A

Project Report

On

SELF-CHECKOUT SYSTEM USING ARDUINO AND RFID

Submitted in partial fulfillment of the requirements for the award of

Diploma in Electronics and Communication Engineering

SUBMITTED

BY

P. JAYANTH **22248-EC-021**

Under the Esteemed Guidance of

Mrs. K. SRAVANTHI (ASST. PROFESSOR)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TEEGALA KRISHNA REDDY ENGINEERING COLLEGE

(Affiliated to SBTET, 2 Shift Polytechnic College , Medbowli , Meerpet, Hyderabad-500097)

A

Project Report

On

SELF-CHECKOUT SYSTEM USING ARDUINO AND RFID

Submitted in partial fulfillment of the requirements for the award of

Diploma in Electronics and Communication Engineering

SUBMITTED

BY

P. JAYANTH **22248-EC-021**

Under the Esteemed Guidance of

Mrs. K. SRAVANTHI (ASST. PROFESSOR)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TEEGALA KRISHNA REDDY ENGINEERING COLLEGE

(Affiliated to SBTET, 2 Shift Polytechnic College, Medbowli , Meerpet, Hyderabad-500097)

CERTIFICATE

This is to certify that the project entitled **"SELF-CHECKOUT SYSTEM USING ARDUINO AND RFID"** is a Bonafide work carried out by **P. Jayanth** under esteemed guidance and supervision of **Mrs. K. SRAVANTHI (Assistant Professor)** in the partial fulfillment of the academic requirement for the award of degree of Diploma in **Electronics and Communication Engineering** submitted to the **DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING, TEEGALA KRISHNA REDDY ENGINEERING COLLEGE**, Hyderabad during the academic year 2024-2025.

INTERNAL GUIDE	HOD	PRINCIPAL
Mrs. K. SRAVANTHI	Mrs. K. SRILATHA	Dr. P. VENKATRAM REDDY

INTERNAL EXAMINER	EXTERNAL EXAMINER

DECLARATION

I hereby declare that the work embodied in this dissertation entitled “SELF-CHECKOUT USING ARDUINO AND RFID” is carried out under the guidance of Mrs. K. SRAVANTHI Assistant Professor Department of Electronics and Communication Engineering, Teegala Krishna Reddy Engineering College, Hyderabad.

Also, I declare that the matter embodied in this project have not been reproduced or copied from any source.

SUBMITTED BY

P. JAYANTH **22248-EC-021**

ACKNOWLEDGEMENT

I extend my deep sense of gratitude to **Dr. P. VENKAT RAM REDDY, Principal** Teegala Krishna Reddy Engineering College, Meerpet, for permitting me to undertake this project.

I am indebted to **Mrs. K. SRILATHA, Assistant professor & Head of the Department**, Electronics and Communication Engineering, Teegala Krishna Reddy Engineering College, Meerpet for her support and guidance throughout this project.

I am indebted to our guide **Mrs. K. SRAVANTHI, Assistant professor**, Electronics and Communication Engineering, Teegala Krishna Reddy Engineering College, Meerpet for her support and guidance throughout this project.

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance have crowned our efforts with success.

Finally, I express thanks to one and all that have helped me in successfully completing this project. Further I would like to thank my family and friends for their moral support and encouragement.

SUBMITTED BY

P. JAYANTH **22248-EC-021**

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The Self-Checkout System redefines retail by enabling customers to autonomously scan items, bag purchases, and process payments, delivering a fast, user-friendly shopping experience. This comprehensive documentation serves as an essential guide for implementing and operating the system, ensuring seamless adoption across diverse retail settings. By significantly reducing checkout times and alleviating staff workload, the system enhances operational efficiency and customer satisfaction, addressing the growing demand for convenient, technology-driven shopping solutions. Its intuitive interface accommodates users of all skill levels, making it a cornerstone for modern retail environments seeking to remain competitive in an evolving market. The Self-Checkout System also facilitates real-time data collection, empowering retailers with valuable insights into inventory, sales trends, and customer behaviour to optimize business strategies. This guide provides clear, actionable instructions for developers, administrators, and store managers to deploy, configure, and maintain the system effectively. By emphasizing best practices and troubleshooting strategies, it ensures reliable performance and scalability, even in high-traffic scenarios. The Self-Checkout System is not just a technological advancement but a transformative tool that elevates the shopping experience, streamlines store operations, and drives retail innovation. This documentation equips stakeholders with the knowledge to harness its full potential, fostering long-term success and adaptability in the dynamic retail landscape.

CHAPTER- 1

INTRODUCTION

1.1 Introduction

In the rapidly changing world of today, technology continues to redefine mundane interactions, especially where shopping is involved. Typical retailing processes, especially in supermarkets, usually involve lengthy checkout queues, manual scanning of products purchased, and lengthy payment processes. This results in lost customer satisfaction and increased inefficiency, especially when demand is high. In a bid to turn this problem around, the idea of a **Self-Checkout System using Arduino and RFID** was developed to automate and streamline the billing process at the shopping cart.

This project integrates RFID technology with an Arduino Nano microcontroller to develop an interactive, real-time billing system. Each product in the store is labelled with an RFID card containing essential information like the product name and price. When a customer places a product in the shopping trolley, the RFID reader scans the tag and shows the item name and price on a 16x2 LCD display with I2C interface. A buzzer built into the system gives immediate audio feedback upon successful scans. The system updates a running total, enabling users to keep track of their expenditure while shopping.

In order to improve user agency, two activation buttons are incorporated: one designated for the confirmation of a selected item and another aimed at rescinding the subsequent scanned item. The overall configuration is both space-efficient and energy-conserving, rendering it appropriate for practical application in real-world scenarios. Ultimately, this intelligent trolley design aspires to decrease transaction duration, eradicate manual billing inaccuracies, and enhance consumer satisfaction via automation. It provides an insight into the future of retail, wherein convenience and efficiency are the primary catalysts of the shopping experience.

1.2 Objective

The primary objective of this project is to create a Self-checkout system that facilitates easy billing through RFID technology coupled with an Arduino Nano. The system aims to reduce long queues during checkout by allowing customers to directly scan the items through RFID tags. It has an LCD display to show the names of the items, prices, and the running bill. It also has push buttons to accept or reject the scanned items and uses a buzzer as a source of sound feedback. Overall, the project is intended to increase shopping efficiency, improve customer convenience, and minimize human errors in billing.

1.3 Literature Survey

The concept of automating shopping processes has gained significant attention in recent years, driven by advancements in embedded systems, wireless communication, and identification technologies. This literature survey explores existing research and implementations relevant to the development of a Self-checkout System using Arduino Nano and RFID, highlighting key contributions, technologies, and gaps addressed by the current project.

One of the foundational works in this domain is by Raju et al. (2015), who proposed an RFID-based smart shopping system to reduce manual billing efforts. Their system utilized RFID tags on products and an RFID reader interfaced with a microcontroller to identify items and calculate totals. The authors emphasized RFID’s reliability over barcodes due to its non-line-of-sight scanning capability, a feature leveraged in the present project with the Arduino Nano. However, their implementation relied on a larger microcontroller setup, contrasting with the compact and cost-effective Nano used here.

Similarly, Patil et al. (2017) developed a "Smart Trolley for Malls" using RFID and ZigBee modules. Their system transmitted item data wirelessly to a central billing station, enhancing scalability across large stores. While effective, the inclusion of ZigBee

increased complexity and cost, which the current project avoids by focusing on localized processing with Arduino Nano and a simple LCD display. This ensures affordability and ease of deployment in smaller retail settings.

Another notable contribution comes from Kumar and Singh (2019), who integrated IoT with RFID for a smart shopping cart. Their system uploaded real-time data to a cloud server, allowing customers to track purchases via a mobile app. While innovative, this approach requires constant internet connectivity, a limitation the present project circumvents by keeping operations offline and self-contained within the trolley. The Arduino Nano's processing power, though modest, suffices for the standalone billing logic required here.

Research by Sharma et al. (2020) explored the use of Arduino-based systems for retail automation, focusing on sensor integration. They combined RFID with weight sensors to verify item presence, addressing potential errors in tag scanning. While this enhances accuracy, it also increases hardware costs and complexity. The current project prioritizes simplicity, relying solely on RFID for identification, with the Arduino Nano efficiently handling tag data and price lookups.

Finally, commercial implementations like Amazon Go (2021) showcase advanced automation using RFID, cameras, and AI. While highly effective, such systems are resource-intensive and impractical for small-scale retailers. The proposed Self-checkout System bridges this gap, offering a low-cost, microcontroller-driven alternative tailored for accessibility.

In conclusion, while prior works demonstrate the potential of RFID in shopping automation, they often involve complex setups or high costs. This project leverages the Arduino Nano's simplicity and RFID's efficiency to create an affordable, user-friendly solution, addressing limitations in scalability, cost, and connectivity observed in the literature.

CHAPTER- 2

BLOCK DIAGRAM AND SCHEMATIC DIAGRAM

2.1 Block Diagram:

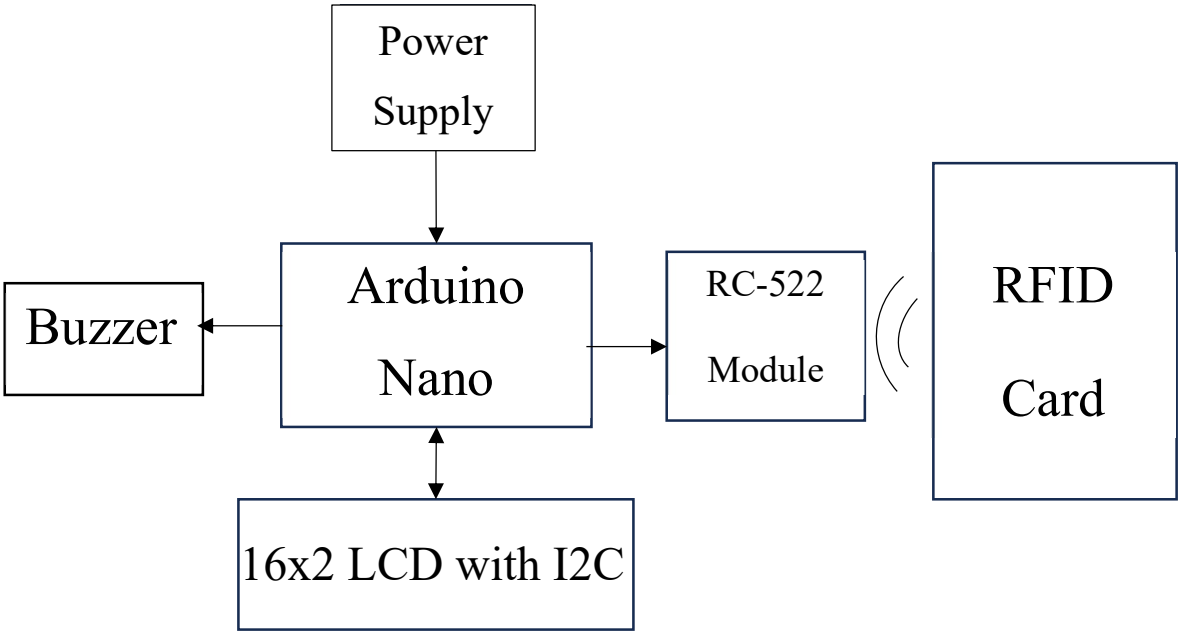

Fig.2.1 Block diagram

COMPONENTS USED

- Arduino Nano
- Buzzer
- 16x2 LCD with I2C
- MFRC522 Reader (RC522 Module)
- Power Supply
- RFID Cards

