage	–0.5 V to 3.8 V
Output terminal current	–1 mA to 20 mA
Storage temperature range, T _{stg}	–40°C to 85°C
ESD tolerance, human body model	2000 V

† Stresses beyond those listed under “absolute maximum ratings” may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under “recommended operating conditions” is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltages are with respect to GND.

Recommended Operating Conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V _{DD} (TCS34721 & TCS34725) (I ² C V _{bus} = V _{DD})	2.7	3	3.6	V
Supply voltage, V _{DD} (TCS34723 & TCS34727) (I ² C V _{bus} = 1.8 V)	2.7	3	3.3	V
Operating free-air temperature, T _A	–30		70	°C

TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

TAOS135 – AUGUST 2012

Operating Characteristics, $V_{DD} = 3\text{ V}$, $T_A = 25^\circ\text{C}$ (unless otherwise noted)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{DD} Supply current	Active		235	330	μA
	Wait state		65		
	Sleep state — no I^2C activity		2.5	10	
V_{OL} INT, SDA output low voltage	3 mA sink current	0		0.4	V
	6 mA sink current	0		0.6	
I_{LEAK} Leakage current, SDA, SCL, INT pins		-5		5	μA
I_{LEAK} Leakage current, LDR pin		-5		5	μA
V_{IH} SCL, SDA input high voltage	TCS34721 & TCS34725	0.7 V_{DD}			V
	TCS34723 & TCS34727	1.25			
V_{IL} SCL, SDA input low voltage	TCS34721 & TCS34725		0.3 V_{DD}		V
	TCS34723 & TCS34727		0.54		

Optical Characteristics, $V_{DD} = 3\text{ V}$, $T_A = 25^\circ\text{C}$, $\text{AGAIN} = 16\times$, $\text{ATIME} = 0xF6$ (unless otherwise noted) (Note 1)

PARAMETER	TEST CONDITIONS	Red Channel			Green Channel			Blue Channel			Clear Channel			UNIT
		MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
R_e Irradiance responsivity	$\lambda_D = 465\text{ nm}$ Note 2	0%		15%	10%		42%	65%		88%	11.0	13.8	16.6	counts/ $\mu\text{W}/\text{cm}^2$
	$\lambda_D = 525\text{ nm}$ Note 3	4%		25%	60%		85%	10%		45%	13.2	16.6	20.0	
	$\lambda_D = 615\text{ nm}$ Note 4	80%		110%	0%		14%	5%		24%	15.6	19.5	23.4	

- NOTES: 1. The percentage shown represents the ratio of the respective red, green, or blue channel value to the clear channel value.
2. The 465 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics:
dominant wavelength $\lambda_D = 465\text{ nm}$, spectral halfwidth $\Delta\lambda_{1/2} = 22\text{ nm}$.
3. The 525 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics:
dominant wavelength $\lambda_D = 525\text{ nm}$, spectral halfwidth $\Delta\lambda_{1/2} = 35\text{ nm}$.
4. The 615 nm input irradiance is supplied by a AlInGaP light-emitting diode with the following characteristics:
dominant wavelength $\lambda_D = 615\text{ nm}$, spectral halfwidth $\Delta\lambda_{1/2} = 15\text{ nm}$.

RGBC Characteristics, $V_{DD} = 3\text{ V}$, $T_A = 25^\circ\text{C}$, $\text{AGAIN} = 16\times$, $\text{AEN} = 1$ (unless otherwise noted)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Dark ADC count value	$E_e = 0$, $\text{AGAIN} = 60\times$, $\text{ATIME} = 0xD6$ (100 ms)	0	1	5	counts
ADC integration time step size	$\text{ATIME} = 0xFF$	2.27	2.4	2.56	ms
ADC number of integration steps (Note 5)		1		256	steps
ADC counts per step (Note 5)		0		1024	counts
ADC count value (Note 5)	$\text{ATIME} = 0xC0$ (153.6 ms)	0		65535	counts
Gain scaling, relative to 1 $\times$ gain setting	4 $\times$	3.8	4	4.2	$\times$
	16 $\times$	15	16	16.8	
	60 $\times$	58	60	63	

NOTE 5: Parameter ensured by design and is not tested.



PARAMETER	TEST CONDITIONS	CHANNEL	MIN	TYP	MAX	UNIT
Wait step size	WTIME = 0xFF		2.27	2.4	2.56	ms
Wait number of integration steps (Note 1)			1		256	steps

PARAMETER†		TEST CONDITIONS	MIN	TYP	MAX	UNIT
f _(SCL)	Clock frequency (I ² C only)		0		400	kHz
t _(BUF)	Bus free time between start and stop condition		1.3			μs
t _(HDSTA)	Hold time after (repeated) start condition. After this period, the first clock is generated.		0.6			μs
t _(SUSTA)	Repeated start condition setup time		0.6			μs
t _(SUSTO)	Stop condition setup time		0.6			μs
t _(HDDAT)	Data hold time		0			μs
t _(SUDAT)	Data setup time		100			ns
t _(LOW)	SCL clock low period		1.3			μs
t _(HIGH)	SCL clock high period		0.6			μs
t _F	Clock/data fall time				300	ns
t _R	Clock/data rise time				300	ns
C _i	Input pin capacitance				10	pF

The diagram illustrates the timing relationships for the I2C protocol. It shows two signals: SCL (Serial Clock Line) and SDA (Serial Data Line). The SCL signal is a periodic clock, and the SDA signal carries the data. The diagram is divided into four sections: Stop Condition, Start Condition, and two data transfer cycles. Key timing parameters are labeled:

- $t_{(LOW)}$: SCL low pulse width.
- $t_{(R)}$: SCL rise time.
- $t_{(F)}$: SCL fall time.
- $t_{(HDSTA)}$: SDA hold time after start condition.
- $t_{(HIGH)}$: SCL high pulse width.
- $t_{(SUSTA)}$: SDA setup time before start condition.
- $t_{(HDDAT)}$: SDA hold time after data transfer.
- $t_{(SUDAT)}$: SDA setup time before data transfer.
- $t_{(SUSTO)}$: SDA setup time before stop condition.
- $t_{(BUF)}$: SDA bus free time after stop condition.
- V_{IH} and V_{IL} : Input high and low voltage levels for SDA.

The diagram also indicates the state of the SDA signal during the Stop Condition (P) and Start Condition (S) periods.

