CHAPTER 1

INTRODUCTION

Traditional ticketing system for railways involves serpentine queues for passengers and requires constant human processing. This wastes a lot of passenger's time and energy.

Attempts to fix this problem include coupon books for passengers and more recently smart cards for standard railways. In standard railway, the purpose of the smartcard is to automatically print a ticket for the passenger. Using the smartcard requires the passenger to wait for the machine to generate the ticket.

However, if a smart card system is to be implemented for Metrorail the purpose of this smartcard will be to give the passenger access to the station by commanding the turnstile to rotate(or a flap gate to open) and then again enable the passenger to exit at the destination station.

Building on this idea, we propose a more advanced automatic ticketing system using Near Field Communication keeping in mind the specific needs and requirements of Metrorail systems in India and also the convenience of the passengers.

The proposed system consists of an NFC based device which is kept at the station and an NFC tag/ NFC enabled mobile phone which the user carries, the tag/mobile phone used by the user has a unique identification number of the user. The user has the NFC tag which is used to gain access to the station as well as to exit from the station. All this transaction happens at the touch of the tap at the NFC device by the NFC tag. This saves a lot of user's time and energy. The user does not require an NFC enabled phone, the NFC tag can be used which can be bought by anyone at a very less cost. Thus the system is available to everyone at a very less cost.

The project consists of an NFC based device which is kept at the station and an NFC tag which the user carries, the tag used by the user is the unique identification card of the user. The user taps the tag at the device kept at the station and the device reads the unique ID, and send the users information like the UID, the station code where the user boarded and the number of passengers, these details are sent to the server and the server maintains a record of the user, that the user is travelling.

Once the user completes the journey and he want an exit from the station the user has to tap at the RFID device then only the exit doors open and the user can exit.

When the user taps the tag at the destination station based on the two stations the appropriate amount is deducted from the users account.

CHAPTER 2

INTRODUCTION TO ARDUINO MICROCONTROLLER

2.1 Overview:

The Arduino Micro is a microcontroller board based on the ATmega32u4 (datasheet), developed in conjunction withAdafruit. It has 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHzcrystal oscillator, a micro USB connection, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a micro USB cable to get started. It has a form factor that enables it to be easily placed on a breadboard.

The Micro is similar to the Arduino Leonardo in that the ATmega32u4 has built-in USB communication, eliminating the need for a secondary processor. This allows the Micro to appear to a connected computer as a mouse and keyboard, in addition to a virtual (CDC) serial / COM port.

2.2 Power

The Arduino Micro can be powered via the micro USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from a DC power supply or battery. Leads from a battery or DC power supply can be connected to the Gnd and Vin pins.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin.

5V. The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.

3V. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

Ground pins.

2.3 Memory

The ATmega32u4 has 32 KB (with 4 KB used for the bootloader). It also has 2.5 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library)

Input and Output

Each of the 20 digital i/o pins on the Micro can be used as an input or output, using pinMode(), digitalWrite() and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms.

In addition, some pins have specialized functions:

Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data using the ATmega32U4 hardware serial capability. Note that on the Micro, the Serial class refers to USB (CDC) communication; for TTL serial on pins 0 and 1, use the Serial1 class.

TWI: 2 (SDA) and 3 (SCL). Support TWI communication using the Wire library.

External Interrupts: 0(RX), 1(TX), 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt()function for details.

PWM: 3, 5, 6, 9, 10, 11 and 13. Provide 8-bit PWM output with the analogWrite() function.

SPI: on the ICSP header. These pins support SPI communication using the SPI library. Note that the SPI pins are not connected to any of the digital I/O pins as they are on the Arduino Uno, they are only available on the ICSP connector and on the nearby pins labelled MISO, MOSI and SCK.

RX_LED/SS this is an additional pin with respect to the Leonardo. It is connected to the RX_LED that indicates the activity of transmission during USB communication, but is can also use as slave select pin (SS) in SPI communication.

LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

Analog Inputs: A0-A5, A6 - A11 (on digital pins 4, 6, 8, 9, 10, and 12). The Micro has a total of 12 analog inputs, pins from A0 to A5 are labelled directly on the pins and the other ones that you can access in code using the constants from A6 trough A11 are shared respectively on digital pins 4, 6, 8, 9, 10, and 12. All of which can also be used as digital I/O. Each analog input provide 10 bits of resolution (i.e. 1024 different values). By default the analog inputs measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function.

There are a couple of other pins on the board:

AREF. Reference voltage for the analog inputs. Used with analogReference().

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

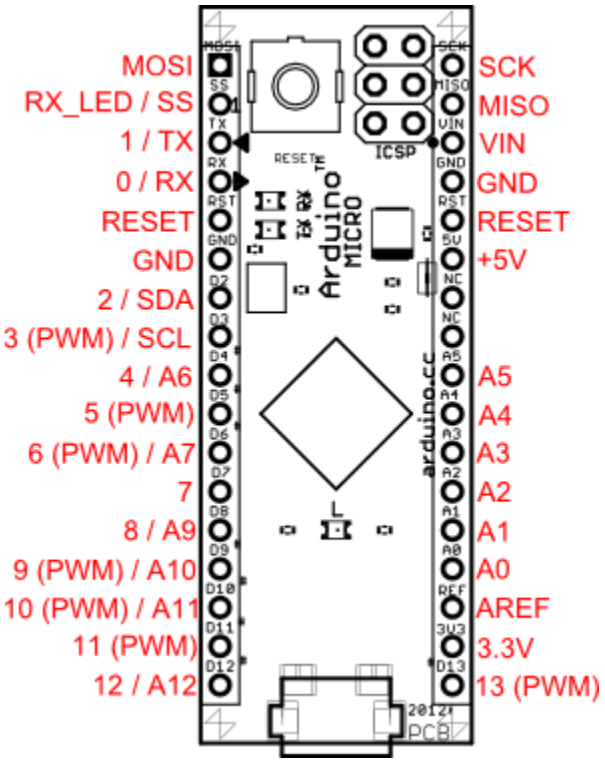


Figure 2.1 PIN DIAGRAM of Arduino Micro

2.4 Communication

The Micro has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers.