2.2 Schematic Diagram

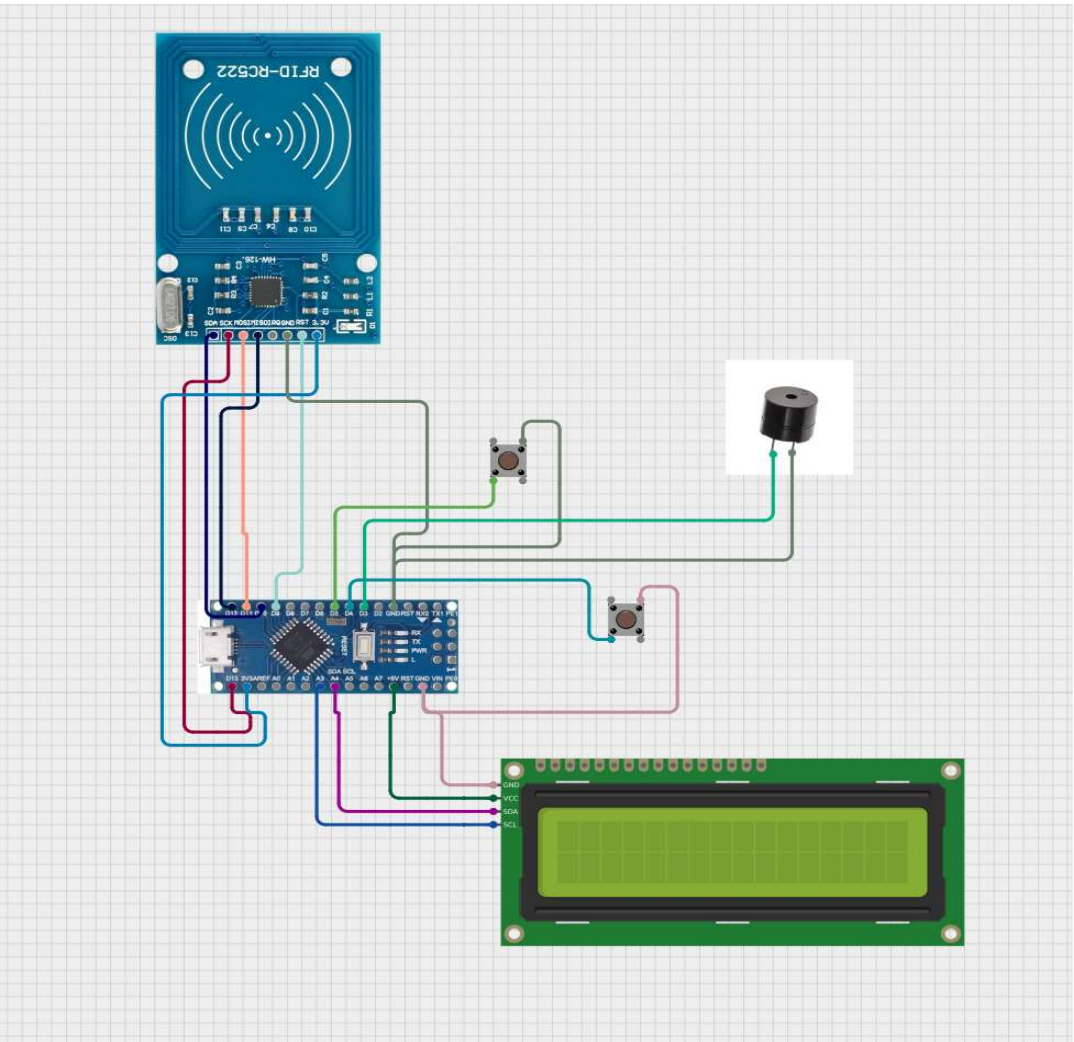

Fig.2.2 Schematic Diagram

The schematic diagram depicts a Self-checkout system using an Arduino Nano as the microcontroller. It integrates an RFID module (MFRC522) for product identification, connected via multiple pins, and an LCD display for real-time billing updates. A buzzer and push button are included for alerts and user input, with all components wired on a breadboard for prototyping.

CHAPTER- 3

HARDWARE COMPONENTS

3.1 Arduino

Arduino is an open-source electronics platform designed to make hardware and software development accessible to beginners and professionals alike. Introduced in 2005 by Massimo Banzi and colleagues in Italy, it consists of a microcontroller board (e.g., Arduino Uno, Nano) and an Integrated Development Environment (IDE) for writing and uploading code. The platform supports a variety of boards, each with different capabilities, such as the compact Arduino Nano used in projects like the Self-checkout system. Arduino boards feature digital and analog input/output pins, allowing connection to sensors, actuators, and displays. Its simplicity, affordability, and extensive community support have made it popular for prototyping in robotics, IoT, and automation. Programs, written in a simplified C/C++ syntax, are compiled and uploaded via USB. With a vast library of add-ons and tutorials, Arduino empowers users to create innovative solutions, from home automation to educational projects, fostering creativity and technical skill development across diverse fields.

3.1.1 Arduino Nano

The Arduino Nano is a compact, breadboard-friendly microcontroller board designed to cater to a wide range of embedded system projects with its small footprint. Introduced by Arduino in 2008, it is an excellent choice for applications requiring minimal space, such as the Self-Checkout System. The board's open-source hardware and software ecosystem, paired with the Arduino IDE, allows users to write and upload code, supported by a vast library of add-ons and community-driven tutorials. This accessibility has made it a favourite among hobbyists, students, and professionals for prototyping in robotics, Internet of Things (IoT) applications, and automation projects. Its design emphasizes portability and ease of use, making it a versatile tool for innovative

solutions, from educational experiments to commercial prototypes. The extensive community support further enriches its ecosystem, offering resources like forums, example projects, and libraries that simplify development. Whether used in a classroom or a lab, the Arduino Nano stands out for its balance of performance, compactness, and adaptability, driving its widespread adoption across diverse technical fields.

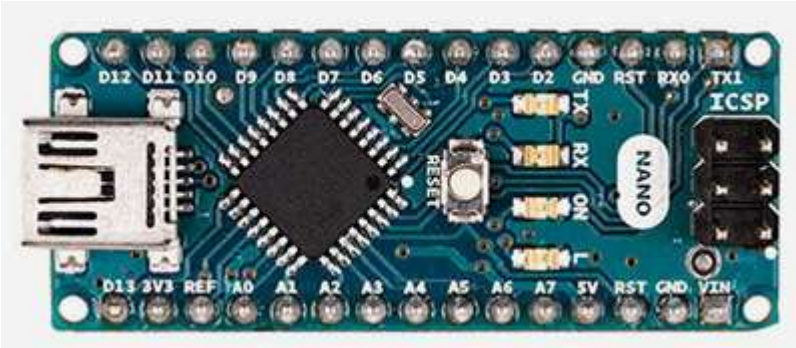

Fig.3.1 Arduino Nano

This popularity stems from its ability to bridge the gap between complex electronics and beginner-friendly design, fostering a collaborative environment where users share ideas and solutions. The Nano's adaptability extends to various industries, including education, where it serves as a hands-on learning tool, and small-scale entrepreneurship, where it powers cost-effective prototypes. Its compact nature also appeals to creators working on wearable technology or portable devices, broadening its application scope. The open-source philosophy encourages continuous improvement, with the community contributing to an ever-growing repository of resources that cater to both novice and expert users. This collaborative spirit ensures that the Arduino Nano remains relevant, evolving with the needs of its diverse user base. Its role in sparking creativity and technical innovation underscores its significance, making it a cornerstone for those looking to explore the frontiers of embedded systems with a reliable and accessible platform.

3.1.2 Why choose Arduino Nano?

The Arduino Nano stands out as the preferred choice for the Self-Checkout system project due to its unique combination of compactness, versatility, and cost-effectiveness, setting it apart from other microcontroller boards available in the market. Unlike larger

boards such as the Arduino Uno, the Nano's diminutive size (18mm x 45mm) allows for seamless integration into space-constrained designs, making it ideal for embedding within a trolley without compromising functionality. This small footprint is a significant advantage over bulkier alternatives like the Arduino Mega, which, while powerful, is overkill for a project with moderate processing needs and limited I/O requirements.

The Nano's breadboard-friendly design further enhances its appeal, enabling rapid prototyping without the need for soldering, a feature less practical with boards requiring custom shields or additional hardware.

Performance-wise, the Arduino Nano, powered by the ATmega328P with a 16 MHz clock speed and 32 KB flash memory, offers sufficient computational capability for tasks like RFID data processing and LCD updates, without the complexity of more advanced boards like the Raspberry Pi. The Pi, while robust for multitasking, demands a full operating system and higher power, making it less efficient for a standalone, low-power application. The Nano's support for multiple communication protocols (UART, I2C, SPI) ensures compatibility with essential components like RFID modules and displays, matching the needs of this project without the excess features of boards like the ESP32, which are geared toward wireless applications.

Additionally, the Nano's open-source ecosystem and extensive community support provide a wealth of tutorials, libraries, and forums, simplifying development compared to less-documented alternatives. Its low-power modes and flexible power options (via mini-USB or external 6-20V) align with the project's goal of energy efficiency, a consideration less emphasized in high-performance boards. For a budget-conscious, portable, and straightforward solution, the Arduino Nano outperforms competitors, delivering the right balance of functionality and practicality for this innovative shopping trolley design.

3.1.3 Technical Specifications of Arduino Nano:

The Arduino Nano is a compact and powerful microcontroller board based on the ATmega328P, designed for small-scale electronics projects. It features 14 digital I/O pins (6 with PWM capability), 8 analog input pins, and a 16 MHz clock speed, offering

versatility for various applications. Operating at 5V, it supports communication through UART, I2C, and SPI interfaces. With 32 KB of flash memory, 2 KB of SRAM, and 1 KB of EEPROM, it provides ample space for storing code and data. Its small size and breadboard compatibility make it an excellent choice for prototyping, embedded systems, and DIY electronics.

The below table shows the specifications of Arduino Nano:

Specification	Details
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 can be used as PWM outputs)
Analog Input Pins	8
DC Current per I/O Pin	40 mA (maximum)
DC Current for 3.3V Pin	50 mA (maximum)
Flash Memory	32 KB (ATmega328P) (of which 2 KB is used by bootloader)
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz
LED_BUILTIN	Pin 13 (on-board LED)
Dimensions	45 x 18 mm
Weight	7 g (approx.)
Communication Interfaces	UART, I2C, SPI
USB-to-Serial Chip	CH340G or FTDI (depending on the version)
Reset Button	Yes
PWM Pins	6 pins (3, 5, 6, 9, 10, 11)
I2C Pins	A4 (SDA), A5 (SCL)
SPI Pins	D10 (SS), D11 (MOSI), D12 (MISO), D13 (SCK)

Table 3.1 Specifications of Arduino

Use of Arduino Nano in this project:

In the Self-Checkout System, the Arduino Nano acts as the central controller, orchestrating all components for seamless operation. This compact microcontroller, powered by a 9V battery, interfaces with the RC522 RFID Module to read product tags via SPI, processing their unique IDs to match with a preloaded database of item names and prices. It drives the 16x2 LCD with I2C to display real-time details like scanned items and totals, ensuring shoppers stay informed. The Nano also controls the buzzer, triggering beeps (e.g., via PWM) to confirm scans or signal errors, enhancing user feedback. Push buttons connected to its digital pins allow manual inputs, such as finalizing the bill, which the Nano processes instantly. With its 14 digital I/O pins, 6 analog inputs, and 16 MHz clock, the Nano efficiently manages data flow, making the trolley smart, responsive, and user-friendly.