5

TYPICAL CHARACTERISTICS

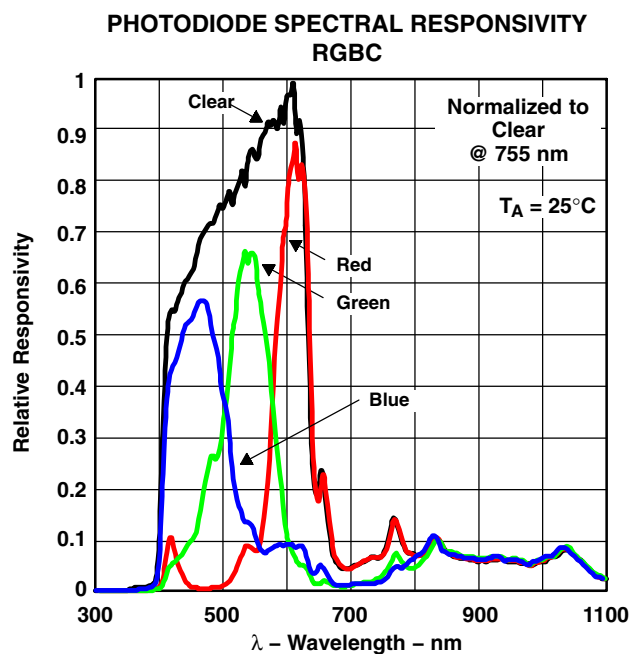


Figure 2

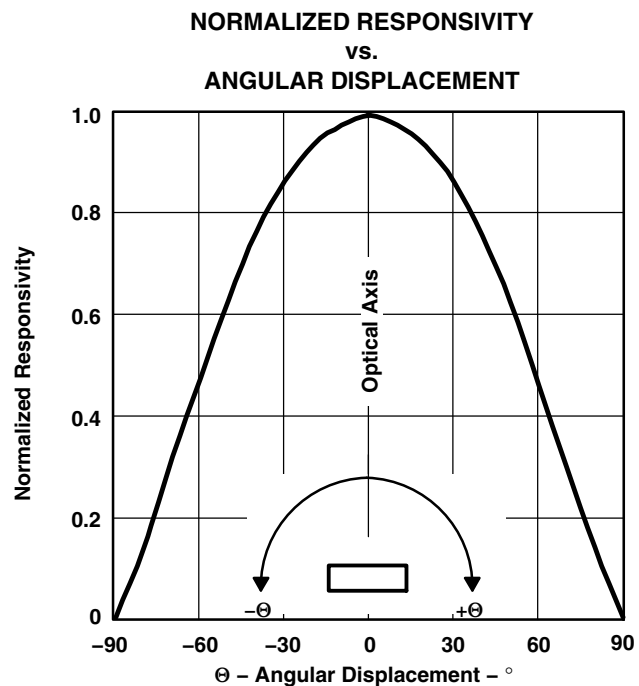


Figure 3

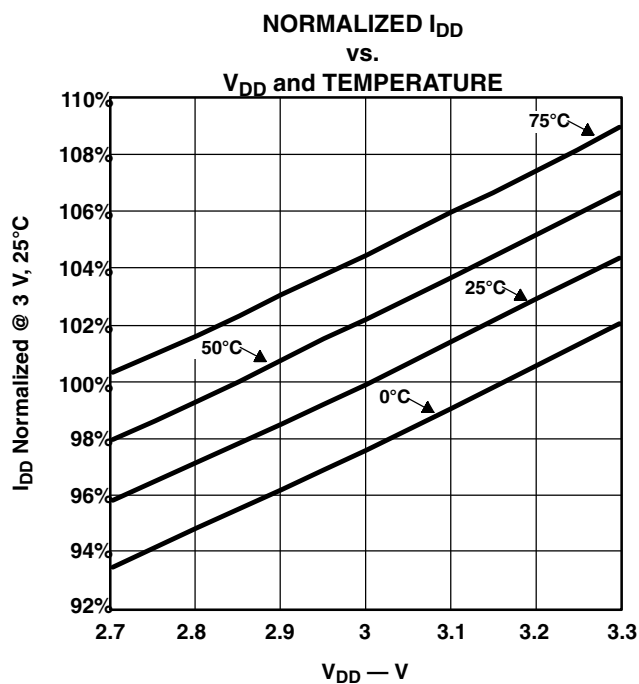


Figure 4

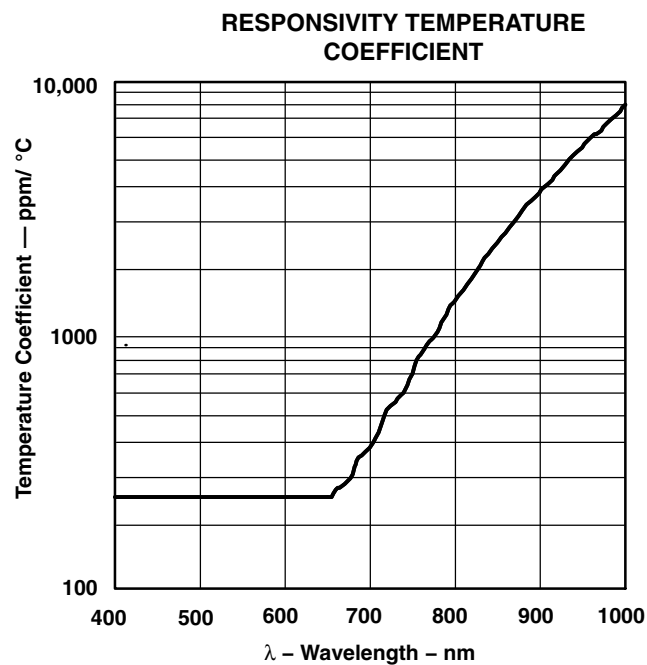


Figure 5

PRINCIPLES OF OPERATION

System States

An internal state machine provides system control of the RGBC and power management features of the device. At power up, an internal power-on-reset initializes the device and puts it in a low-power Sleep state.

When a start condition is detected on the I²C bus, the device transitions to the Idle state where it checks the Enable Register (0x00) PON bit. If PON is disabled, the device will return to the Sleep state to save power. Otherwise, the device will remain in the Idle state until the RGBC function is enabled (AEN). Once enabled, the device will execute the Wait and RGBC states in sequence as indicated in Figure 5. Upon completion and return to Idle, the device will automatically begin a new Wait-RGBC cycle as long as PON and AEN remain enabled.