TheATmega32U4 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). The 32U4 also allows for serial (CDC) communication over USB and appears as a virtual com port to software on the computer. The chip also acts as a full speed USB 2.0 device, using standard USB COM drivers. On Windows, a .inf file is required.

The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Micro's digital pins.

The ATmega32U4 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

The Micro appears as a generic keyboard and mouse, and can be programmed to control these input devices using the Keyboard and Mouse classes.

2.5 Programming

The Micro can be programmed with the Arduino software (download). Select "Arduino Micro from the Tools >Boardmenu. For details, see the reference and tutorials.

The ATmega32U4 on the Arduino Micro comes pre-burned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the AVR109 protocol.

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

Automatic (Software) Reset and Bootloader Initiation

Rather than requiring a physical press of the reset button before an upload, the Micro is designed in a way that allows it to be reset by software running on a connected computer. The reset is triggered when the Micro's virtual (CDC) serial / COM port is opened at 1200 baud and then closed.

When this happens, the processor will reset, breaking the USB connection to the computer (meaning that the virtual serial / COM port will disappear). After the processor resets, the bootloader starts, remaining active for about 8 seconds. The bootloader can also be initiated by pressing the reset button on the Micro. Note that when the board first powers up, it will jump straight to the user sketch, if present, rather than initiating the bootloader.

Because of the way the Micro handles reset it's best to let the Arduino software try to initiate the reset before uploading, especially if you are in the habit of pressing the reset button before uploading on other boards. If the software can't reset the board you can always start the bootloader by pressing the reset button on the board.

2.6 USB Overcurrent Protection

The Micro has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Micro PCB are 4.8cm and 1.77cm respectively, with the USB connector extending beyond the former dimension. The layout allows for easy placement on a solderless breadboard.

CHAPTER 3

INTRODUCTION TO NEAR FEILD COMMUNICATION (NFC)

3.1 Overview

NFC technology enables simple and safe two-way interactions between electronic devices, allowing consumers to perform contactless transactions, access digital content, and connect electronic devices with a single touch. NFC complements many popular consumer level wireless technologies, by utilizing the key elements in existing standards for contactless card technology. NFC can be compatible with existing contactless card infrastructure and it enables a consumer to utilize one device across different systems.

Extending the capability of contactless card technology, NFC also enables devices to share information at a distance that is less than 4 centimeters with a maximum communication speed of 424 kbps. Users can share business cards, make transactions.

NFC's bidirectional communication ability is ideal for establishing connections with other technologies by the simplicity of touch. For example, if a user wants to connect a mobile device to a stereo system to play music, he can simply touch the device to the stereo's NFC touch-point and the devices will negotiate the best wireless technology to use.

NFC-enabled devices are unique in that they can support three modes of operation:

1) Card emulation

Card emulation mode enables NFC-enabled devices to act like smart cards, allowing users to perform transactions such as purchases, ticketing, and transit access control with just a touch.

In Card Emulation mode, the NFC-enabled device communicates with an external reader much like a traditional contactless smart card. This enables contactless payments and ticketing by NFC-enabled devices without changing existing infrastructure.

Adding NFC to a contactless infrastructure enables two-way communications. For the air transport industry, this could mean updating seat information while boarding, or adding frequent flyer points when making a payment.

2) Peer-to-peer mode

This mode enables two NFC-enabled devices to communicate with each other to exchange information and share files, so that users of NFC-enabled devices can quickly share contact information and other files with a touch. For example, users can share Bluetooth or WiFi link set-up parameters or exchange data such as virtual business cards or digital photos.

3) Reader/Writer

Reader/writer mode enables NFC-enabled devices to read information stored on inexpensive NFC tags which provides a great marketing tool for companies. The device which is implemented in the project uses this mode.

3.2 Operating Specification

NFC operates by following two protocols

1) Near Field Communication – Interface and Protocol (NFCIP-1), which defines an interface and protocol for simple wireless interconnection of closely coupled devices operating at 13.56 MHz.

2) The NFC Data Exchange Format (NDEF) specification defines a message encapsulation format to exchange information, e.g. between an NFC Forum Device and another NFC Forum Device or an NFC Forum Tag.

a) About Near Field Communication – Interface and Protocol (NFCIP-1)

This International Standard defines communication modes for Near Field Communication Interface and Protocol (NFCIP-1) using inductive coupled devices operating at the centre frequency of 13.56 MHz for interconnection of computer peripherals. It also defines both the Active and the Passive communication modes of Near Field Communication Interface and Protocol (NFCIP-1) to realize a communication network using Near Field Communication devices for networked products and also for consumer equipment. This Standard specifies, in particular, modulation schemes, codings, transfer speeds, and frame format of the RF interface, as well as initialization schemes and conditions required for data collision control during initialization. Furthermore, this Standard defines a transport protocol including protocol activation and data exchange methods.

NFCIP-1 Targets and Initiators can implement both the Active and the Passive communication modes.

In the Active communication mode, both the Initiator and the Target use their own RF field to communicate. The Initiator starts the NFCIP-1 transaction. The Target responds to an Initiator command in the Active communication mode by modulating its own RF field.

In the Passive communication mode, the Initiator generates the RF field and starts the transaction. The Target responds to an Initiator command in the Passive communication mode by modulating the Initiators' RF field which is referred to as load modulation.

This Standard specifies requirements for modulation, bit rates and bit coding. In addition it specifies requirements for the start of communication, the end of communication, the bit and byte representation, the framing and error detection, the single device detection, the protocol and parameter selection and the data exchange and de-selection of Near Field Communication Interface and Protocol (NFCIP-1) devices.

Transactions start with device initialization and end with device de-selection. Initiators and Targets exchange commands, responses and data in alternating or half duplex communication.

NFCIP-1 devices are capable to start transactions at bit rates of *fc*/128, *fc*/64 and *fc*/32. Initiators select one of those bit rates to start a transaction and they may change the bit rate using PSL_REQ/PSL_RES commands during a transaction.

The mode (Active or Passive) cannot be changed during one transaction.