3.1.4 Arduino Nano Pin Reference:

The Arduino Nano, built around the ATmega328P microcontroller, offers a compact yet powerful platform for electronics projects with its 30 male I/O headers tailored for breadboard use. It features 14 digital pins (D0-D13) for versatile input and output tasks, where D0 (RX) and D1 (TX) handle UART TTL serial communication, linking the Nano to other devices for data exchange. Six of these pins—D3, D5, D6, D9, D10, and D11—support Pulse Width Modulation (PWM), enabling precise control over components like motors or LEDs by varying signal duty cycles. The Nano also includes eight analog input pins (A0-A7), each delivering 10-bit ADC resolution to measure voltages from 0 to 5V, ideal for reading sensors. Among these, A4 (SDA) and A5 (SCL) serve dual purposes as I2C communication lines for interfacing with modules like displays or sensors, while A6 and A7 are dedicated solely to analog inputs. Power management is robust, with a VIN pin accepting 6-20V unregulated input, a 5V pin providing regulated output, and a 3.3V pin supplying up to 50mA, complemented by multiple GND pins for grounding circuits. An AREF pin allows customization of the analog reference voltage, enhancing measurement precision when paired with external

sources. The Nano's SPI interface, utilizing D10 (SS), D11 (MOSI), D12 (MISO), and D13 (SCK), facilitates high-speed communication with peripherals like RFID readers, while a built-in LED tied to D13 offers a simple output for testing or status indication. Unlike larger Arduino boards, the Nano omits a DC power jack, relying on a Mini-B USB for both power and programming, streamlining its design. This pin configuration makes the Nano a flexible choice for compact projects, balancing digital, analog, and communication capabilities within a small footprint, perfectly suited for applications requiring efficiency and adaptability.

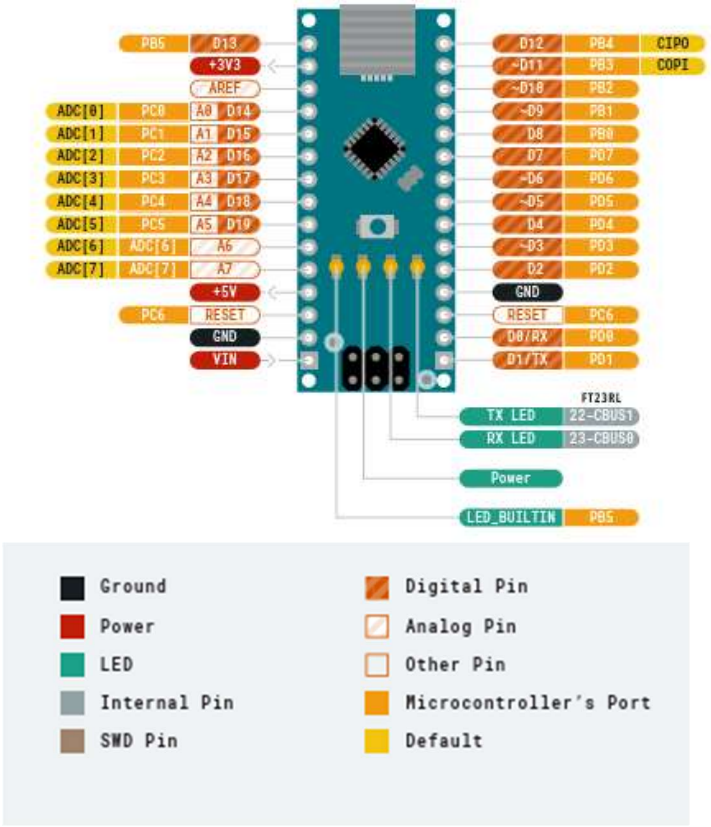

Fig.3.2 Pin diagram of Arduino

3.2 Power Supply:

A 9V battery is a compact, rectangular power source commonly used in small electronic devices, delivering a nominal voltage of 9 volts through its positive and negative terminals. Typically constructed as a single electrochemical cell or a series of smaller cells (e.g., six 1.5V cells in alkaline versions), it generates electricity via a

chemical reaction between its anode (zinc) and cathode (manganese dioxide), with an electrolyte facilitating ion flow. The battery's distinctive snap connector—featuring a male positive and female negative terminal—ensures secure attachment to circuits, making it ideal for portable applications requiring moderate power.

Fig.3.3 9V battery

In operation, a 9V battery supplies direct current (DC) to a circuit when connected. For instance, in a simple LED setup, the positive terminal links to the LED's anode through a resistor, while the negative terminal connects to the cathode, completing the circuit. Electrons flow from the negative to the positive terminal, illuminating the LED as energy converts from chemical to electrical form. In a buzzer circuit, the battery powers an oscillator, producing sound waves as current drives the component's vibration, demonstrating its ability to sustain active elements.

The battery's capacity, often measured in milliampere-hours (mAh), dictates runtime—higher capacity means longer use before depletion. In a basic radio, it might power a receiver for hours, converting stored energy into audio signals via a speaker. When used with a microcontroller like the Arduino Nano, the 9V battery connects to the VIN pin, providing unregulated power that the board's regulator steps down to 5V or 3.3V for stable operation. As the chemical reaction exhausts, voltage drops, eventually rendering the battery unable to drive the circuit—like a dimming LED or silencing buzzer—requiring replacement. This reliable, straightforward design makes the 9V battery a staple for powering small-scale electronics efficiently.

Specifications of 9V battery:

Parameter	Description
Nominal Voltage	9 volts
Terminal Type	Snap connector (PP3 type)
Dimensions	48.5 mm (H) × 26.5 mm (L) × 17.5 mm (W)
Shape	Rectangular (with rounded edges)
Weight Range	~30–50 grams (varies by chemistry)

Table 3.2 Specifications of 9V battery

3.2.1 Why choose 9V battery?

Using a 9V battery for the "Self-Checkout System Using Arduino and RFID" project offers distinct advantages over a traditional power supply with transformers and rectifiers, aligning with the prototype's design goals. First, portability is a key benefit. A 9V battery, compact and lightweight, powers the Arduino Nano and RFID module without tethering the trolley to a wall outlet, enabling mobility across a store—unlike a bulky transformer-based supply requiring a cord and plug. This eliminates the need for shoppers to navigate around cables, enhancing practicality in a dynamic retail environment.

Space-saving is another critical reason. A 9V battery, roughly 48mm x 26mm x 17mm, fits snugly onto the trolley's frame, occupying minimal space compared to a traditional setup with a transformer, rectifier, and associated wiring. These components, often housed in a larger enclosure (e.g., 100mm x 50mm or more), demand additional mounting area and increase the trolley's footprint, clashing with a sleek and compact design.

The project's minimum power requirements further favour a 9V battery. The Arduino Nano operates efficiently at 6-20V via VIN, drawing ~20mA idle, while the MFRC522 RFID module and 16x2 LCD require ~50mA and ~20mA, respectively—totalling under

100mA. A typical 9V battery (e.g., alkaline, ~500mAh) sustains this low current for hours, meeting the prototype's needs without the overkill of a transformer supply, which might deliver 1A or more, wasting energy through heat in rectification and regulation.

Additionally, a 9V battery simplifies assembly. It avoids the complexity of transformers stepping down AC, rectifiers converting to DC, and capacitors smoothing output—reducing setup time and potential failure points. For a proof-of-concept trolley, this straightforward, space-efficient power source ensures functionality without excess infrastructure.

3.3 RC522 RFID Module:

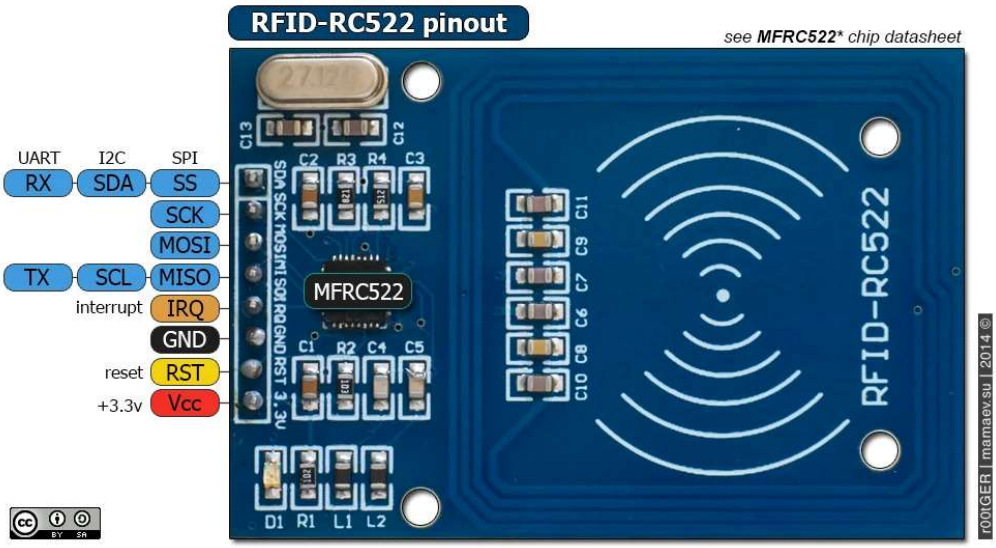

Fig3.4 RFID-RC522 module with pinout

The RC522 RFID module, officially the MFRC522, is a versatile, low-power integrated circuit designed by NXP Semiconductors for contactless communication at 13.56 MHz It serves as a reader/writer for RFID tags, supporting the ISO/IEC 14443, A protocol and MIFARE standards, such as MIFARE Classic 1K and 4K tags. These tags store data like unique identifiers or small datasets, making the module ideal for applications requiring fast, wireless identification, such as access control, inventory tracking, or smart systems. Operating at 13.56 MHz, it uses electromagnetic fields to power passive tags and retrieve their information within a typical range of 5-10 cm, depending on tag size and environmental factors.

The module integrates a transceiver with an onboard PCB coil antenna, eliminating the need for external RF components. It modulates and demodulates signals internally, handling tasks like data encoding and collision detection to ensure reliable communication with multiple tags. Its low-power design draws minimal current in idle states, peaking during RF transmission, which suits portable, battery-powered projects.

Interfacing with microcontrollers like the Arduino Nano is straightforward via the SPI (Serial Peripheral Interface) protocol. The module features eight pins: SDA (slave select), SCK (clock), MOSI (master out), MISO (master in), IRQ (interrupt request), GND (ground), RST (reset), and VCC (power). VCC operates at 2.5-3.3V, aligning with common microcontroller logic levels, while RST resets the chip and IRQ signals events like tag detection. SPI communication enables high-speed data transfer, with the Arduino sending commands to read or write tag data, processed by the MFRC522's 64-byte buffer.

Specifications of RC522 RFID Module:

Parameter	**Specification**
Operating Voltage	2.5V – 3.3V (typically powered with 3.3V)
Logic Level Voltage	3.3V (I/O lines **not** 5V tolerant)
Current Consumption	~13–26 mA (active), <80 µA (standby)
Communication Interface	SPI (default), I2C, UART (not commonly used)
SPI Clock Speed	Up to 10 MHz
Operating Frequency	13.56 MHz
Supported Card Types	ISO/IEC 14443 Type A (MIFARE Classic, Ultralight, etc.)
Read Range	Up to ~5 cm (with supplied small antenna)
Module Dimensions	~40 mm × 60 mm
Typical Tags Supported	MIFARE 1K, MIFARE 4K, Ultralight

Table3.3 Specifications of RC522 RFID Module

3.3.1 Why choose RC522 RFID Module?

The RC522 RFID module, or MFRC522, offers compelling advantages that make it a superior choice over other RFID modules for many applications, including its seamless fit for a Self-Checkout System. Its operation at 13.56 MHz with support for the ISO/IEC 14443 A protocol and MIFARE tags (e.g., Classic 1K, 4K) ensures compatibility with widely available, standardized tags, unlike some modules limited to proprietary or less common frequencies like 125 kHz. This frequency strikes a balance between range (up to 10 cm) and data speed, ideal for quick, reliable tag reads in a dynamic environment—other modules might sacrifice speed for distance or vice versa.