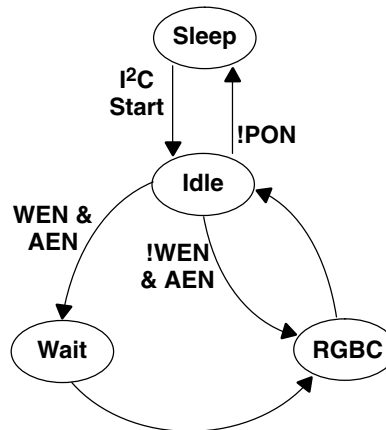


Figure 6. Simplified State Diagram

TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

TAOS135 – AUGUST 2012

RGBC Operation

The RGBC engine contains RGBC gain control (AGAIN) and four integrating analog-to-digital converters (ADC) for the RGBC photodiodes. The RGBC integration time (ATIME) impacts both the resolution and the sensitivity of the RGBC reading. Integration of all four channels occurs simultaneously and upon completion of the conversion cycle, the results are transferred to the color data registers. This data is also referred to as channel *count*.

The transfers are double-buffered to ensure that invalid data is not read during the transfer. After the transfer, the device automatically moves to the next state in accordance with the configured state machine.

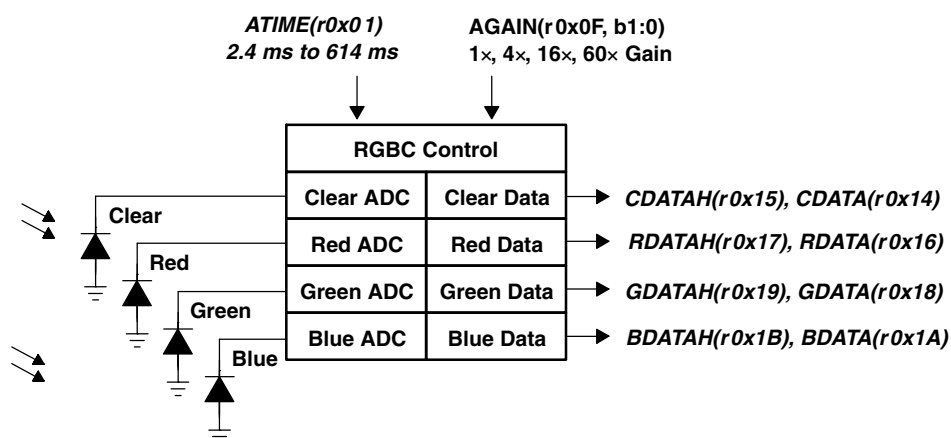


Figure 7. RGBC Operation

NOTE: In this document, the nomenclature uses the bit field name in italics followed by the register address and bit number to allow the user to easily identify the register and bit that controls the function. For example, the power on (PON) is in register 0x00, bit 0. This is represented as *PON (r0x00:b0)*.

The registers for programming the integration and wait times are a 2's complement values. The actual time can be calculated as follows:

$$ATIME = 256 - \text{Integration Time} / 2.4 \text{ ms}$$

Inversely, the time can be calculated from the register value as follows:

$$\text{Integration Time} = 2.4 \text{ ms} \times (256 - ATIME)$$

For example, if a 100-ms integration time is needed, the device needs to be programmed to:

$$256 - (100 / 2.4) = 256 - 42 = 214 = 0xD6$$

Conversely, the programmed value of 0xC0 would correspond to:

$$(256 - 0xC0) \times 2.4 = 64 \times 2.4 = 154 \text{ ms.}$$

Interrupts

The interrupt feature simplifies and improves system efficiency by eliminating the need to poll the sensor for light intensity values outside of a user-defined range. While the interrupt function is always enabled and its status is available in the status register (0x13), the output of the interrupt state can be enabled using the RGBC interrupt enable (AIEN) field in the enable register (0x00).

Two 16-bit interrupt threshold registers allow the user to set limits below and above a desired light level. An interrupt can be generated when the Clear data (CDATA) is less than the Clear interrupt low threshold (AILTx) or is greater than the Clear interrupt high threshold (AIHTx).

It is important to note that the thresholds are evaluated in sequence, first the low threshold, then the high threshold. As a result, if the low threshold is set above the high threshold, the high threshold is ignored and only the low threshold is evaluated.

To further control when an interrupt occurs, the device provides a persistence filter. The persistence filter allows the user to specify the number of consecutive out-of-range Clear occurrences before an interrupt is generated. The persistence filter register (0x0C) allows the user to set the Clear persistence filter (APERS) value. See the persistence filter register for details on the persistence filter value. Once the persistence filter generates an interrupt, it will continue until a special function interrupt clear command is received (see command register).