3.3 General Protocol flow

The General Protocol flow between NFCIP-1 devices shall be conducted through the following consecutive operations:

1) Any NFCIP-1 device shall be in Target mode initially and not generate an RF field, and shall wait for a command from an Initiator.

2) The NFCIP-1 device may switch to Initiator mode and select either Active or Passive communication mode and transfer speed.

3) Initiators shall test for external RF field presence and shall not activate their RF field if an external RF field is detected. See 8.4.

4) If an external RF field is not detected, the Initiator shall activate its own RF field for the activation of Target.

5) Exchange commands and responses in the same communication mode and the transfer speed.

In this project we will be using Passive Communication mode only.

b) About NFC Data Exchange Format (NDEF) specification

The NFC Data Exchange Format specification defines the NDEF data structure format as well as rules to construct a valid NDEF message as an ordered and unbroken collection of NDEF records. Furthermore, it defines the mechanism for specifying the types of application data encapsulated in NDEF records.

3.4 NDEF Encapsulation Constructs

3.4.1 Message

An NDEF message is composed of one or more NDEF records. The first record in a message is marked with the *MB* (Message Begin) flag set and the last record in the message is marked with the *ME* (Message End) flag set (see figure 1). The minimum message length is one record which is achieved by setting both the *MB* and the *ME* flag in the same record. Note that at least two record chunks are required in order to encode a chunked payload (see section 2.3.3). The maximum number of NDEF records that can be carried in an NDEF message is unbounded.

NDEF messages MUST NOT overlap; that is, the *MB* and the *ME* flags MUST NOT be used to nest NDEF messages. NDEF messages MAY be nested by carrying a full NDEF message as a payload within an NDEF record.

NDEF Message						
R_1 MB=1	…	R_r	…	R_s	…	R_t ME=1

Figure 3.1: Example of an NDEF Message with a Set of Records

The message head is to the left and the tail to the right, with the logical record indices t > s > r > 1. The *MB* (Message Begin) flag is set in the first record (index 1) and the *ME* (Message End) flag is set in the last record (index t).

Actual NDEF records do not carry an index number; the ordering is implicitly given by the order in which the records are serialized. For example, if records are repackaged by an intermediate application, then that application is responsible for ensuring that the order of records is preserved.

3.4.2 Record

A record is the unit for carrying a payload within an NDEF message. Each payload is described by its own set of parameters.

3.4.2 Record Chunks

A record chunk carries a chunk of a payload. Chunked payloads can be used to partition dynamically generated content or very large entities into multiple subsequent record chunks serialized within the same NDEF message.

Chunking is not a mechanism for introducing multiplexing or data streaming into NDEF and it MUST NOT be used for those purposes. It is a mechanism to reduce the need for outbound buffering on the generating side. An NDEF message can contain zero or more chunked payloads. Each chunked payload is encoded as an *initial* record chunk followed by zero or more *middle* record chunks and finally by a *terminating* record chunk. Each record chunk is encoded as an NDEF record using the following encoding rules:

• The *initial* record chunk is an NDEF record with the *CF* (Chunk Flag) flag set. The type of the entire chunked payload MUST be indicated in the TYPE field regardless of whether the PAYLOAD_LENGTH field value is zero or not. The ID field MAY be used to carry an identifier of the entire chunked payload. The PAYLOAD_LENGTH field of this initial record indicates the size of the data carried in the PAYLOAD field of the initial record only, not the entire payload size.

• Each *middle* record chunk is an NDEF record with the *CF* flag set indicating that this record chunk contains the next chunk of data of the same type and with the same identifier as the initial record chunk. The value of the TYPE_LENGTH and the *IL* fields MUST be zero and the TNF (Type Name Format) field value MUST be 0x06 *(Unchanged)* (see section 3.2.6). The PAYLOAD_LENGTH field indicates the size of the data carried in the PAYLOAD field of this single middle record only (see section 2.4.1).

• The *terminating* record chunk is an NDEF record with the *CF* flag cleared, indicating that this record chunk contains the last chunk of data of the same type and with the same identifier as the initial record chunk. As with the middle record chunks, the value of the TYPE_LENGTH and the *IL* fields MUST be zero and the TNF (Type Name Format) field value MUST be 0x06 *(Unchanged)* (see section 3.2.6). The PAYLOAD_LENGTH field indicates the size of the data carried in the PAYLOAD field of this terminating record chunk (see section 2.4.1).

A chunked payload MUST be entirely encapsulated within a single NDEF message. That is, a chunked payload MUST NOT span multiple NDEF messages. As a consequence, neither an initial nor a middle record chunk can have the *ME* (Message End) flag set.

3.5 NDEF Requirements

3.5.1 Message requirements

1) Each NDEF message MUST be exchanged in its entirety.

2) The first record in a message is marked with the MB (Message Begin) flag set.

3) The last record in the message is marked with the ME (Message End) flag set.

4) NDEF messages MUST NOT overlap; that is, the MB and the ME flags MUST NOT be used to nest NDEF messages.

3.5.2 Record chunk requirements

1) Each chunked payload is encoded as an initial record chunk followed by 0 or more middle record chunks and finally by a terminating record chunk.

2) The initial record chunk is an NDEF record with the CF (Chunk Flag) flag set.

3) The type of the entire chunked payload MUST be indicated in the TYPE field of the initial record chunk.

4) The PAYLOAD_LENGTH field of the initial record indicates the size of the data carried in the PAYLOAD field of the initial record only, not the entire payload size.

5) Each middle record chunk is an NDEF record with the CF flag set.

6) For each middle record chunk the value of the TYPE_LENGTH and the IL fields MUST be 0.

7) For each middle record chunk the TNF (Type Name Format) field value MUST be 0x06 (Unchanged).

8) For each middle record chunk, the PAYLOAD_LENGTH field indicates the size of the data carried in the PAYLOAD field of this single record only.

9) The terminating record chunk is an NDEF record with the CF flag cleared.

10) For the terminating record chunk, the value of the TYPE_LENGTH and the IL fields MUST be 0.

11) For the terminating record chunk, the TNF (Type Name Format) field value MUST be 0x06 (Unchanged).