The RC522's SPI interface simplifies integration with microcontrollers like the Arduino Nano, using just four pins (SDA, SCK, MOSI, MISO) for high-speed communication, plus RST and IRQ for control. Its onboard PCB antenna eliminates external coil design, a hassle with some modules requiring custom tuning, reducing setup complexity and ensuring consistent performance. Power efficiency sets the RC522 apart too. Operating at 2.5-3.3V with low idle current and moderate RF draw, it aligns with battery-powered designs, unlike power-hungry alternatives that drain resources faster or demand higher voltages.

Finally, the RC522's compact form factor and all-in-one design minimize space needs, a boon for a trolley where bulkier modules with separate antennas could clutter the build. Its combination of compatibility, ease of use, efficiency, and reliability makes the RC522 a standout choice over less optimized RFID options.

Use case in this project:

The RFID module in a Self-Checkout System automates product identification by scanning RFID tags attached to items, instantly retrieving details like price and name. This enhances checkout speed, reduces errors, and improves the shopping experience. Additionally, the system supports real-time inventory tracking by updating stock levels as items are added and includes theft prevention through exit gate alerts for unpaid goods.

Overall, RFID technology makes the trolley efficient, user-friendly, and adaptable for modern retail environments.

3.4 Piezo-Buzzer:

A passive piezo-buzzer is a simple, versatile audio device used in electronics to produce sound through the piezoelectric effect, relying on an external signal source for operation. Unlike its active counterpart, which generates a fixed tone when powered by a DC voltage due to an internal oscillator, a passive piezo-buzzer requires an alternating current (AC) or pulsed signal—typically from a microcontroller, signal generator, or timer circuit—to drive its sound output. This makes it highly customizable, as the frequency, duration, and pattern of the sound depend entirely on the input signal, offering flexibility for various applications.

At its core, a passive piezo-buzzer consists of a piezoelectric ceramic disk attached to a metal diaphragm. When an alternating voltage is applied, the ceramic material deforms—expanding and contracting rapidly—causing the diaphragm to vibrate and emit sound waves. The pitch is determined by the frequency of the signal (commonly 1 kHz to 5 kHz), while the volume, typically around 70-90 dB at 10 cm, depends on the signal amplitude and the buzzer's design. Operating voltages range from 3V to 24V, with low current consumption (5-20 mA), making it energy-efficient for battery-powered devices.

Fig3.5 Piezo-Buzzer

Passive piezo-buzzers are widely used in consumer electronics, alarms, toys, and DIY projects due to their affordability, compact size (often 10-30 mm in diameter), and

durability. They excel in scenarios requiring distinct audio feedback, such as button presses, timers, or alerts, though they demand programming or circuitry to generate sound, unlike active buzzers. Limitations include lower sound complexity compared to speakers and a reliance on external drivers.

Use case of buzzer in this project:

In the Self-Checkout System, the buzzer provides audio feedback to enhance user interaction. Controlled by the Arduino Nano, this passive piezo-buzzer emits sounds for key events, complementing the 16x2 LCD's visual cues. When the RC522 RFID Module scans a product's tag, the Arduino triggers a short beep (e.g., 500 ms at 2 kHz) to confirm the item's addition to the bill, preventing errors. For unrecognized tags or issues, a longer or patterned tone (e.g., two 300 ms beeps) alerts the user to check the display. Pressing the push button to finalize billing activates a distinct 1-second tone, signalling completion. Powered via the 9V battery through the Arduino's 5V pin, the buzzer's low current draw (5-15 mA) ensures efficiency. Connected to a digital pin and ground, it delivers intuitive alerts, improving usability and efficiency in a busy shopping environment.

3.5 Push Buttons

Fig3.6 Push Button

In the Self-Checkout System, push buttons serve as critical user interface components, enabling manual control and interaction with the system via the Arduino Nano. These momentary switches, typically normally open (NO), allow shoppers to initiate or confirm actions, enhancing the trolley's usability. Connected to the Arduino's digital pins with pull-down resistors (e.g., 10kΩ to ground), they ensure a stable LOW state when

unpressed, switching to HIGH when pressed, which the Nano detects and processes accordingly. Powered by the 9V battery through the Arduino's 5V supply, their simplicity and low power draw (negligible current when idle) make them ideal for this portable application.

The primary role of push buttons is to provide user-driven inputs. For instance, one button might confirm the addition of a scanned item after the RC522 RFID Module reads a tag, ensuring intentional registration and preventing accidental entries. Another could finalize the billing process, prompting the Arduino to calculate the total, display it on the 16x2 LCD, and trigger a buzzer tone for confirmation. A third button might reset the system, clearing the current bill to start a new shopping session—useful in multi-user scenarios. These actions are programmed into the Arduino, with debouncing (e.g., a 50 ms delay in code) to eliminate false triggers from mechanical contact bounce, ensuring reliable operation.

Physically, the buttons are mounted on the trolley handle for ergonomic access, offering tactile feedback with a satisfying click. Their durability (rated for tens of thousands of presses) suits frequent use in a retail environment. By bridging the gap between automated RFID scanning and human control, push buttons make the trolley interactive and intuitive, empowering users to manage their shopping experience efficiently while complementing the system's smart features.

Specifications of push buttons:

Parameter	**Specification**
Type	Tactile Push Button
Operating Voltage	3.3V – 5V (compatible with Arduino)
Current Rating	Typically, 50 mA – 100 mA
Contact Resistance	< 100 mΩ
Number of Terminals	4 pins (2 pairs internally connected)
Standard Size	6 mm x 6 mm (common for breadboard use)

Table3.4 Specifications of push buttons

Use case of push buttons:

In the Self-Checkout System, push buttons are key user interface elements, enabling manual interaction via the Arduino Nano. These momentary, normally open (NO) switches connect to digital pins with pull-down resistors (e.g., 10kΩ), ensuring a LOW state when unpressed and HIGH when pressed, which the Nano detects. Powered by the 9V battery through the Arduino's 5V supply, they draw negligible current, fitting the portable design.

Their main role is to facilitate user inputs. One button might confirm an item's addition after the RC522 RFID Module scans a tag, preventing unintended entries. Another finalizes billing, prompting the Nano to compute the total, update the 16x2 LCD, and activate the buzzer for confirmation. A third could reset the system, clearing the bill for a new session. Debouncing (e.g., a 50 ms delay) in the Arduino code ensures reliable triggering by filtering contact bounce.

Mounted on the trolley handle for easy access, these buttons provide tactile feedback with a click, enhancing usability. Rated for thousands of presses, they withstand frequent use in a retail setting. By integrating human control with automated RFID scanning, push buttons make the trolley intuitive and responsive, allowing shoppers to manage their experience effectively.

3.6 16x2 LCD with I2C:

The 16x2 LCD with I2C is crucial for the trolley's visual interface, displaying 16 characters across two lines to show real-time shopping updates. Its I2C backpack reduces wiring to four pins (VCC, GND, SDA, SCL), connecting to the Arduino Nano via A4 and A5. Powered by a 9V battery through the Nano's regulator, it consumes 20-40 mA, depending on backlight use. The LiquidCrystal_I2C library ensures efficient communication, leaving Nano pins free for the RC522 RFID module, push buttons, and buzzer, supporting the trolley's portable design. The LCD instantly displays item details from RFID scans and the cumulative bill, updated as users confirm actions. Its adjustable backlight ensures readability in varied lighting, with a potentiometer for contrast tuning.

Built to endure vibrations and impacts, the cost-effective LCD suits retail settings. By providing clear feedback, it enhances trust and efficiency, making the RFID-based shopping process intuitive and reliable.

Fig3.7 16x2 LCD with I2C

The LCD shows item details from RFID scans and the cumulative bill, updating instantly as users confirm actions. Its adjustable backlight ensures visibility in varied lighting, from dim stores to bright markets, with a potentiometer for contrast tuning. Durable and cost-effective, it withstands retail vibrations and impacts. By providing clear feedback, the LCD enhances trust and efficiency, making the RFID-based shopping process intuitive and reliable.

Specifications of 16x2 LCD with I2C interface:

Specifications	**Details**
Type	16x2 Alphanumeric LCD
Interface	I2C (SDA, SCL)
I2C Address	Usually, 0x27 or 0x3F
Voltage	5V DC
Contrast	Adjustable via potentiometer
Controller	HD44780 with PCF8574 (I2C)

Table3.5 Specifications of 16x2 LCD with I2C

CHAPTER- 4

INSTALLATION OF IDE SOFTWARE

4.1 Arduino IDE Installation:

The Arduino Integrated Development Environment (IDE) is the primary software tool for programming the Arduino Nano in the Self-Checkout System. To begin, download the latest version from the official Arduino website (arduino.cc), compatible with your operating system (Windows, macOS, or Linux). After downloading, unzip the file and launch the IDE by double-clicking the executable. Connect the Arduino Nano to your computer using a USB A-to-Mini-B cable; the board draws power directly from the USB, illuminating its power LED. In the IDE, select **Tools > Board > Arduino Nano** and choose the correct port under **Tools > Port** (e.g., COM3 on Windows), identifiable by disconnecting and reconnecting the board to see which port disappears and reappears. Install necessary libraries—such as MFRC522 for RFID, LiquidCrystal_I2C for the LCD, and Wire for I2C communication—via **Sketch > Include Library > Manage Libraries**. This setup enables coding and uploading the firmware to control the trolley's operations.

Step 1 − First you must have your Arduino board (you can choose your favourite board) and a USB cable. In case you use Arduino UNO, Arduino Nano, Arduino Mega 2560, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the following image.

Fig4.1 USB A to Mini-B cable

Step 2 − Download Arduino IDE Software

You can get different versions of Arduino IDE from the https://www.arduino.cc/en/main/software on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.

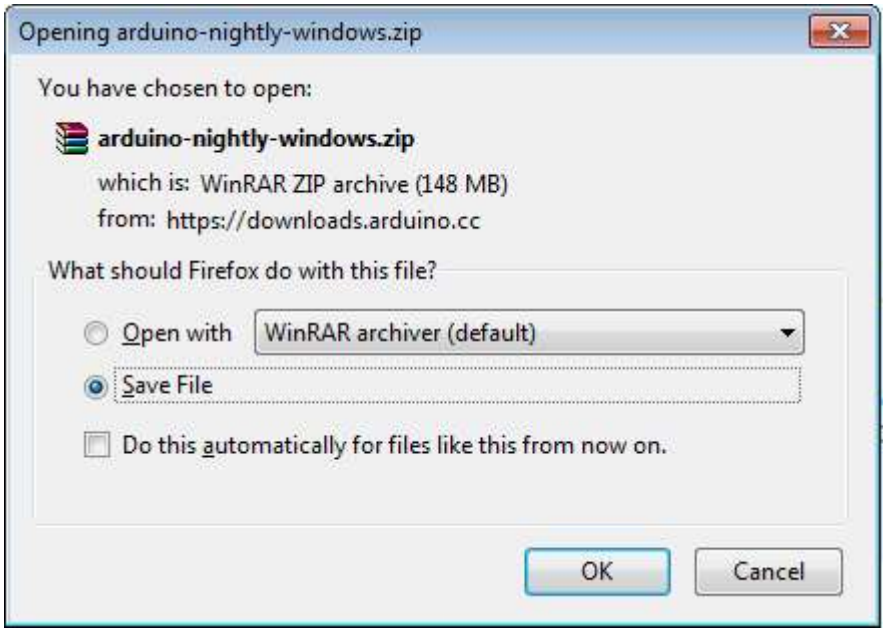

Fig4.2 Unzipping a file

Step 3 − Power up your board:

The Arduino Uno, Mega, and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (labelled PWR) should glow.