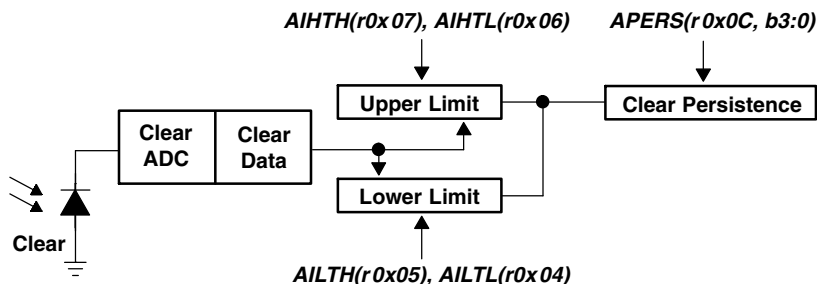


Figure 8. Programmable Interrupt

TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

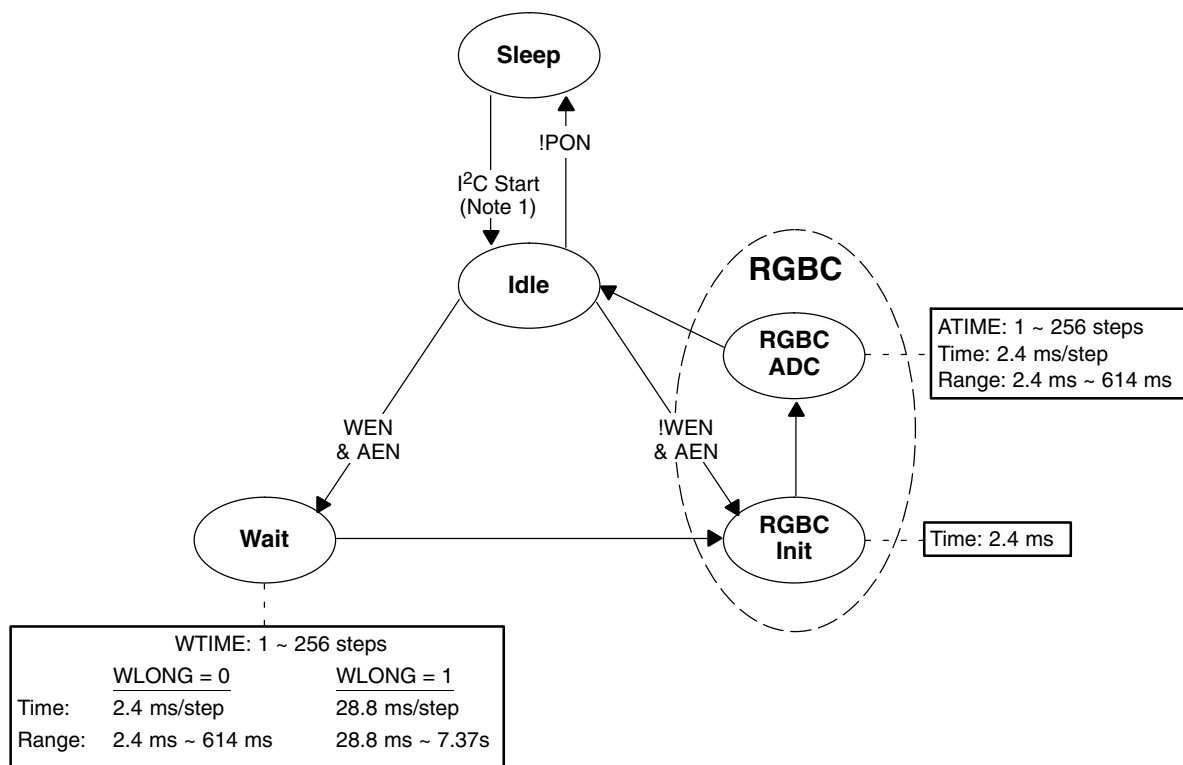
TAOS135 – AUGUST 2012

System Timing

The system state machine shown in Figure 5 provides an overview of the states and state transitions that provide system control of the device. This section highlights the programmable features, which affect the state machine cycle time, and provides details to determine system level timing.

When the power management feature is enabled (WEN), the state machine will transition to the Wait state. The wait time is determined by WLONG, which extends normal operation by 12X when asserted, and WTIME. The formula to determine the wait time is given in the box associated with the Wait state in Figure 9.

When the RGBC feature is enabled (AEN), the state machine will transition through the RGBC Init and RGBC ADC states. The RGBC Init state takes 2.4 ms, while the RGBC ADC time is dependent on the integration time (ATIME). The formula to determine RGBC ADC time is given in the associated box in Figure 9. If an interrupt is generated as a result of the RGBC cycle, it will be asserted at the end of the RGBC ADC.



- Notes: 1. There is a 2.4 ms warm-up delay if PON is enabled. If PON is not enabled, the device will return to the Sleep state as shown.
2. PON, WEN, and AEN are fields in the Enable register (0x00).

Figure 9. Detailed State Diagram

Power Management

Power consumption can be managed with the Wait state, because the Wait state typically consumes only 65 μA of I_{DD} current. An example of the power management feature is given below. With the assumptions provided in the example, average I_{DD} is estimated to be 152 μA .

Table 1. Power Management

SYSTEM STATE MACHINE STATE	PROGRAMMABLE PARAMETER	PROGRAMMED VALUE	DURATION	TYPICAL CURRENT
Wait	WTIME	0xEE	43.2 ms	0.065 mA
	WLONG	0		
RGBC Init			2.40 ms	0.235 mA
RGBC ADC	ATIME	0xEE	43.2 ms	0.235 mA

$$\text{Average } I_{DD} \text{ Current} = ((43.2 \times 0.065) + (43.2 \times 0.235) + (2.40 \times 0.235)) / 89 \approx 152 \mu\text{A}$$

Keeping with the same programmed values as the example, Table 2 shows how the average I_{DD} current is affected by the Wait state time, which is determined by WEN, WTIME, and WLONG. Note that the worst-case current occurs when the Wait state is not enabled.

Table 2. Average I_{DD} Current

WEN	WTIME	WLONG	WAIT STATE	AVERAGE I_{DD} CURRENT
0	n/a	n/a	0 ms	291 μA
1	0xFF	0	2.40 ms	280 μA
1	0xEE	0	43.2 ms	152 μA
1	0x00	0	614 ms	82 μA
1	0x00	1	7.37 s	67 μA

TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

TAOS135 – AUGUST 2012

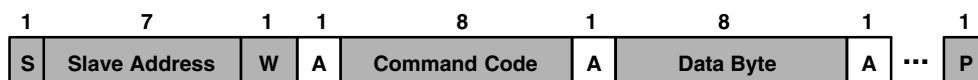
I²C Protocol

Interface and control are accomplished through an I²C serial compatible interface (standard or fast mode) to a set of registers that provide access to device control functions and output data. The devices support the 7-bit I²C addressing protocol.

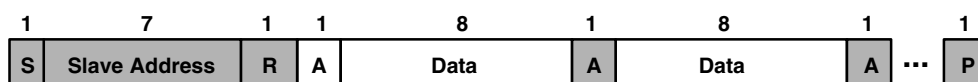
The I²C standard provides for three types of bus transaction: read, write, and a combined protocol (Figure 10). During a write operation, the first byte written is a command byte followed by data. In a combined protocol, the first byte written is the command byte followed by reading a series of bytes. If a read command is issued, the register address from the previous command will be used for data access. Likewise, if the MSB of the command is not set, the device will write a series of bytes at the address stored in the last valid command with a register address. The command byte contains either control information or a 5-bit register address. The control commands can also be used to clear interrupts.

The I²C bus protocol was developed by Philips (now NXP). For a complete description of the I²C protocol, please review the NXP I²C design specification at <http://www.i2c-bus.org/references/>.

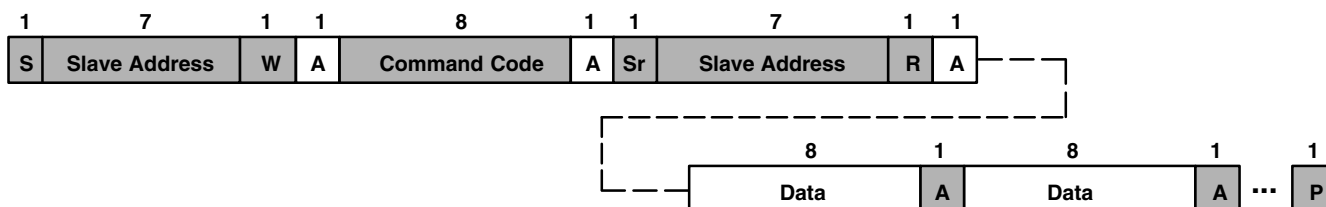
- A** Acknowledge (0)
- N** Not Acknowledged (1)
- P** Stop Condition
- R** Read (1)
- S** Start Condition
- Sr** Repeated Start Condition
- W** Write (0)
- ...** Continuation of protocol
-  Master-to-Slave
-  Slave-to-Master



I²C Write Protocol



I²C Read Protocol



I²C Read Protocol — Combined Format

Figure 10. I²C Protocols

Register Set

The TCS3472 is controlled and monitored by data registers and a command register accessed through the serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in Table 3.