12) For the terminating record chunk, the PAYLOAD_LENGTH field indicates the size of the data carried in the PAYLOAD field of this record only.

13) A chunked payload MUST be entirely encapsulated within a single NDEF message.

14) An initial record chunk MUST NOT have the ME (Message End) flag set.

15) A middle record chunk MUST NOT have the ME (Message End) flag set.

CHAPTER 4

INTRODUCTION TO NFC TAG

The NFC Reader/writer we are using in the project is compatible with Mifare card, which has 1K (1024) bytes of EEPROM memory in it. User can access to these memory using the reader/writer. The card divides the memory into segments called sectors. 1K card has 16 sectors, with each sector have 4 blocks, and each block have 16 bytes of memory. Each sector of memory is protected by two different keys, called A and B. Each key can be programmed to allow access operations such as reading, writing, increasing value blocks, etc. However, 16 bytes of each sector (1 block) are reserved for key A, B and access conditions; and cannot normally be used for user data. Not to forget, the 1st 16 bytes (block 0) contain the serial number of the card (4 bytes) and information about the card manufacturer data and read only. That will end up the actual usable memory for 1K Mifare Classic card to 752 bytes. That is quite some space to store value, names, biodata, time, etc.

4.1 Memory Organization (1K)

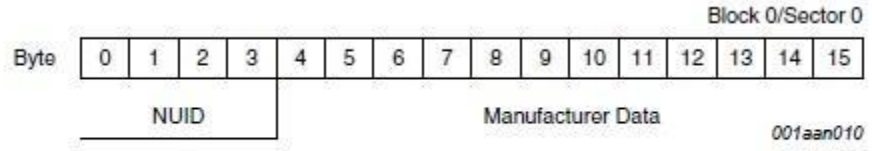
Here is the table showing the memory organization of Mifare Classic 1K Tag.

Sector	Block	Byte Number within a Block 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Description
15	3	Key A						Access Bits				Key B						Sector Trailer 15
	2																	Data
	1																	Data
	0																	Data
14	3	Key A						Access Bits				Key B						Sector Trailer 14
	2																	Data
	1																	Data
	0																	Data
:	:																	
:	:																	
:	:																	
1	3	Key A						Access Bits				Key B						Sector Trailer 1
	2																	Data
	1																	Data
	0																	Data
0	3	Key A						Access Bits				Key B						Sector Trailer 0
	2																	Data
	1																	Data
	0	Manufacturer Data																Manufacturer Block

Figure 4.1 Memory organization of Mifare Classic 1K Transponder/Tag/Card

There are 16 sectors, starting from sector 0, and each sector will have 4 blocks, which start from block 0 to block 3. Each block will have 16 bytes of memory. The access to data on each sector is controlled by the value of last block, which is block 3 (sector trailer) of each sector. Normally the block is being number from 0 to 63, user will need to decide which sector he/she wanted to access and provide the correct Keys and access bit. Block 3 (Sector Trailer) of each sector store 2 keys value and access bits. It is named as Key A and Key B, both are 6 bytes and the Access bits are 4 bytes (actual is 3 bytes).

Block 0 of sector 0 (1st data block of 1st sector) is manufacturer block as is it readable only because it is being programmed by manufacturer during production and is One Time Programmable only.

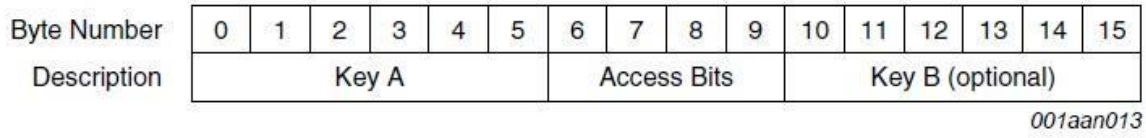


Manufacturer block

Manufacturer block (Block 0 of Sector 0) in Mifare Classic 1K

Therefore, there are 3 user data blocks in each sector, except 2 user data block for sector 0 because of manufacturer block.

Sector trailer is block 3 of each sector, there are 3 portion of data in it. We know each block is 16 bytes, sector trailer too. 6 bytes is allocated for Key A (front), another 6 bytes for Key B (back), middle 4 bytes are for Access bits, total 16 bytes.



Sector trailer

CHAPTER 5

INTRODUCTION TO NFC READER/WRITER MODULE

NFC Reader/Writer is the main component of the NFC device which will be setup at the source and destination stations to do the passenger transactions. In this project we will use The CR038A Mifare Reader/Writer to read information from Mifare card (NFC tag). The CR038 Mifare reader/writer uses 5V TTL UART to communicate with host. The host can be a computer or microcontroller. In this project the host is Arduino Micro. The reader writer only responds to the commands it receives from the micro-controller.

5.1 Operation Protocol

1. Initialize the CR038 device.
2. Activate the antenna for further operation. The tag requires power from device antenna.
3. Request Mifare Standard. CR038 will reply with card type that readable from the antenna. The Mifare card must be on the CR038.
4. Anti-collision loop. The NUID for Mifare card will be read by CR038 and return to host.

5. Select card with the NUID you just get from previous step.
6. Now authentication a particular sector.
7. If authentication is successful, you can now read or write data from/to blocks within the same sector of that Mifare card.