Step 4 − Launch Arduino IDE- After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE.

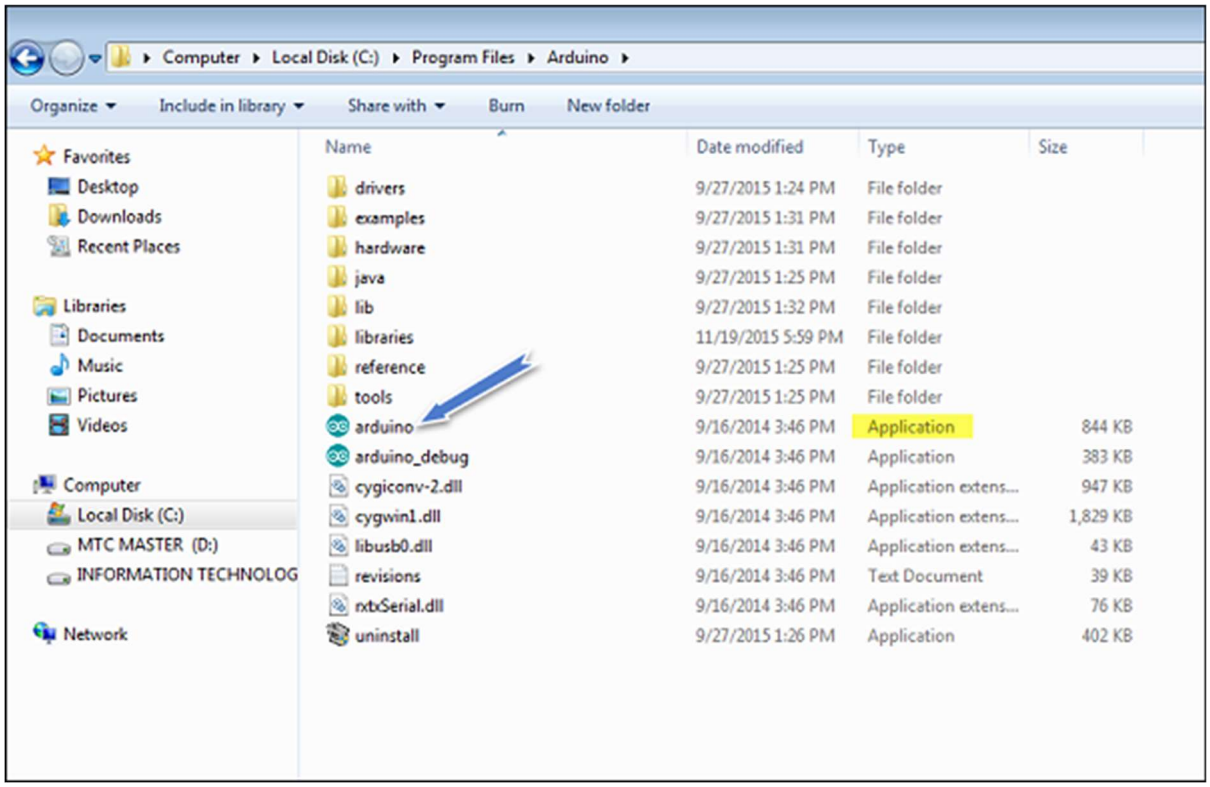

Fig4.3 Opening IDE

Step 5 − Open your first project:

Once the software starts, you have two options −

- Create a new project.
- Open an existing project example.

To create a new project, select File → **New**.

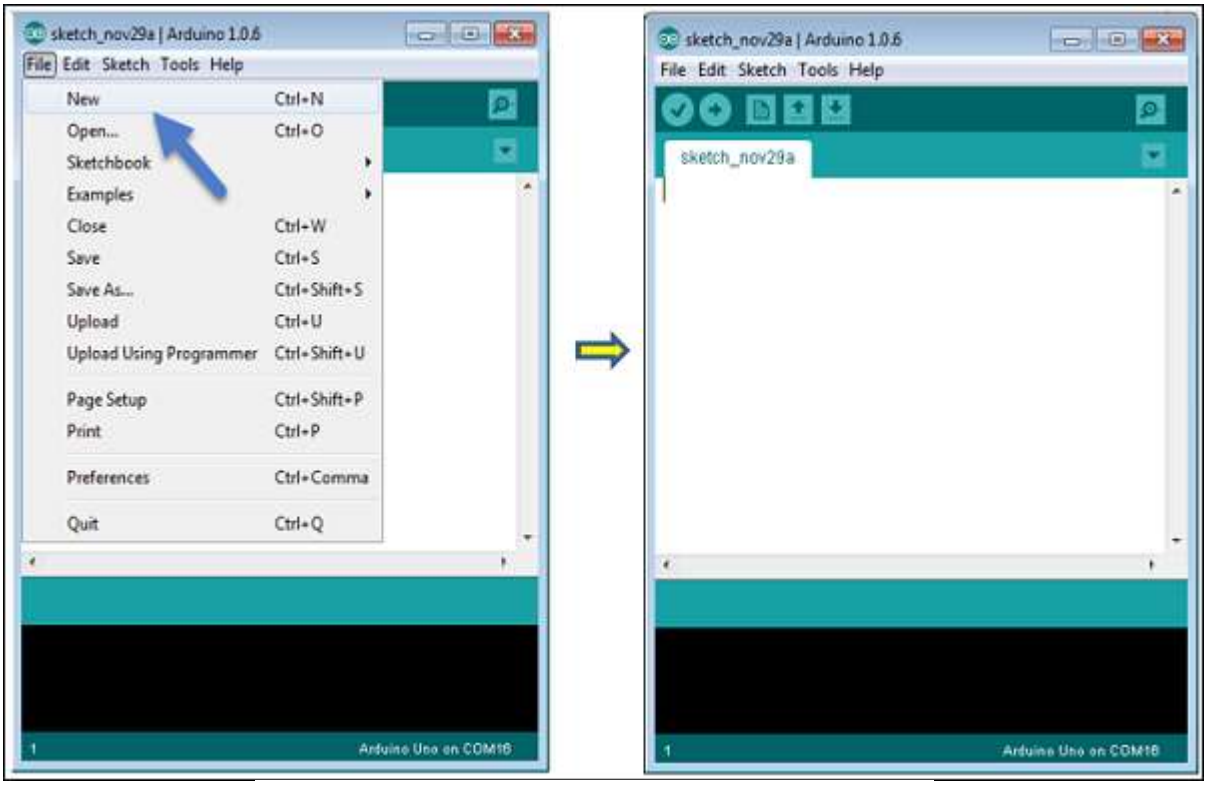

Fig4.4 Opening New Sketch

To open an existing project example, select File → Example → Basics → Blink.

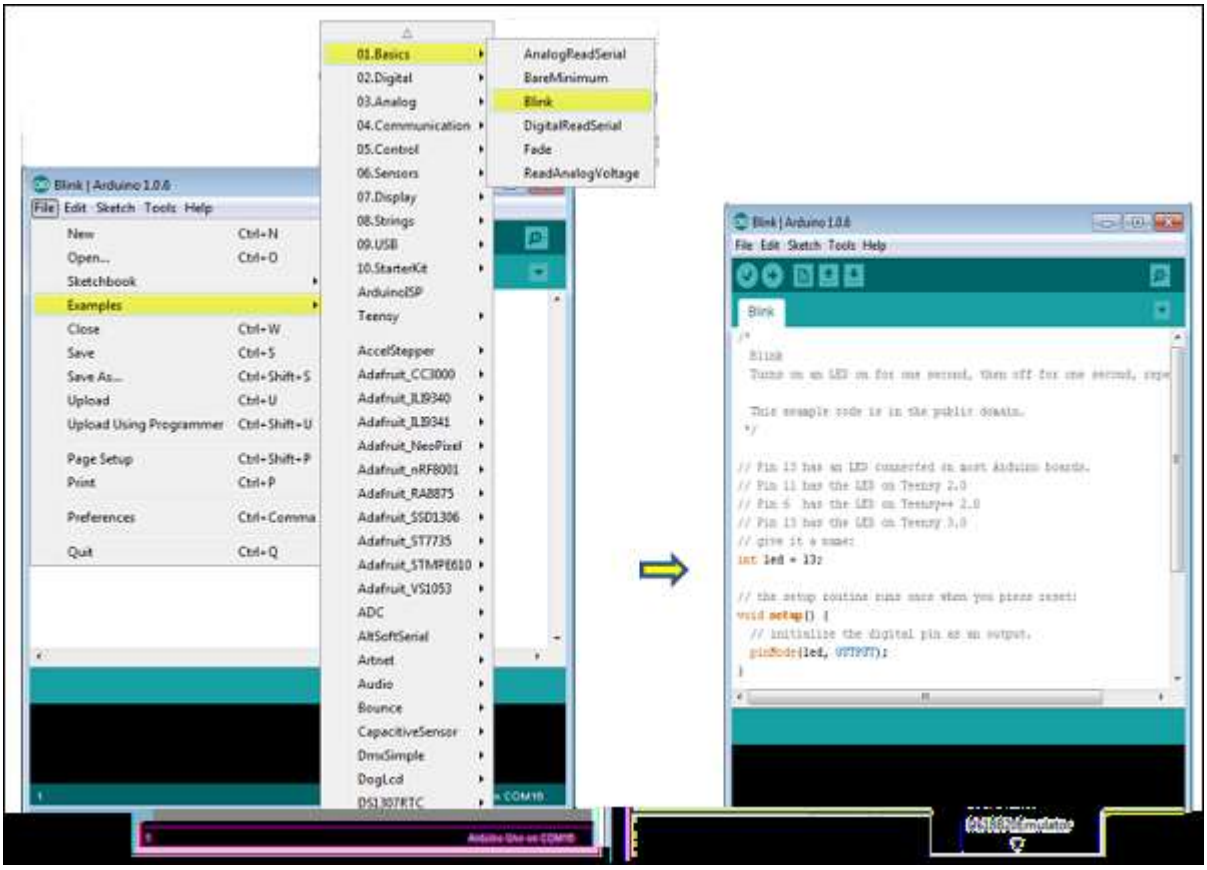

Fig4.5 Opening an example

Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

Step 6 − Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

Go to Tools → Board and select your board.