Table 3. Register Address

ADDRESS	REGISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
--	COMMAND	W	Specifies register address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	ATIME	R/W	RGBC time	0xFF
0x03	WTIME	R/W	Wait time	0xFF
0x04	AILTL	R/W	Clear interrupt low threshold low byte	0x00
0x05	AILTH	R/W	Clear interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	Clear interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	Clear interrupt high threshold high byte	0x00
0x0C	PERS	R/W	Interrupt persistence filter	0x00
0x0D	CONFIG	R/W	Configuration	0x00
0x0F	CONTROL	R/W	Control	0x00
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	CDATAL	R	Clear data low byte	0x00
0x15	CDATAH	R	Clear data high byte	0x00
0x16	RDATAL	R	Red data low byte	0x00
0x17	RDATAH	R	Red data high byte	0x00
0x18	GDATAL	R	Green data low byte	0x00
0x19	GDATAH	R	Green data high byte	0x00
0x1A	BDATAL	R	Blue data low byte	0x00
0x1B	BDATAH	R	Blue data high byte	0x00

The mechanics of accessing a specific register depends on the specific protocol used. See the section on I²C protocols on the previous pages. In general, the COMMAND register is written first to specify the specific control-status-data register for subsequent read/write operations.

TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

TAOS135 – AUGUST 2012

Command Register

The command register specifies the address of the target register for future write and read operations.

Table 4. Command Register

	7	6	5	4	3	2	1	0	
COMMAND	CMD	TYPE		ADDR/SF					--

FIELD	BITS	DESCRIPTION	
CMD	7	Select Command Register. Must write as 1 when addressing COMMAND register.	
TYPE	6:5	Selects type of transaction to follow in subsequent data transfers:	
		FIELD VALUE	INTEGRATION TIME
		00	Repeated byte protocol transaction
		01	Auto-increment protocol transaction
		10	Reserved — Do not use
		11	Special function — See description below
		Byte protocol will repeatedly read the same register with each data access. Block protocol will provide auto-increment function to read successive bytes.	
ADDR/SF	4:0	Address field/special function field. Depending on the transaction type, see above, this field either specifies a special function command or selects the specific control-status-data register for subsequent read and write transactions. The field values listed below only apply to special function commands:	
		FIELD VALUE	READ VALUE
		00110	Clear channel interrupt clear
		other	Reserved — Do not write
		The Clear channel interrupt clear special function clears any pending interrupt and is self-clearing.	

Enable Register (0x00)

The Enable register is used primarily to power the TCS3472 device on and off, and enable functions and interrupts as shown in Table 5.

Table 5. Enable Register

	7	6	5	4	3	2	1	0	
ENABLE	Reserved			AIEN	WEN	Reserved	AEN	PON	Address 0x00
FIELD	BITS	DESCRIPTION							
Reserved	7:5	Reserved. Write as 0.							
AIEN	4	RGBC interrupt enable. When asserted, permits RGBC interrupts to be generated.							
WEN	3	Wait enable. This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.							
Reserved	2	Reserved. Write as 0.							
AEN	1	RGBC enable. This bit activates the two-channel ADC. Writing a 1 activates the RGBC. Writing a 0 disables the RGBC.							
PON ^{1, 2}	0	Power ON. This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator.							

- NOTES: 1. See Power Management section for more information.
2. A minimum interval of 2.4 ms must pass after PON is asserted before an RGBC can be initiated.

TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

TAOS135 – AUGUST 2012

RGBC Timing Register (0x01)

The RGBC timing register controls the internal integration time of the RGBC clear and IR channel ADCs in 2.4-ms increments. Max RGBC Count = $(256 - \text{ATIME}) \times 1024$ up to a maximum of 65535.

Table 6. RGBC Timing Register

FIELD	BITS	DESCRIPTION			
		VALUE	INTEG_CYCLES	TIME	MAX COUNT
ATIME	7:0	0xFF	1	2.4 ms	1024
		0xF6	10	24 ms	10240
		0xD5	42	101 ms	43008
		0xC0	64	154 ms	65535
		0x00	256	700 ms	65535

Wait Time Register (0x03)

Wait time is set 2.4 ms increments unless the WLONG bit is asserted, in which case the wait times are 12× longer. WTIME is programmed as a 2's complement number.

Table 7. Wait Time Register

FIELD	BITS	DESCRIPTION			
		REGISTER VALUE	WAIT TIME	TIME (WLONG = 0)	TIME (WLONG = 1)
WTIME	7:0	0xFF	1	2.4 ms	0.029 sec
		0xAB	85	204 ms	2.45 sec
		0x00	256	614 ms	7.4 sec

RGBC Interrupt Threshold Registers (0x04 – 0x07)

The RGBC interrupt threshold registers provides the values to be used as the high and low trigger points for the comparison function for interrupt generation. If the value generated by the clear channel crosses below the lower threshold specified, or above the higher threshold, an interrupt is asserted on the interrupt pin.

Table 8. RGBC Interrupt Threshold Registers

REGISTER	ADDRESS	BITS	DESCRIPTION
AILTL	0x04	7:0	RGBC clear channel low threshold lower byte
AILTH	0x05	7:0	RGBC clear channel low threshold upper byte
AIHTL	0x06	7:0	RGBC clear channel high threshold lower byte
AIHTH	0x07	7:0	RGBC clear channel high threshold upper byte

Persistence Register (0x0C)

The persistence register controls the filtering interrupt capabilities of the device. Configurable filtering is provided to allow interrupts to be generated after each integration cycle or if the integration has produced a result that is outside of the values specified by the threshold register for some specified amount of time.