5.2 HARDWARE INTERFACE OF THE READER/WRITER

The UART settings are:

- **Baud rate**: 19200 bps (default)
- **Data**: 8 bits
- **Stop**: 1 bit
- **Parity**: None
- **Flow control**: None

The small flex-cable breakout board allows easy access to CR038. There are only 5 meaningful pins on the breakout board. Pin-out of CR038 breakout board

Pin Description	**Function**	**Pin No.**
positive supply to CR038 board	+ve power, 5VDC	1
Connect to RX of host, example microcontroller's RX pin, 5V TTL	UART's TX, transmitter of CR038	2
Connect to TX of host, example microcontroller's TX pin, 5V TTL	UART's RX, receiver of CR038	3
Ground of the power and signal	-ve power, 0V, GND	4
positive supply to CR038 board	+ve power, 5VDC	5

5.3 SOFTWARE SPECIFICATION

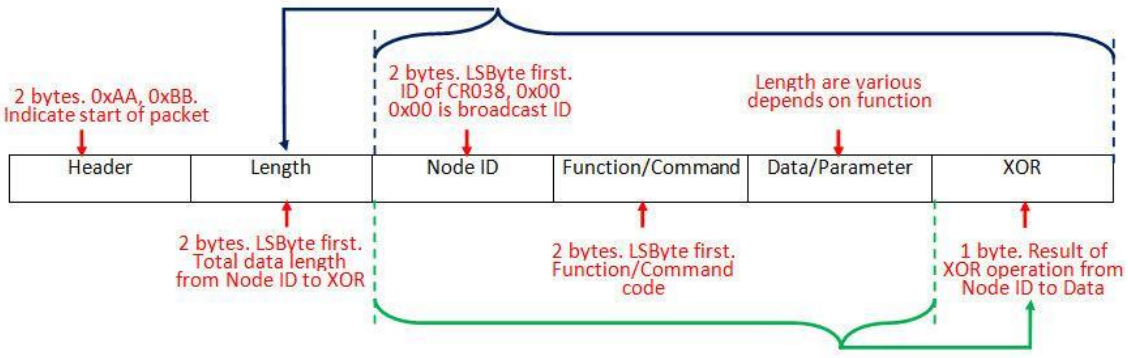
R038 uses packetized UART communication. This means command/functions and results are communicated in several bytes of data with header, length, data, also checksum.

5.3.2 Data Packet

The packet general contains header, followed by length, Node ID (CR038 ID number), function/command Code, the data or parameters, lastly the checksum, it uses XOR. The CR038 will only execute a command/function if and only if it receive whole packet of data correctly. It must comes with 2 bytes of header 0xAA, and 0xBB, follow by 2 bytes of data length, Node ID, Function Code, Data (optional) and result of XOR. The XOR is result (1 byte) from performing exclusive OR operation from Node ID byte until last byte of Data.

5.3.3 Packet format:

XOR	Data	Function Code	Node ID	Length	Header



Packet from Host to CR038, the command:

Remarks	**Data Length (Byte)**	**Data**

Fixed : 0xAA follow by 0xBB.	02	Header
Length of Node ID to DATA, lower byte first.	02	Length
Destination Node ID. This is CR038 ID. Low byte first 00 00: Broadcast ID.	02	Node ID
Function or Command code. Lower byte first.	02	Function/Command Code
Data or parameter needed for function/command. It can be 0 or maximum is 208 bytes. length depends on function.	00~0xD0	Data/Parameter
result from XOR operation each byte from from Node ID to Data.	01	XOR

Packet from CR038 to host, the event :

Remarks	**Data Length (Byte)**	**Data**
Fixed : 0xAA follow by 0xBB.	02	Header
Length of Node ID to DATA, lower byte first.	02	Length
Node ID of CR038, lower byte first.	02	Node ID

Function/Command code received from host. To indicate this packet is reply on the function sent.	02	Function/Command Code
1 byte, function/command result, 0 = Success, Not 0 = fail.	1	Status
Data or parameter return from CR038 based on function/command. It can be 0 or maximum is 208 bytes. length depends on function.	00~D0	Data/Parameter
XOR byte from Node ID to Data	01	XOR

Function/Command Set

1) Initialize Port

Code-0x0101

Function: Set Baud Rate of CR038's UART

Packet Format (Hexadecimal): AA BB 06 00 00 00 01 01 Baud XOR

06 00 = Packet length, 2 bytes, lower byte first. This indicate there are 6 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

01 01 = Function/Command Code for Initialize Port, 2 bytes, lower byte first.

Baud = Parameter for Baud rate. 1 byte.

- 0 = 4800 bps
- 1 = 9600 bps
- 2 = 14400 bps
- 3 = 19200 bps

- 4 = 28800 bps
- 5 = 38400 bps
- 6 = 57600 bps
- 7 = 115200 bps

2) Set LED Status

code: 0x0107

Function: Set LED status on CR038

Packet Format (Hexadecimal): AA BB 06 00 00 00 07 01 LED XOR

06 00 = Packet length, 2 bytes, lower byte first. This indicate there are 6 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

07 01 = Function/Command Code Set LED status, 2 bytes, lower byte first.

LED = Parameter for LED status. 1 byte.

- 0 = Red LED OFF
- 1 to 3 = Red LED ON

3) Set Antenna Status

Code: 0x010c

Function: Set Antenna status on CR038, needed to access (read or write) to Mifare card.

Packet Format (Hexadecimal): AA BB 06 00 00 00 0C 01 Antenna XOR

06 00 = Packet length, 2 bytes, lower byte first. This indicate there are 6 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

0C 01 = Function/Command Code for Set Antenna Status, 2 bytes, lower byte first.

Antenna = Parameter for Antenna status. 1 byte.

- 0 = Antenna OFF
- 1 = Antenna ON

4) Mifare Card Type Request

Code: 0x0201

Function: Read Mifare Card type, the Mifare card must be near to CR038 (best is on top) and Antenna must be activated. This step is needed if you want to access the Mifare card.

Packet Format (Hexadecimal): AA BB 06 00 00 00 01 02 Request XOR

06 00 = Packet length, 2 bytes, lower byte first. This indicates there are 6 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

01 02 = Function/Command Code to request Mifare card type, 2 bytes, lower byte first.

Request = Request Mifare Card type code. 1 byte.

- 0x52 = Request all Type A card in the reading range
- 0x26 = Request all idle card

5) Mifare Anti-collision

Code: 0x0202

Function: Read the NUID of Mifare card/tag/transponder. The Mifare card must be near to CR038 (best is on top). This step is needed if you want to access the Mifare card.

Packet Format (Hexadecimal): AA BB 05 00 00 00 02 02 00

05 00 = Packet length, 2 bytes, lower byte first. This indicate there are 5 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

02 02 = Function/Command Code to perform anti-collision. It is to read the NUID from Mifare card, 2 bytes, lower byte first.

00 = result of exclusive OR operation from Node ID to Function/Command code, in this example.