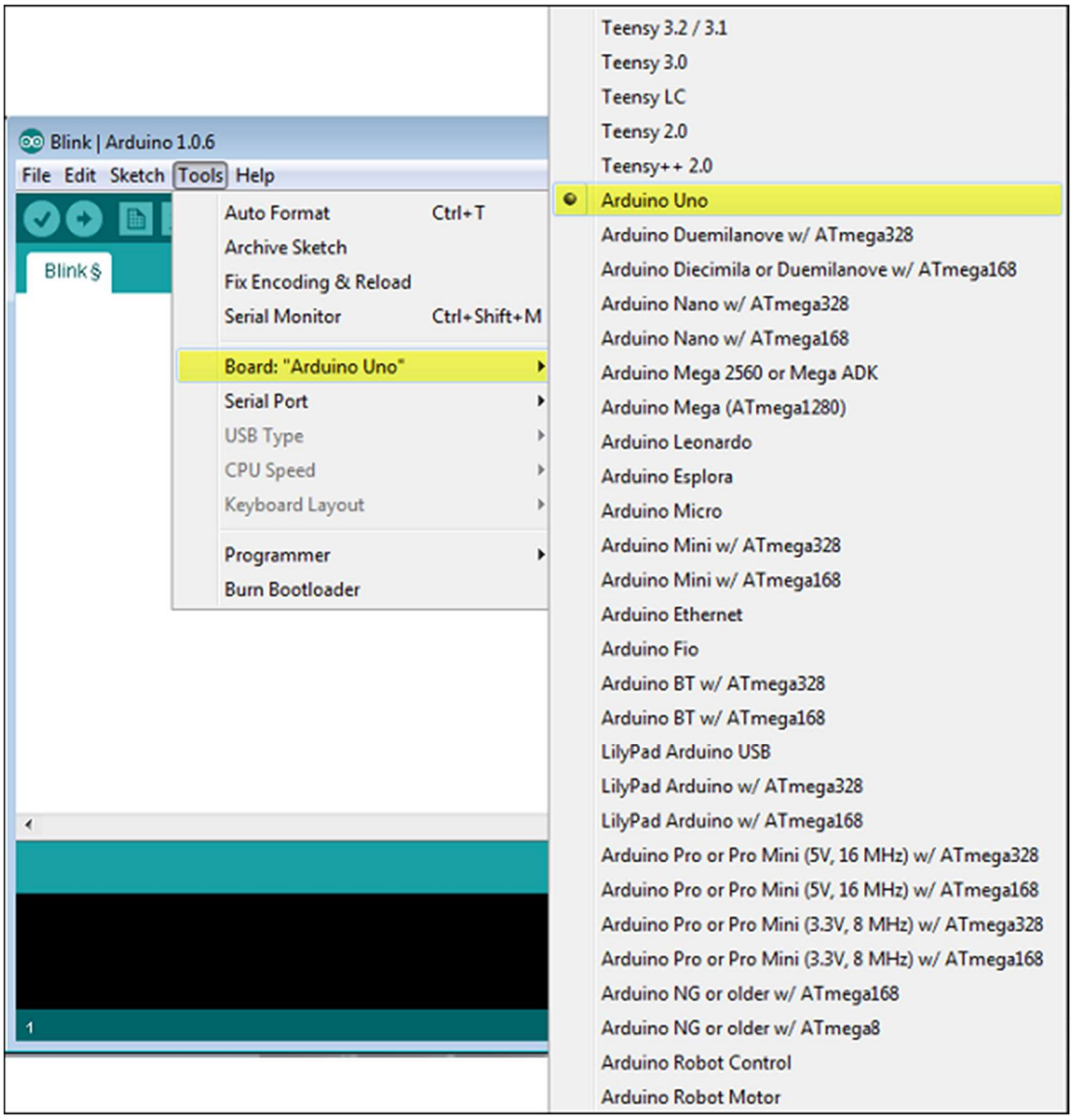

Fig4.6 Selecting the Arduino board

Step 7 − Select your serial port.

Select the serial device of the Arduino board. Go to **Tools → Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.

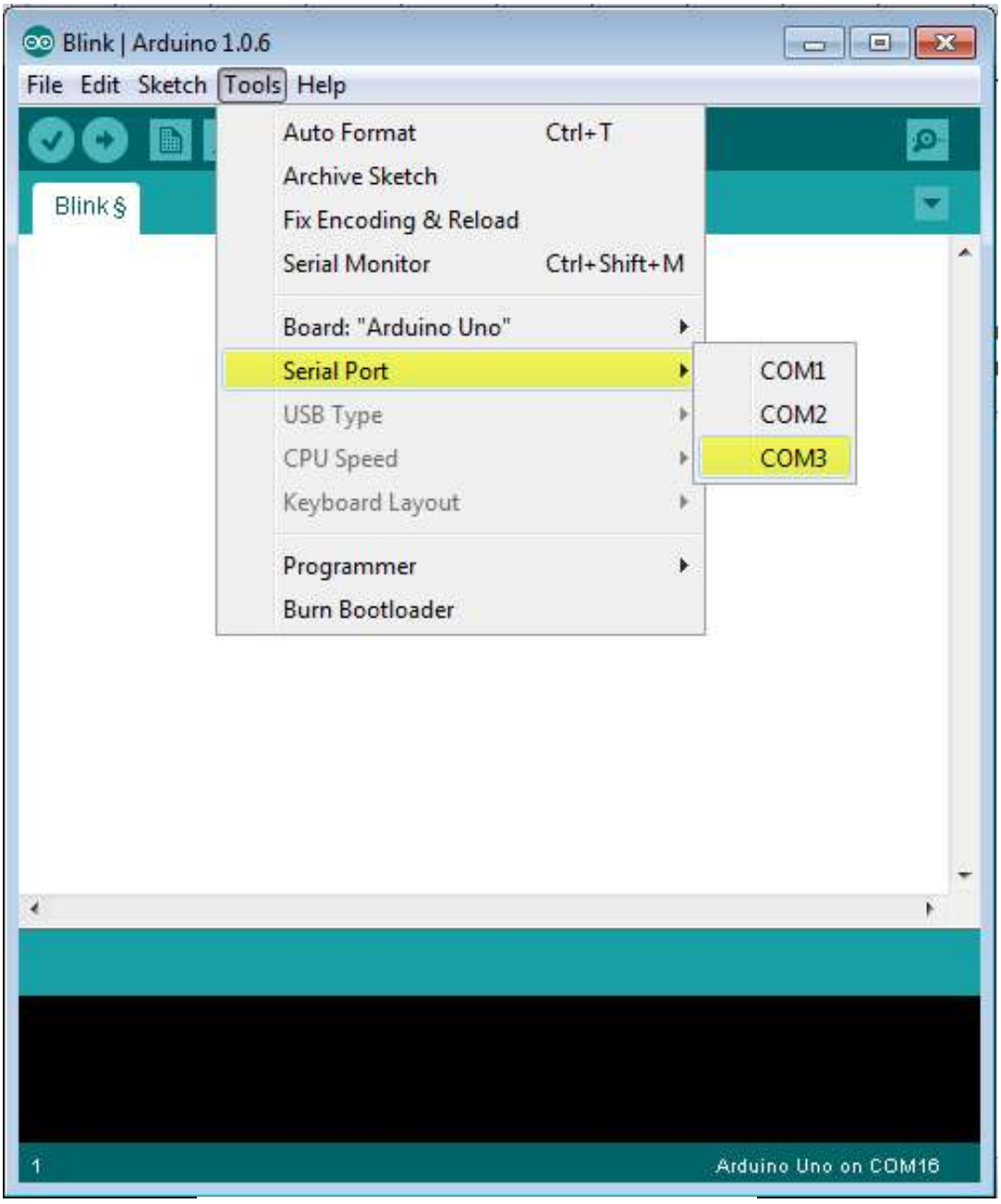

Fig4.7 Selecting the port of Arduino

Step 8 − Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.

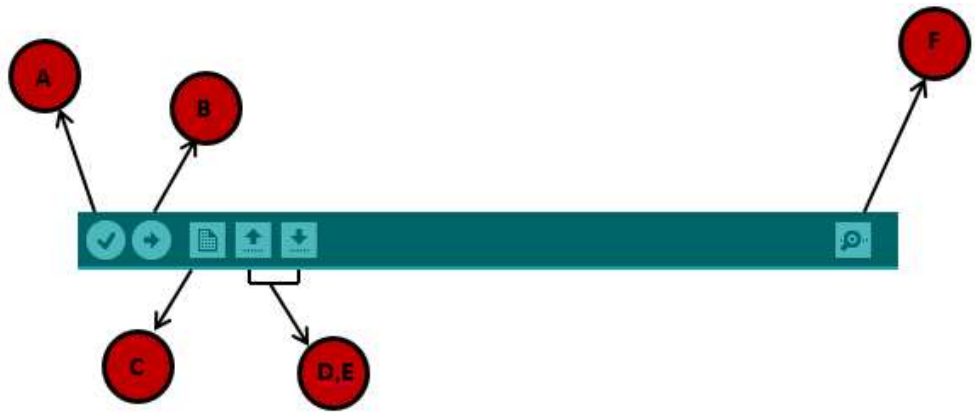

Fig4.8 Basic buttons for compiling

A − Used to check if there is any compilation error.

B − Used to upload a program to the Arduino board.

C − Shortcut used to create a new sketch.

D − Used to directly open one of the example sketch.

E − Used to save your sketch.

F − Serial monitor used to receive serial data from the board and send the serial data to the board.

Requirements of Arduino IDE:

- **Operating System**: Windows 7+, macOS 10.15+, or Linux (e.g., Ubuntu 18.04+).
- **Processor**: Minimum 1 GHz; 2 GHz recommended.
- **RAM**: At least 512 MB free; 1 GB preferred.
- **Storage**: ~200 MB free, plus 50-100 MB for libraries.
- **USB**: One USB 2.0/3.0 port for Nano connection.
- **Display**: 1024x768 resolution minimum.

- **Software**: Java Runtime Environment 8+ (usually bundled).
- **Internet**: Needed for initial download and libraries.
- **Drivers**: CH340/CH341 for Nano (Windows may need manual install).

4.2 Software Logic and Coding

The software for the Self-Checkout System is written in C/C++ within the Arduino IDE, managing the interaction between hardware components. The core program initializes the RC522 RFID Module, LCD, buzzer, and push buttons, then enters a loop to monitor inputs and update outputs. The setup() function configures pin modes (e.g., digital inputs for buttons, output for the buzzer), initializes the RFID reader via SPI, and sets up the LCD over I2C. In the loop() function, the Arduino continuously checks for RFID tag presence using the MFRC522 library's PICC_IsNewCardPresent() and PICC_ReadCardSerial() functions, retrieving the tag's unique ID (UID). This UID is matched against a predefined array of product data (e.g., item names and prices stored in PROGMEM to save SRAM), which the Nano then displays on the LCD and logs in a running total. Push button presses—detected via digital reads with software debouncing (e.g., a 50 ms delay)—trigger actions like confirming an item or finalizing the bill, accompanied by buzzer beeps (e.g., tone(buzzerPin, 2000, 500) for 500 ms at 2 kHz). The LCD updates dynamically using lcd.print() commands, while error handling (e.g., unrecognized tags) triggers a distinct buzzer pattern.

4.3 Efficiency of Arduino IDE:

The Arduino IDE's efficiency makes it an ideal choice for programming the Self-Checkout System, streamlining development and deployment. Its lightweight design (under 200 MB) runs smoothly on modest hardware, unlike resource-heavy alternatives like PlatformIO, which demands more memory and setup time, or AVR-GCC, requiring manual toolchain configuration—both less practical for rapid prototyping. The simplified C/C++ syntax abstracts complex microcontroller details, allowing quick coding compared to AVR-GCC's low-level programming, which needs direct register

manipulation and lacks built-in abstractions, slowing development for projects like this integrating RFID, LCD, and buzzer control. Built-in functions like digitalRead(), tone(), and a rich library ecosystem (e.g., MFRC522, LiquidCrystal_I2C) reduce coding time versus writing custom drivers in AVR-GCC or managing dependencies in PlatformIO's complex project structure. The IDE's one-click compile-and-upload process outperforms Arduino CLI's command-line approach, which, while scriptable, lacks an integrated GUI for instant feedback, crucial for verifying RFID reads or button inputs. Real-time serial monitoring further accelerates debugging over PlatformIO's separate terminal setup or AVR-GCC's reliance on external tools like AVRDUDE. Cross-platform compatibility (Windows, macOS, Linux) ensures consistency without the configuration overhead of AVR-GCC's platform-specific setups, while the shallow learning curve—unlike PlatformIO's steeper requirements for VS Code integration—suits beginners and fast-paced projects. Open-source flexibility allows customization if needed, but its focused feature set avoids the bloat of alternatives, making it perfectly efficient for the trolley's straightforward needs.

CHAPTER- 5

SYSTEM INTEGRATION AND IMPLEMENTATION

The Self-Checkout System's functionality hinges on seamless hardware integration. This chapter details the pin connections between the Arduino Nano and components, provides a unified code for operation, and outlines precautions to ensure safe assembly, bridging hardware and software for an efficient, user-friendly shopping experience.