Table 9. Persistence Register

	7	6	5	4	3	2	1	0	
PERS	Reserved				APERS				Address 0x0C
FIELD	BITS	DESCRIPTION							
PPERS	7:4	Reserved							
APERS	3:0	Interrupt persistence. Controls rate of interrupt to the host processor.							
		FIELD VALUE	MEANING	INTERRUPT PERSISTENCE FUNCTION					
		0000	Every	Every RGBC cycle generates an interrupt					
		0001	1	1 clear channel value outside of threshold range					
		0010	2	2 clear channel consecutive values out of range					
		0011	3	3 clear channel consecutive values out of range					
		0100	5	5 clear channel consecutive values out of range					
		0101	10	10 clear channel consecutive values out of range					
		0110	15	15 clear channel consecutive values out of range					
		0111	20	20 clear channel consecutive values out of range					
		1000	25	25 clear channel consecutive values out of range					
		1001	30	30 clear channel consecutive values out of range					
		1010	35	35 clear channel consecutive values out of range					
		1011	40	40 clear channel consecutive values out of range					
		1100	45	45 clear channel consecutive values out of range					
		1101	50	50 clear channel consecutive values out of range					
		1110	55	55 clear channel consecutive values out of range					
		1111	60	60 clear channel consecutive values out of range					

TCS3472

COLOR LIGHT-TO-DIGITAL CONVERTER

with IR FILTER

TAOS135 – AUGUST 2012

Configuration Register (0x0D)

The configuration register sets the wait long time.

Table 10. Configuration Register

	7	6	5	4	3	2	1	0	
CONFIG	Reserved						WLONG	Reserved	Address 0x0D

FIELD	BITS	DESCRIPTION
Reserved	7:2	Reserved. Write as 0.
WLONG	1	Wait Long. When asserted, the wait cycles are increased by a factor 12× from that programmed in the WTIME register.
Reserved	0	Reserved. Write as 0.

Control Register (0x0F)

The Control register provides eight bits of miscellaneous control to the analog block. These bits typically control functions such as gain settings and/or diode selection.

Table 11. Control Register

	7	6	5	4	3	2	1	0	
CONTROL	Reserved						AGAIN		Address 0x0F

FIELD	BITS	DESCRIPTION
Reserved	7:2	Reserved. Write bits as 0
AGAIN	1:0	RGBC Gain Control.
		FIELD VALUE RGBC GAIN VALUE
		00 1× gain
		01 4× gain
		10 16× gain
		11 60× gain

ID Register (0x12)

The ID Register provides the value for the part number. The ID register is a read-only register.

Table 12. ID Register

	7	6	5	4	3	2	1	0	
ID	ID								Address 0x12

FIELD	BITS	DESCRIPTION
ID	7:0	Part number identification
		0x44 = TCS34721 and TCS34725 0x4D = TCS34723 and TCS34727



Status Register (0x13)

The Status Register provides the internal status of the device. This register is read only.

Table 13. Status Register

	7	6	5	4	3	2	1	0	
STATUS	Reserved			AINT	Reserved			AVALID	Address 0x13

FIELD	BIT	DESCRIPTION
Reserved	7:5	Reserved.
AINT	4	RGBC clear channel Interrupt.
Reserved	3:1	Reserved.
AVALID	0	RGBC Valid. Indicates that the RGBC channels have completed an integration cycle.

RGBC Channel Data Registers (0x14 – 0x1B)

Clear, red, green, and blue data is stored as 16-bit values. To ensure the data is read correctly, a two-byte read I²C transaction should be used with a read word protocol bit set in the command register. With this operation, when the lower byte register is read, the upper eight bits are stored into a shadow register, which is read by a subsequent read to the upper byte. The upper register will read the correct value even if additional ADC integration cycles end between the reading of the lower and upper registers.

Table 14. ADC Channel Data Registers

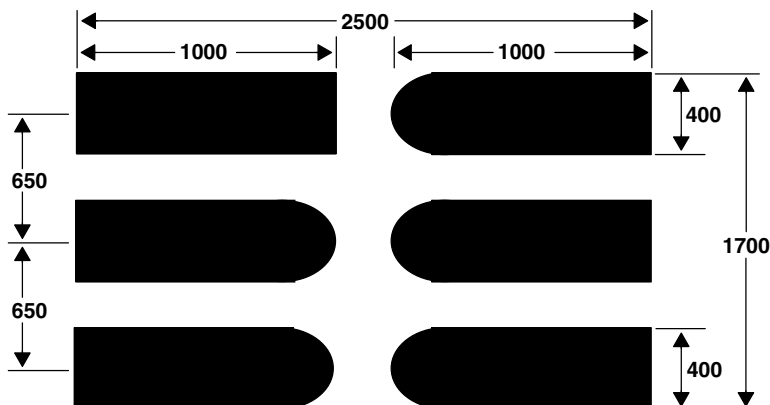
REGISTER	ADDRESS	BITS	DESCRIPTION
CDATA	0x14	7:0	Clear data low byte
CDATAH	0x15	7:0	Clear data high byte
RDATA	0x16	7:0	Red data low byte
RDATAH	0x17	7:0	Red data high byte
GDATA	0x18	7:0	Green data low byte
GDATAH	0x19	7:0	Green data high byte
BDATA	0x1A	7:0	Blue data low byte
BDATAH	0x1B	7:0	Blue data high byte

APPLICATION INFORMATION: HARDWARE

PCB Pad Layout

Suggested PCB pad layout guidelines for the Dual Flat No-Lead (FN) surface mount package are shown in Figure 11.

Note: Pads can be extended further if hand soldering is needed.



NOTES: A. All linear dimensions are in micrometers.
B. This drawing is subject to change without notice.

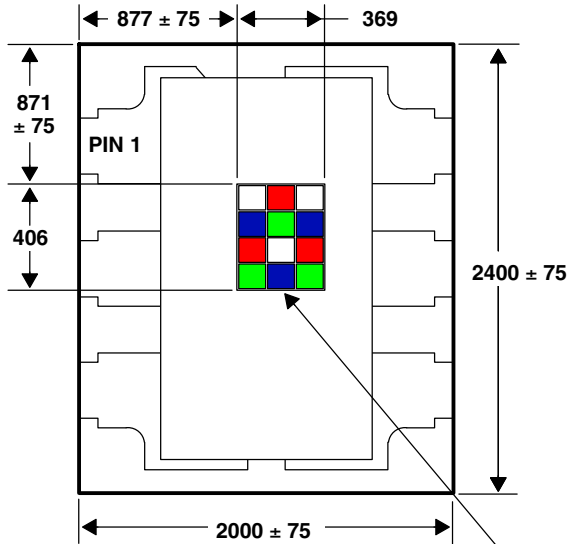
Figure 11. Suggested FN Package PCB Layout