6) Mifare Select Card

Code: 0x0203

Function: Select a particular Mifare Card/Tag with the NUID. The Mifare card must be near to CR038 (best is on top). This step is needed if you want to access the Mifare card.

Packet Format (Hexadecimal): AA BB 09 00 00 00 03 02 XX XX XX XX XOR

09 00 = Packet length, 2 bytes, lower byte first. This indicate there are 9 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

03 02 = Function/Command Code to select particular with NUID. It will activate the particular Mifare card, 2 bytes, lower byte first.

XX XX XX XX = 4 bytes of NUID you obtain from Mifare Anti-collision, lower byte 1st.

7) Mifare Halt

Code: 0x0204

Function: To place the selected Mifare Card/Tag in halt mode, to deactivate the card. Once the card is halted, you will need to start from Mifare Request again to activate the card. The Mifare card must be near to CR038 (best is on top).

Packet Format (Hexadecimal): AA BB 05 00 00 00 04 02 06

AA BB = Header, 2 bytes.

05 00 = Packet length, 2 bytes, lower byte first. This indicate there are 5 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

04 02 = Function/Command Code to perform halt (deactivate) on selected card, 2 bytes, lower byte first.

06 = result of exclusive OR operation from Node ID to Function/Command code, in this example.

8) Mifare Authentication2

Code: 0x0207

Function: To perform authentication with selected Mifare card for memory access. You can select secret Key A or B as authentication password. If a fresh new card from factory, Key A is use for authentication purpose and is hexadecimal FF, FF, FF, FF, FF, FF (6 bytes of 0xFF). You will need to select the particular block for this purpose. It will authenticate for whole sector. Example if you choose the block to be 10, you are actually authenticate for whole sector 2. Access to block 8, 9, 10 and 11(block 11 is sector trailer) will be allowed if the authentication is successful. The selected Mifare card must be near to CR038 (best is on top). There are several steps before you can do authentication, please check the flow.

Packet Format (Hexadecimal): AA BB 0D 00 00 00 07 02 Mode Block Key XOR

0D 00 = Packet length, 2 bytes, lower byte first. This indicate there are 13 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

07 02 = Function/Command Code to perform authentication process with selected card, 2 bytes, lower byte first.

Mode = Authentication mode, to select using Key A or Key B for authentication purpose. 0x60 is to use Key A, 0x61 is to use Key B.

Block = Authentication block, with the same sector, it will match the sector trailer.

Key = Authentication Key, 6 bytes, lower byte 1st. This 6 bytes key must match the Key (A or B) in the sector trailer on the sector chosen.

9) Mifare Read

Code: 0x0208

Function: To read the data of certain block (16 bytes) within a sector in the selected Mifare card, authentication must be successful. The selected Mifare card must be near to CR038 (best is on top). There are several steps before you can do authentication, please check the flow.

Packet Format (Hexadecimal): AA BB 06 00 00 00 08 02 Block XOR

06 00 = Packet length, 2 bytes, lower byte first. This indicate there are 6 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

08 02 = Function/Command Code to read a block of data from selected card, 2 bytes, lower byte first.

Block = Data block which you want to read.

10) Mifare Write

Code: 0x0209

Function: To write data into certain block (16 bytes) within a sector in the selected Mifare card, authentication must be successful. The selected Mifare card must be near to CR038 (best is on top). There are several steps before you can do authentication, please check the flow.

Packet Format (Hexadecimal): AA BB 16 00 00 00 09 02 Block D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_9 D_A D_B D_C D_D D_E D_F XOR

16 00 = Packet length, 2 bytes, lower byte first. This indicate there are 22 bytes of data from Node ID to XOR.

00 00 = Node ID, Serial number of CR038, 2 bytes, lower byte first. 00 00 mean broadcast, it works for any ID.

09 02 = Function/Command Code to write data to a block of memory in selected card, 2 bytes, lower byte first.

Block = Data block which you want to read.

16 bytes of data to write = From D_0 to D_F. This is the 16 bytes of data needed for the CR038 to write into the Mifare card.

CHAPTER 6

INTRODUCTION TO PROCESING

Processing is a programming language, development environment, and online community. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. Initially created to serve as a software sketchbook and to teach computer programming fundamentals within a visual context, Processing evolved into a development tool for professionals. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production

6.1 Overview

The Processing Development Environment (PDE) makes it easy to write Processing programs. Programs are written in the Text Editor and started by pressing the Run button. In Processing, a computer program is called a *sketch*. Sketches are stored in the *Sketchbook*, which is a folder on your computer. It's easy to open the sketches by clicking on the Open button.

Sketches can draw two- and three-dimensional graphics. The default renderer is for drawing two-dimensional graphics. The P3D renderer makes it possible to draw three-dimensional graphics, which includes controlling the camera, lighting, and materials. The P2D renderer is a fast, but last accurate renderer for drawing two-dimensional graphics. Both the P2D and P3D renderers are accelerated if your computer has an OpenGL compatible graphics card.

The capabilities of Processing are extended with *Libraries* and *Tools*. Libraries make it possible for sketches to do things beyond the *core* Processing code. There are hundreds of libraries contributed by the Processing community that can be added to your sketches to enable new things like playing sounds, doing computer vision, and working with advanced 3D geometry. Tools extend the PDE to help make creating sketches easier by providing interfaces for tasks like selecting colors.

Processing has different *programming modes* to make it possible to deploy sketches on different platforms and program in different ways. The current default programming modes are *Java* and *Experimental*. Other programming modes, such as *JavaScript* and *Android*, are added by selecting "Add Mode..." from the menu in the upper-right corner of the PDE.

6.2 Processing Development Environment (PDE) :

The **Processing Development Environment (PDE)** consists of a simple text editor for writing code, a message area, a text console, tabs for managing files, a toolbar with buttons for common actions, and a series of menus. The menus options change from mode to mode. The default Java mode is documented here.

Figure 6.1 Processing Development Environment

Programs written using Processing are called sketches. These sketches are written in the text editor. It has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by Processing sketches including complete error messages and text output from sketches with the print() and println() functions.