5.1 Pin Connections between Components:

This section outlines the detailed steps for wiring all components of the Self-Checkout System to the Arduino Nano, ensuring proper functionality. Connections can be made using jumper wires on a breadboard for prototyping or soldered onto a perfboard for a permanent setup. Use 22-24 AWG solid-core wires for breadboards or stranded wires for flexibility in soldered designs, and maintain colour coding (e.g., red for power, black for ground) for clarity.

- **Arduino Nano to RC522 RFID Module**:

o Connect RC522's VCC to Nano's 3.3V pin (preferred for most modules to avoid overvoltage; check datasheet for 5V tolerance), GND to GND, SDA (slave select) to D10, SCK to D13, MOSI to D11, MISO to D12, and RST to D9. This establishes SPI communication.

o **Consideration**: If the module's range is poor, ensure a stable 3.3V supply; optionally, add a 100 µF capacitor across VCC-GND on the RC522 to filter noise.

Troubleshooting: If RFID fails to initialize, swap MISO/MOSI wires or test with a different D9-D13 pin set to rule out Nano SPI conflicts

- **Arduino Nano to 9V Battery**:

o Attach the 9V battery's positive terminal (red wire from a snap connector) to the Nano's VIN pin, which accepts 7-12V and regulates it to 5V internally. Connect the negative terminal (black wire) to any GND pin on the Nano.

o **Optional**: Add a slide switch in series with the positive wire for easy power control, rated for at least 1A to handle the system's current (~100-150 mA total).

o **Tip**: Secure the battery in a holder on the trolley to prevent movement, and use a multi-meter to verify 9V at VIN before proceeding.

- **Arduino Nano to Buzzer**:

o Wire the buzzer's positive pin (often marked with a "+" or longer leg) to Nano's D8 and the negative pin to GND. A passive piezo-buzzer requires PWM signals for sound.

o **Enhancement**: Insert a 100Ω resistor in series with the positive pin to limit current (~20 mA max) and protect the Nano's pin, though optional for most buzzers.

o **Tip**: Test the buzzer polarity with a 3V coin cell if unsure—reverse if no sound is produced during coding.

- **Arduino Nano to Push Button**:

o Connect one leg of the push button to Nano's D7 and the other to GND via a 10kΩ pull-down resistor. From the D7 leg, run a wire to Nano's 5V pin, so pressing the button pulls D7 HIGH.

o **Detail**: Use a momentary tactile switch (4-pin or 2-pin); for 4-pin types, connect diagonally opposite pins to ensure continuity.

o **Troubleshooting**: If the button registers false presses, increase the resistor to 20kΩ or check for loose breadboard contacts.

- **Arduino Nano to 16x2 LCD with I2C**:

o Link the LCD's I2C module VCC to Nano's 5V, GND to GND, SDA to A4, and SCL to A5, utilizing the Nano's I2C bus. The I2C backpack typically has a default address of 0x27 (verify with an I2C scanner sketch if the display fails).

o **Enhancement**: Solder a 10kΩ potentiometer to the I2C module's contrast pin (if accessible) for manual brightness adjustment, though most modules have a jumper for backlight control.

- **Final Steps**:

o After wiring, visually inspect for crossed wires or exposed pins. Use a multimeter to confirm 5V across VCC-GND points (e.g., Nano's 5V pin to GND) and continuity for each connection.

o **Layout Tip**: Position the RC522 near the trolley's scanning area, the LCD and button on the handle for accessibility, and the buzzer where sound is audible but not obstructed.

o **Power-Up Check**: Connect the 9V battery last, observing the Nano's power LED and LCD backlight to confirm initial success before uploading code.

The following page has the necessary pin connection that are to be made for this project, Self-Checkout System:

Pin Connection Table:

Component	Pin on Arduino Nano	Notes
MFRC522 RFID Reader	**Arduino Nano**	-
- SDA	D10	Chip select
- SCK	D13	SPI Clock
- MOSI	D11	SPI Master Out
- MISO	D12	SPI Master In
- RST	D9	Reset pin
- 3.3V	3.3V	Power
- GND	GND	Ground
16x2 LCD (I2C)	SDA, SCL	A4 (SDA), A5 (SCL)
- VCC	5V	Power
- GND	GND	Ground
Buzzer	D3	Active buzzer
Confirm Button	D4	Finalize product
Cancel Button	D5	Remove next scanned item

Table3.6 Pin connections between components

5.2 Steps for coding the Self-Checkout System

This section outlines the steps to code the Self-Checkout System using your provided program, integrating the RC522 RFID Module, buzzer, confirm and cancel

buttons, and 16x2 LCD with I2C for item scanning, addition/removal, and transaction confirmation.

1. **Install Required Libraries**:

o Open the Arduino IDE, go to **Sketch > Include Library > Manage Libraries**. Search and install MFRC522 (by Miguel Balboa) for RFID functionality and LiquidCrystal_I2C (by Frank de Brabander) for the LCD. Ensure SPI and Wire libraries are present (pre-installed with the IDE).

2. **Configure the Code Structure**:

o Create a new sketch. Add the includes: #include <SPI.h>, #include <MFRC522.h>, #include <Wire.h>, and #include <LiquidCrystal_I2C.h>. Define constants: SS_PIN 10, RST_PIN 9, BUZZER_PIN 3, I2C_ADDR 0x27, CONFIRM_BUTTON 4, and CANCEL_BUTTON 5. Declare objects: MFRC522 mfrc522(SS_PIN, RST_PIN) and LiquidCrystal_I2C lcd(I2C_ADDR, 16, 2). Add global variables: int totalSum = 0, bool removeNextItem = false, and bool wasItemAdded = false.

3. **Write the Setup Function**:

o In void setup(), initialize serial communication with Serial.begin(9600), start SPI with SPI.begin(), and initialize the RFID module with mfrc522.PCD_Init(). Set up the LCD with lcd.init() and lcd.backlight(), then display “Scan Product:” at (0, 0) using lcd.setCursor() and lcd.print(). Define pin modes: BUZZER_PIN as OUTPUT, and CONFIRM_BUTTON and CANCEL_BUTTON as INPUT_PULLUP.

4. **Implement Helper Functions**:

o Add void activateBuzzer() to toggle the buzzer HIGH for 100 ms and LOW for 300 ms using digitalWrite(). Create void confirmTransaction() to reset totalSum to 0, display “Pay At Counter” and the total (or “No items scanned” if zero) on the LCD for 5 seconds, then revert to “Scan Product:”.

5. **Code the Main Loop**:

o In void loop(), check CONFIRM_BUTTON (LOW with INPUT_PULLUP) to call confirmTransaction(). Check CANCEL_BUTTON to set removeNextItem = true, clear

the LCD, and show “Scan to remove”. Detect RFID tags with mfrc522.PICC_IsNewCardPresent() and mfrc522.PICC_ReadCardSerial(). Authenticate block 4 with a default key (0xFF), read data into a buffer, and parse it into priceValue (digits) and productName (letters/spaces). If valid, update totalSum (add or subtract based on removeNextItem), display the action on the LCD, trigger the buzzer, and reset the display after 5 seconds.

6. **Test and Upload**:

o Connect the Nano via USB, select “Arduino Nano” under **Tools > Board**, and the correct port under **Tools > Port**. Click “Verify” to compile, then “Upload” to program the Nano. Open the Serial Monitor (Ctrl+Shift+M) at 9600 baud to debug output.

5.3 Steps for Connecting the Components:

This section details the physical connections between the Arduino Nano and all components—RC522 RFID Module, buzzer, 9V battery, confirm button, cancel button, and 16x2 LCD with I2C—matching the pin assignments in your code.

1. **Prepare Tools and Materials**:

o Gather a breadboard, 22-24 AWG jumper wires (color-coded: red for power, black for ground), a 9V battery with snap connector, and the components. Optionally, use a soldering iron for permanent connections.

2. **Connect Arduino Nano to 9V Battery**:

o Attach the 9V battery’s positive terminal (red wire) to the Nano’s VIN pin and the negative terminal (black wire) to a GND pin. This powers the system, regulated to 5V internally. Secure the battery in a holder on the trolley.

3. **Wire Arduino Nano to RC522 RFID Module**:

o Connect RC522’s VCC to Nano’s 3.3V (or 5V if module supports), GND to GND, SDA to D10 (SS_PIN), SCK to D13, MOSI to D11, MISO to D12, and RST to D9 (RST_PIN). This sets up SPI for RFID tag reading.

4. **Attach Arduino Nano to Buzzer**:

o Link the buzzer's positive pin (marked "+" or longer leg) to Nano's D3 (BUZZER_PIN) and the negative pin to GND. A passive piezo-buzzer is assumed, driven by digitalWrite() for simple beeps.

5. **Connect Arduino Nano to Confirm Button**:

o Wire one leg of the confirm button to Nano's D4 (CONFIRM_BUTTON) and the other to GND. Using INPUT_PULLUP in the code leverages the Nano's internal pull-up resistor, so pressing pulls D4 LOW—no external resistor needed.

6. **Link Arduino Nano to Cancel Button**:

o Connect one leg of the cancel button to Nano's D5 (CANCEL_BUTTON) and the other to GND. Like the confirm button, INPUT_PULLUP detects a LOW state when pressed, simplifying the circuit.

7. **Attach Arduino Nano to 16x2 LCD with I2C**:

o Wire the LCD's I2C module VCC to Nano's 5V, GND to GND, SDA to A4, and SCL to A5. The code uses address 0x27 (I2C_ADDR); confirm with an I2C scanner if the display doesn't work.

8. **Verify and Power On**:

o Double-check all connections with a multimeter (5V at Nano's 5V-GND, continuity for each wire). Position the RC522 for scanning, buttons on the handle, and LCD for visibility. Connect the 9V battery last, ensuring the Nano's power LED lights up.

5.4 Precautions before connection:

To safeguard the Self-Checkout System's components and ensure successful integration, adhere to these precautions before making any connections:

1. **Power Disconnection:**

Unplug the 9V battery and USB cable before wiring to prevent shorts or damage to the Nano or RC522 module.

2. **Voltage Compatibility:**

Confirm the RC522 supports 5V; if unsure, use the Nano's 3.3V pin to avoid damage. Check with a multimeter if needed.

3. **ESD Protection:**

Discharge static by touching a grounded metal object before handling components to avoid damaging pins or chips.

4. **Pin Verification:**

Double-check wiring (e.g., D10-SDA, D9-RST, D3-buzzer, D4-confirm, D5-cancel, A4/A5-I2C) against the code and datasheets to ensure proper communication.

5. **Button Wiring:**

Connect confirm and cancel buttons to GND with no external pull-ups, as code uses INPUT_PULLUP.

6. **Buzzer Polarity:**

Connect the buzzer's "+" (longer leg) to D3; reversed polarity won't harm it but will mute it.

7. **I2C Address Check:**

Ensure the LCD address is 0x27. If it fails to display, run an I2C scanner—some use 0x3F or others.

CHAPTER- 6

SOFTWARE DESCRIPTION

This chapter provides an overview of the software developed for the Self-Checkout System, detailing the Arduino sketch that integrates the Arduino Nano with the RC522 RFID Module, buzzer, buttons, and 16x2 LCD with I2C. The code, written in C/C++ within the Arduino IDE, manages item scanning, user interaction, and feedback, ensuring efficient operation.

6.1 Software Structure:

The software is structured into three main parts: library inclusions, global definitions, and functional blocks. It begins with #include statements for SPI.h, MFRC522.h, Wire.h, and LiquidCrystal_I2C.h to enable SPI communication, RFID functionality, I2C interfacing, and LCD control. Pin definitions (SS_PIN 10, RST_PIN 9, BUZZER_PIN 3, CONFIRM_BUTTON 4, CANCEL_BUTTON 5, I2C_ADDR 0x27) and object declarations (MFRC522, LiquidCrystal_I2C lcd) set up hardware connections. Global variables (totalSum, removeNextItem, wasItemAdded) track the transaction state. The code is organized into setup() for initialization and loop() for continuous operation, with helper functions (activateBuzzer(), confirmTransaction()) for specific tasks.