PACKAGE INFORMATION

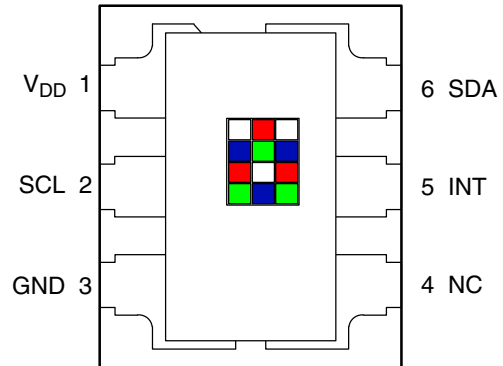
PACKAGE FN

Dual Flat No-Lead

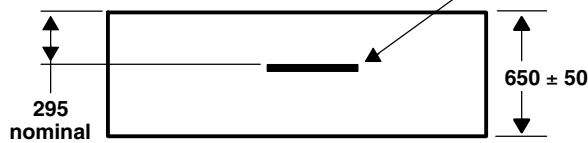
TOP VIEW



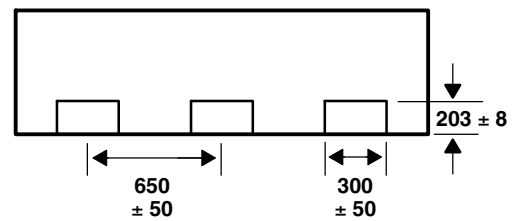
PIN OUT TOP VIEW



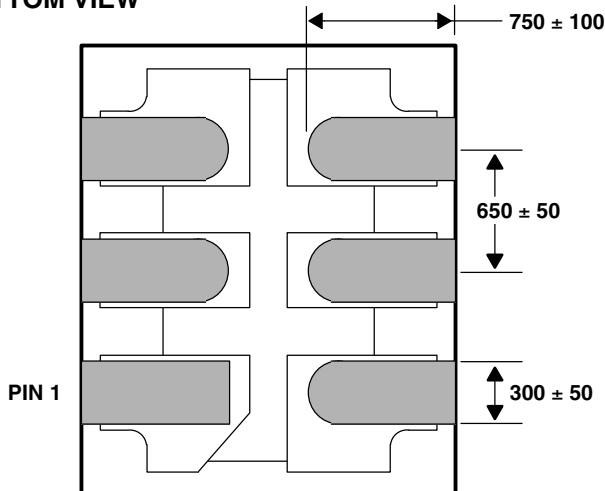
END VIEW



SIDE VIEW



BOTTOM VIEW



Lead Free

- NOTES: A. All linear dimensions are in micrometers. Dimension tolerance is $\pm 20 \mu\text{m}$ unless otherwise noted.
 B. The die is centered within the package within a tolerance of ± 3 mils.
 C. Package top surface is molded with an electrically nonconductive clear plastic compound having an index of refraction of 1.55.
 D. Contact finish is copper alloy A194 with pre-plated NiPdAu lead finish.
 E. This package contains no lead (Pb).
 F. This drawing is subject to change without notice.

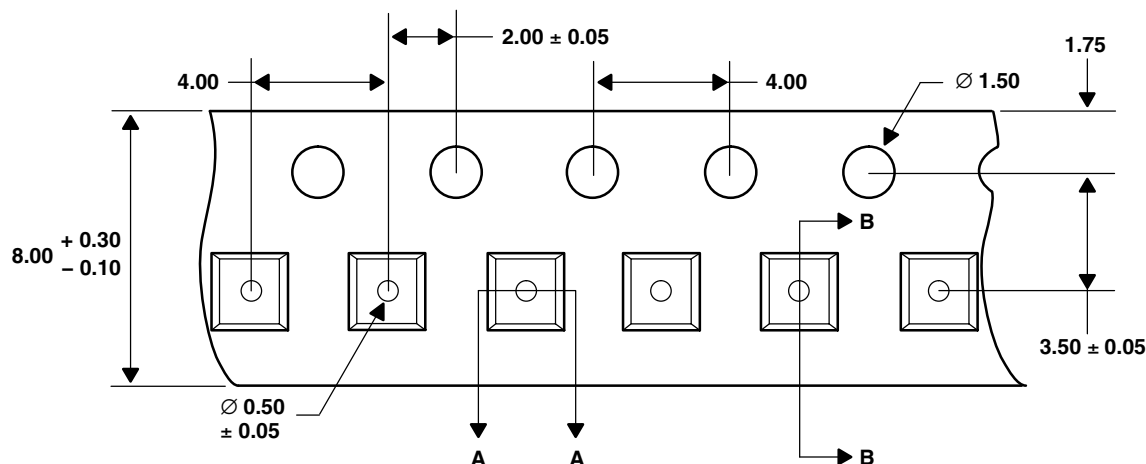
Figure 12. Package FN — Dual Flat No-Lead Packaging Configuration

TCS3472
COLOR LIGHT-TO-DIGITAL CONVERTER
with IR FILTER

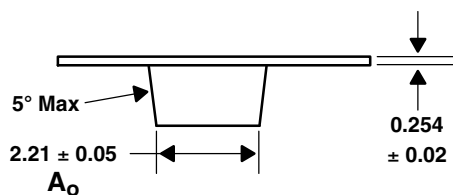
TAOS135 – AUGUST 2012

CARRIER TAPE AND REEL INFORMATION

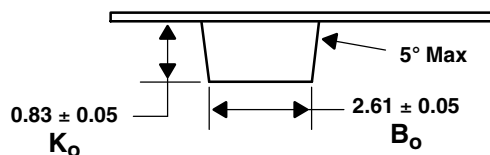
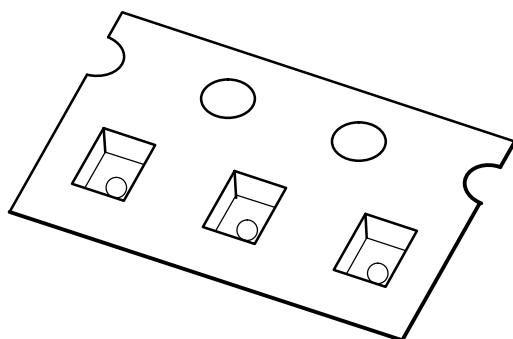
TOP VIEW



DETAIL A



DETAIL B



- NOTES: A. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
 B. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
 C. Symbols on drawing A_o , B_o , and K_o are defined in ANSI EIA Standard 481-B 2001.
 D. Each reel is 178 millimeters in diameter and contains 3500 parts.
 E. TAOS packaging tape and reel conform to the requirements of EIA Standard 481-B.
 F. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
 G. This drawing is subject to change without notice.

Figure 13. Package FN Carrier Tape



SOLDERING INFORMATION

The FN package has been tested and has demonstrated an ability to be reflow soldered to a PCB substrate. The process, equipment, and materials used in these test are detailed below.