The buttons on the toolbar can run and stop programs, create a new sketch, open, save, and export:

Run

Runs the sketch. In Java mode, it compiles the code and opens a new display window.

Stop

Terminates a running sketch.

New

Creates a new sketch (project) in the current window. To create a new sketch in its own window, use File → New.

Open

Provides a menu with options to open files from anywhere on your computer (Open...), from the Example Menu (Examples...), or one of the programs in the Sketchbook. Opening a sketch from the toolbar will replace the sketch in the current window. To open a sketch in a new window, use File → Open.

Save

Saves the current sketch to its current location. If you want to give the sketch a different name, select “Save As” from the File menu.

Export

In Java mode, it exports the current sketch as a Java application and the folder containing the files is opened. (Note: Exporting a sketch will delete the previous contents of the export folder, unless this preference is unchecked in the Preferences.)

Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive which means only those items relevant to the work currently being carried out are available.

File

- *New (Ctrl+N)*
  Creates a new sketch in a new window, named as the current date is the format "sketch_YYMMDDa".

- *Open (Ctrl+O)*
  Open a sketch in a new window.

- *Sketchbook*
  Open a sketch from the sketchbook folder.

- *Recent*
  Open a recently closed sketch.

- *Examples*
  Open one of the examples included with Processing.

- *Close (Ctrl+W)*
  Close the sketch in the frontmost window. If this is the last sketch that's open, you will be prompted whether you would like to quit. To avoid the prompt, use Quit instead of Close when you want to exit the application.

- *Save (Ctrl+S)*
  Saves the open sketch in it's current state.

- *Save as... (Shift+Ctrl+S)*
  Saves the currently open sketch, with the option of giving it a different name. Does not replace the previous version of the sketch.

- *Export (Ctrl+E)*
  In Java mode, exports a Java application as an executable file and opens the folder containing the exported files.

- *Page Setup (Shift+Ctrl+P)*
  Define page settings for printing.

- *Print (Ctrl+P)*
  Prints the code inside the text editor.

- *Preferences (Ctrl+,)*
  Change some of the ways Processing works. (This item is located in the Processing menu on Mac OS X.)

- *Quit (Ctrl+Q)*
  Exits the Processing Environment and closes all Processing windows. (This item is located in the Processing menu on Mac OS X.)

Edit

- *Undo (Ctrl+Z)*
  Reverses the last command or the last entry typed. Cancel the Undo command by choosing Edit » Redo.

- *Redo (Shift+Ctrl+Z)*
  Reverses the action of the last Undo command. This option is only available, if there has already been an Undo action.

- *Cut (Ctrl+X)*
  Removes and copies selected text to the clipboard (an off-screen text buffer).

- *Copy (Ctrl+C)*
  Copies selected text to the clipboard.

- *Copy as HTML (Shift+Ctrl+C)*
  Formats code as HTML, the same way it appears in the Processing environment and copies it to the clipboard so it can be pasted somewhere else.

- *Paste (Ctrl+V)*
  Inserts the contents of the clipboard at the location of the cursor, and replaces any selected text.

- *Select All (Ctrl+A)*
  Selects all of the text in the file which is currently open in the text editor.

- *Auto Format (Ctrl-T)*
  Attempts to format the code into a more human-readable layout. Auto Format was previously called *Beautify*.

- *Comment/Uncomment (Ctrl+/)*
  Comments the selected text. If the selected text is already commented, it uncomments it.

- *Increase Indent (Ctrl+])*
  Indents the selected text two spaces.

- *Decrease Indent (Ctrl+[)*
  If the text is indented, removes two spaces from the indent.

- *Find... (Ctrl+F)*
  Finds an occurance of a text string within the file open in the text editor and gives the option to replace it with a different text.

- *Find Next (Ctrl+G)*
  Finds the next occurance of a text string within the file open in the text editor.

- *Find Previous (Shift+Ctrl+G)*
  Finds the previous occurance of a text string within the file open in the text editor.

Sketch

- *Run (Ctrl+R)*
  Runs the code (compiles the code, opens the display window, and runs the sketch inside)

- *Present (Ctrl+Shift+R)*
  Runs the code in the center of the screen with a solid-color background. Click the "stop" button in the lower left to exit the presentation.

- *Stop*
  If the code is running, stops the execution. Programs written without using the draw() function are stopped automatically after they draw.

- *Import Library*
  Adds the necessary import statements to the top of the current sketch. For example, selecting Sketch » Import Library » video adds the statement "import processing.video.*;" to the top of the file. These import statements are necessary for using Libraries.

- *Show Sketch Folder*
  Opens the folder for the current sketch.

- *Add File*
  Opens a file navigator. Select an image, font, or other media files to add it to the sketch's "data" folder.

Tools

- *Create Font...*
  Converts fonts into the Processing font format and adds to the current sketch. Opens a dialog box which give options for setting the font, it's size, if it is anti-aliased, and if all characters should be generated. If the "All Characters" options is selected, non-English characters such as ü and Å are generated, but the font file is larger in size. The amount of memory required for the font is also determined by the size selected. Processing fonts are textures, so larger fonts require more image data.

- *Color Selector*
  Interface for selecting colors.

- *Archive Sketch*
  Archives a copy of the current sketch in .zip format. The archive is placed in the same folder as the sketch.

- *Movie Maker*
  Creates a QuickTime movie from a sequence of images. Options include setting the size, frame rate, and compression, as well as an audio file.

Help

- *About Processing*
  Opens a concise information panel about the software. (This item is located in the Processing menu on Mac OS X.)

- *Environment*
  Opens the reference for the Processing Development Environment (this page) in the default web browser.

- *Reference*
  Opens the reference in the default web browser. Includes reference for the language, programming environment, libraries, and a language comparison.

- *Find in Reference (Ctrl+Shift+F)*
  Select a word in your sketch and select "Find in Reference" to open its reference HTML page.

- *Getting Started*
  Opens a tutorial on getting started with Processing.

- *Troubleshooting*
  Opens the troubleshooting information in the default web browser.

- *Frequently Asked Questions*
  Answers to some basic question about the Processing project.