6.2 Key Functionalities:

1. **RFID Scanning**: The loop() function uses mfrc522.PICC_IsNewCardPresent() and mfrc522.PICC_ReadCardSerial() to detect tags, authenticating block 4 with a default key (0xFF) and reading data into a buffer. It parses this into productName and priceValue for processing.

2. **Item Management**: If removeNextItem is false, the code adds priceValue to totalSum; if true, it subtracts it, updating the LCD and serial output accordingly.

3. **User Input**: CONFIRM_BUTTON (D4) resets totalSum via confirmTransaction(), while CANCEL_BUTTON (D5) toggles removeNextItem for item removal, both using INPUT_PULLUP for simplicity.

4. **Feedback**: The LCD displays actions (“Added: [product]”, “Total: Rs [sum]”) and errors (“Auth Failed!”), while activateBuzzer() triggers a 100 ms beep for each scan.

6.3 Software Efficiency:

The code optimizes resource use with minimal delays (e.g., 500 ms debounce, 5000 ms display), efficient parsing of tag data, and reliance on Arduino’s built-in pull-ups. The serial monitor at 9600 baud aids debugging, and the modular design allows easy updates, such as adding more tag blocks or feedback patterns.

CHAPTER- 7

ADVANTAGES AND APPLICATIONS

The Self-Checkout System enhances retail efficiency through innovative technology. This chapter examines its advantages, such as streamlined shopping and cost-effectiveness, alongside disadvantages, including technical limitations and dependencies, offering a balanced assessment of its performance and potential challenges in real-world applications.

6.1 Advantages:

1. **Automated Item Tracking**: RC522 RFID Module scans products automatically, eliminating manual barcode scanning and reducing errors.

2. **Real-Time Feedback**: 16x2 LCD displays item details and total; buzzer provides audible confirmation.

3. **User Control**: Confirm and cancel buttons allow adding, removing, or finalizing items flexibly.

4. **Time Efficiency**: Speeds up shopping and checkout by automating item logging and totalling.

6.2 Applications:

1. **Smart Retail Checkouts:** RFID-based systems in stores for automated billing at exit gates.

2. **Inventory Management:** RFID trackers in warehouses to monitor stock levels in real time.

3. **Library Book Tracking:** RFID tags on books for self-checkout and return automation.

CHAPTER- 8

CONCLUSION AND FUTURE SCOPE

8.1 Conclusion:

The Self-Checkout System effectively demonstrates the integration of affordable, accessible technology to enhance the retail experience, utilizing an Arduino Nano as the core controller alongside the RC522 RFID Module, a passive buzzer, two push buttons (confirm and cancel), and a 16x2 LCD with I2C interface. The system achieves its primary objective of automating item tracking by leveraging RFID technology to scan tags seamlessly, parsing their data into product names and prices, as implemented in the provided code. This automation eliminates the need for manual barcode scanning, significantly reducing human error and expediting the addition or removal of items from the trolley. The LCD provides real-time visual feedback, displaying scanned items and the running total (totalSum), while the buzzer's short beeps (100 ms on, 300 ms off) offer audible confirmation, ensuring users remain informed of each action. The inclusion of confirm and cancel buttons, wired with INPUT_PULLUP to pins D4 and D5, grants shoppers' precise control over their selections—allowing them to finalize transactions or remove items—enhancing flexibility and user engagement. Powered by a 9V battery, the trolley's portability makes it practical for mobile use within stores, independent of fixed power sources. The code's robust logic, including authentication of RFID block 4 and error handling for failed reads, ensures reliable operation, while its reset functionality (confirmTransaction()) prepares the system for repeated use. Cost-effectiveness is a key strength, achieved through low-cost components like the Nano and RC522, making this a viable prototype for small-scale retail applications. Overall, the Self-Checkout System minimizes checkout delays, reduces errors, and delivers a user-friendly, efficient shopping solution, fulfilling its design goals with a practical, technology-driven approach.

8.2 Future Scope:

1. **Wireless Connectivity**: Integrate Bluetooth (e.g., HC-05) or Wi-Fi (e.g., ESP8266) to transmit item data and totals to a mobile app or store server.

2. **Expanded Product Database**: Use EEPROM or an SD card module to store a larger, dynamic list of products beyond hardcoded tag data.

3. **Payment Integration**: Add an NFC module or card reader for in-trolley payment processing.

4. **Enhanced Display**: Replace the 16x2 LCD with an OLED or TFT screen for detailed item lists and graphical interfaces.

5. **Battery Optimization**: Implement a rechargeable battery pack with a charging circuit and low-battery warning on the LCD.

6. **User Profiles**: Enable multi-user support via RFID keycards to log individual shopping histories.

7. **Voice Output**: Upgrade the buzzer to a speaker module for voice feedback (e.g., “Milk added”).

8. **Security Features**: Add weight sensors or alarms to detect unpaid items, enhancing anti-theft measures.

9. **Cloud Sync**: Connect to a cloud database for real-time price updates and inventory tracking.

10. **Mobile App Control**: Develop an app to remotely confirm transactions or view trolley status.

APPENDIX: SOFTWARE CODE

```
#include <SPI.h>

#include <MFRC522.h>

#include <Wire.h>

#include <LiquidCrystal_I2C.h>


#define SS_PIN 10   // SDA Pin for RFID

#define RST_PIN 9   // Reset Pin for RFID

#define BUZZER_PIN 3  // Buzzer control pin

#define I2C_ADDR 0x27 // Default I2C address for LCD

#define CONFIRM_BUTTON 4  // Confirm button pin

#define CANCEL_BUTTON 5   // Cancel button pin


MFRC522 mfrc522(SS_PIN, RST_PIN);

LiquidCrystal_I2C lcd(I2C_ADDR, 16, 2); // 16x2 LCD


Int totalSum = 0;

Bool removeNextItem = false;

Bool wasItemAdded = false;


Void setup() {


   Serial.begin(9600);

   SPI.begin();
```

```
    Mfrc522.PCD_Init();

    Lcd.init();
    Lcd.backlight();
    Lcd.setCursor(0, 0);
    Lcd.print("Scan Product:");
    pinMode(BUZZER_PIN, OUTPUT);
    pinMode(CONFIRM_BUTTON, INPUT_PULLUP);
    pinMode(CANCEL_BUTTON, INPUT_PULLUP);
    Serial.println("Place a Product:");

}

Void activateBuzzer() {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(100);
    digitalWrite(BUZZER_PIN, LOW);
    delay(300);

}


Void loop() {

    // Confirm Button: Reset Total Amount
```

```
If (digitalRead(CONFIRM_BUTTON) == LOW) {
   confirmTransaction();

}

// Cancel Button: Activate Remove Mode

If (digitalRead(CANCEL_BUTTON) == LOW) {

   removeNextItem = true;
   wasItemAdded = false;
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("Scan to remove");
   Serial.println("Waiting for item to remove…");

   Delay(500);

}    // Check for new scanned product

If (!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()) {
 Return;

}

Byte block = 4;
Byte buffer[18];
```

```
Byte size = sizeof(buffer);
MFRC522::MIFARE_Key key;
For (byte I = 0; I < 6; i++) key.keyByte[i] = 0xFF;

If (mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(mfrc522.uid)) != MFRC522::STATUS_OK) {

    Serial.println("Auth Failed!");
    Lcd.setCursor(0, 1);
    Lcd.print("Auth Failed!");
    Return;

}

If (mfrc522.MIFARE_Read(block, buffer, &size) == MFRC522::STATUS_OK) {
    Int priceValue = 0;
    String productName = "";

    For (byte I = 0; I < 16; i++) {

        Char c = buffer[i];
        If (isdigit©) {
            priceValue = priceValue * 10 + (c – '0');

        } else if (isalpha© || c == ' ') {
            productName += c;
```

```
    }

}

If (priceValue > 0 && productName.length() > 0) {
    Lcd.clear();
    Lcd.setCursor(0, 0);

If (removeNextItem) {

        If (totalSum >= priceValue) {
            totalSum -= priceValue;
            Serial.print("Removed Item: ");
            Serial.print(productName);
            Serial.print(" Rs ");
            Serial.println(priceValue);
            Lcd.print("Removed Item");
            Lcd.setCursor(0, 1);
            Lcd.print(productName);

        } else {

            Serial.print("Didn't Add Item: ");
            Serial.println(productName);
            Lcd.print("Didn't Add Item");
```

```
        Lcd.setCursor(0, 1);
        Lcd.print(productName);

    }
    removeNextItem = false;
  }

  else {

    totalSum += priceValue;
    wasItemAdded = true;

    Serial.print("Added: ");
    Serial.print(productName);
    Serial.print(" Rs ");
    Serial.println(priceValue);
    Lcd.print("Added: ");
    Lcd.print(productName);

    }

  Serial.print("Total: Rs ");
  Serial.println(totalSum);
  Lcd.setCursor(0, 1);
  Lcd.print("Total: Rs ");
  Lcd.print(totalSum);
```

```
            activateBuzzer();
            delay(5000);  // Increased delay to 5 seconds
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Scan Product:");

        }

    } else {

        Serial.println("Read failed!");
        Lcd.setCursor(0, 1);
        Lcd.print("Read Failed!");

    }

    Mfrc522.PICC_HaltA();
    Mfrc522.PCD_StopCrypto1();

}

Void confirmTransaction() {
 Lcd.clear();

     If (totalSum == 0) {
        Lcd.setCursor(0, 0);
```

```
    Lcd.print("No items scanned");
    Serial.println("No items scanned!");
    Delay(5000); // Show message for 5 seconds

} else  {

    Lcd.setCursor(0, 0);
    Lcd.print("Pay At Counter");
    Lcd.setCursor(0, 1);
    Lcd.print("Total: Rs ");
    Lcd.print(totalSum);
    Serial.println("Transaction Confirmed!");
    Delay(5000); // Display total for 5 seconds before reset

}

totalSum = 0;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Scan Product:");
Serial.println("Total reset to 0.");

}
```

REFERENCES

1. **Arduino Official Website**. (2023). *Arduino Nano Documentation*. Available: https://www.arduino.cc/en/Guide/NANO

- Source for Arduino Nano specifications and pinout details.

2. **MFRC522 Library by Miguel Balboa**. (2023). *GitHub Repository*. Available: https://github.com/miguelbalboa/rfid

- Documentation and source code for the MFRC522 RFID library used with the RC522 module.

3. **LiquidCrystal_I2C Library by Frank de Brabander**. (2023). *GitHub Repository*. Available: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

- Reference for the I2C LCD library used to interface the 16x2 LCD.

4. **NXP Semiconductors**. (2018). *MFRC522 Standard Performance MIFARE and NTAG Frontend Datasheet*. Available: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

- Technical datasheet for the RC522 RFID module's chip, detailing SPI communication and tag reading.

5. **Arduino IDE Documentation**. (2023). *Arduino Software Guide*. Available: https://www.arduino.cc/en/Guide/Environment

- Official guide for installing and using the Arduino IDE for programming the Nano.

6. **SparkFun Electronics**. (2020). *Hookup Guide: RFID Basics*. Available: https://learn.sparkfun.com/tutorials/rfid-basics

- Tutorial on RFID technology principles applied in the project.

7. **Osoyoo**. (2021). *Arduino Lesson: Using Push Button*. Available: https://osoyoo.com/2017/07/03/arduino-lesson-push-button/

- Reference for wiring and coding push buttons with Arduino.