The solder reflow profile describes the expected maximum heat exposure of components during the solder reflow process of product on a PCB. Temperature is measured on top of component. The components should be limited to a maximum of three passes through this solder reflow profile.

Table 15. Solder Reflow Profile

PARAMETER	REFERENCE	DEVICE
Average temperature gradient in preheating		2.5°C/sec
Soak time	t_{soak}	2 to 3 minutes
Time above 217°C (T1)	t_1	Max 60 sec
Time above 230°C (T2)	t_2	Max 50 sec
Time above $T_{\text{peak}} - 10^\circ\text{C}$ (T3)	t_3	Max 10 sec
Peak temperature in reflow	T_{peak}	260°C
Temperature gradient in cooling		Max -5°C/sec

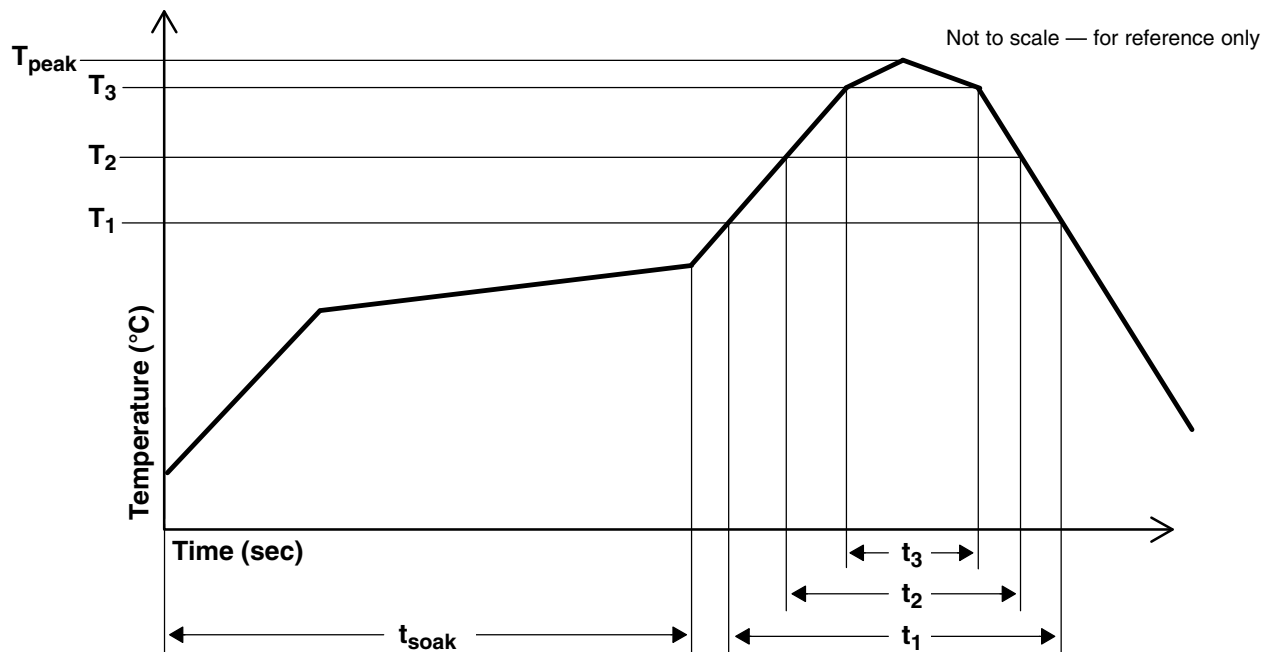


Figure 14. Solder Reflow Profile Graph

STORAGE INFORMATION

Moisture Sensitivity

Optical characteristics of the device can be adversely affected during the soldering process by the release and vaporization of moisture that has been previously absorbed into the package. To ensure the package contains the smallest amount of absorbed moisture possible, each device is dry-baked prior to being packed for shipping. Devices are packed in a sealed aluminized envelope called a moisture barrier bag with silica gel to protect them from ambient moisture during shipping, handling, and storage before use.

The Moisture Barrier Bags should be stored under the following conditions:

Temperature Range	< 40°C
Relative Humidity	< 90%
Total Time	No longer than 12 months from the date code on the aluminized envelope if unopened.

Rebaking of the reel will be required if the devices have been stored unopened for more than 12 months and the Humidity Indicator Card shows the parts to be out of the allowable moisture region.

Opened reels should be used within 168 hours if exposed to the following conditions:

Temperature Range	< 30°C
Relative Humidity	< 60%

If rebaking is required, it should be done at 50°C for 12 hours.

The FN package has been assigned a moisture sensitivity level of MSL 3.



PRODUCTION DATA — information in this document is current at publication date. Products conform to specifications in accordance with the terms of Texas Advanced Optoelectronic Solutions, Inc. standard warranty. Production processing does not necessarily include testing of all parameters.

LEAD-FREE (Pb-FREE) and GREEN STATEMENT

Pb-Free (RoHS) TAOS' terms *Lead-Free* or *Pb-Free* mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TAOS Pb-Free products are suitable for use in specified lead-free processes.

Green (RoHS & no Sb/Br) TAOS defines *Green* to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

Important Information and Disclaimer The information provided in this statement represents TAOS' knowledge and belief as of the date that it is provided. TAOS bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TAOS has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TAOS and TAOS suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

NOTICE

Texas Advanced Optoelectronic Solutions, Inc. (TAOS) reserves the right to make changes to the products contained in this document to improve performance or for any other purpose, or to discontinue them without notice. Customers are advised to contact TAOS to obtain the latest product information before placing orders or designing TAOS products into systems.

TAOS assumes no responsibility for the use of any products or circuits described in this document or customer product design, conveys no license, either expressed or implied, under any patent or other right, and makes no representation that the circuits are free of patent infringement. TAOS further makes no claim as to the suitability of its products for any particular purpose, nor does TAOS assume any liability arising out of the use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS, INC. PRODUCTS ARE NOT DESIGNED OR INTENDED FOR USE IN CRITICAL APPLICATIONS IN WHICH THE FAILURE OR MALFUNCTION OF THE TAOS PRODUCT MAY RESULT IN PERSONAL INJURY OR DEATH. USE OF TAOS PRODUCTS IN LIFE SUPPORT SYSTEMS IS EXPRESSLY UNAUTHORIZED AND ANY SUCH USE BY A CUSTOMER IS COMPLETELY AT THE CUSTOMER'S RISK.

LUMENOLOGY, TAOS, the TAOS logo, and Texas Advanced Optoelectronic Solutions are registered trademarks of Texas Advanced Optoelectronic Solutions Incorporated.