- *Visit Processing.org*
  Opens default web browser to the Processing.org homepage.

6.3 PREFERENCES

The Processing Development Environment (PDE) is highly configurable. The most common preferences can be modified in the Preferences window, located in the File menu on Windows and Linux and in the Processing menu on Mac Os X. The full list of preferences are stored in the "preferences.txt" file. This file can be opened and edited directly only when Processing is not running. You can find the location of this file on your computer by reading the bottom-left corner of the Prefereces window.

- *Sketchbook location*
  Any folder can be used as the Sketchbook. Input a new location or select "Browse" to set the folder you want to use.

- *Editor font size*
  Sets the font size of the code in the text editor. Restart Processing after making this change.

- *Use smooth text in editor window*
  By default, the text in the editor is aliased. When checked, the editor switches to an anti-aliased (smoothed) font. Restart Processing after making this change.

- *Increase maximum available memory*
  Allocates more RAM to Processing sketches when they run. Sketches that use media files (images, audio, etc.) sometimes require more RAM. Increase the amount of RAM if a sketch is throwing Out of Memory Errors.

- *Delete previous folder on export*
  When checked (default behavior), Processing deletes the complete export folder before re-creating it and adding the new media.

- *Check for updates on startup*
  When checked (default behavior), you'll be informed of new Processing software releases as they become available through a small dialog box that opens as Processing starts.

6.4 SKETCHES AND SKETCHBOOK

All Processing projects are called sketches. Each sketch has its own folder. The main file for each sketch has the same name as the folder and is found inside. For example, if the sketch is named "Sketch_123", the folder for the sketch will be called "Sketch_123" and the main file will be called "Sketch_123.pde". The PDE file extension is an acronym for the Processing Development Environment.

Processing sketches can be stored anywhere on your computer, but by default they are stored in the sketchbook, which will be in different places on your computer or network depending if you use PC, Mac, or Linux and how the preferences are set. To locate this folder, select the "Preferences" option from the File menu (or from the "Processing" menu on the Mac) and look for the "Sketchbook location".

A sketch folder sometimes contains other folders for media files and other code. When a font or image is added to a sketch by selecting "Add File..." from the Sketch menu, a "data" folder is created. Files may also be added to your Processing sketch by dragging them into the text editor. Image and sound files dragged into the application window will automatically be added to the current sketch's "data" folder. All images, fonts, sounds, and other data files loaded in the sketch must be in this folder.

6.5 Tabs, Multiple Files, and Classes

It can be inconvenient to write a long program within a single file. When Processing sketches grow to hundreds or thousands of lines, breaking them into modular units helps manage the different parts. Processing manages files with the Sketchbook and each sketch can have multiple files that are managed with tabs.

The arrow button to the right of the tabs in the Processing Development Environment is used to manage these files. Click this button to reveal options to create a new tab, rename the current tab, and delete the current tab. If a project has more than one tab, they can also be hidden and revealed. Hiding a tab temporarily removes that code from the sketch (it will not be compiled with the sketch when you press Run).

Tabs are intended for more advanced users, and for this reason, the menu that controls the tabs is intentionally made less prominent.

6.6 Programming Modes

Processing has different *programming modes* to make it possible to deploy sketches on different platforms and program in different ways. The current default programming modes are *Java* and *Experimental*. Other programming modes, such as *JavaScript* and *Android*, are added by selecting "Add Mode..." from the menu in the upper-right corner of the PDE.

6.7 Java Mode

This mode makes it possible to write short programs to draw to the screen, but also enables complex Java programs as well. It's can be used simply by beginners, but it scales to professional Java software development. Sketches written in this mode can be exported as Java Applications to run on Linux, Mac OS X, and Windows operating systems.

6.8 Experimental Mode

This is a prototype of a potential future version of the Processing Development Environment. It includes features to check for errors in the code while it's written, to follow variables as they change, to debug a program with break points, and more.

6.9 JavaScript Mode

Sketches written in this mode can be exported to run inside web browsers using HTML5 and WebGL. This mode is documented on the JavaScript page of the Processing Wiki. To add this mode, click on the mode button in the upper-right corner of the PDE and select "Add **Mode...**"

6.10 Android Mode

Sketches written in this mode can be exported to run on Android phones and tablets. This mode is documented on the Processing for Android page of the Processing Wiki. To add this mode, click on the mode button in the upper-right corner of the PDE and select "Add Mode..."

6.11 Adding Libraries, Tools, and Modes

Processing 2.0 includes a set of new features to make it easier to install, update, and remove Libraries, Tools, and Modes.

Add contributed libraries by selecting "Add Library..." from the "Import Library..." submenu within the Sketch menu. Not all available libraries have been converted to show up in "Add Library...". If a library isn't there, it will need to be installed manually. Follow the How to Install a Contributed Library instructions on the Processing Wiki for more information.

Add contributed tools by selecting "Add Tool..." from the Tools menu to select a Tool to download.

Add contributed modes by selecting "Add Mode..." from the Mode menu in the upper-right corner of the PDE.

CHAPTER 7

HARDWARE DESIGN

The hardware used in the prototype consists of an Arduino Micro board, a NFC/RFID 13.56MHz module, Mifare/transponder tag and a computer which functions as a central server of the ticketing system.

7.1 Microcontroller Board: The Arduino Micro is a microcontroller board based on the ATmega32u4, developed in conjunction with Adafruit. It has 20 digital input/output pins, a 16 MHz crystal oscillator, a micro USB connection, an ICSP header, and a reset button. The UART port of Arduino Micro is used to communicate with NFC/RFID reader module. And the USB2.0 port is used to communicate with the computer based server. Both communications use speed of 19200 baud/sec. The board will take passenger count for each user and give control input for turnstile. All entry, exit and recharge point have a microcontroller board.

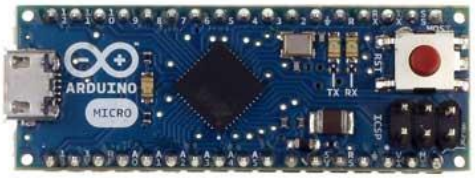


Figure 7.1 Arduino Micro

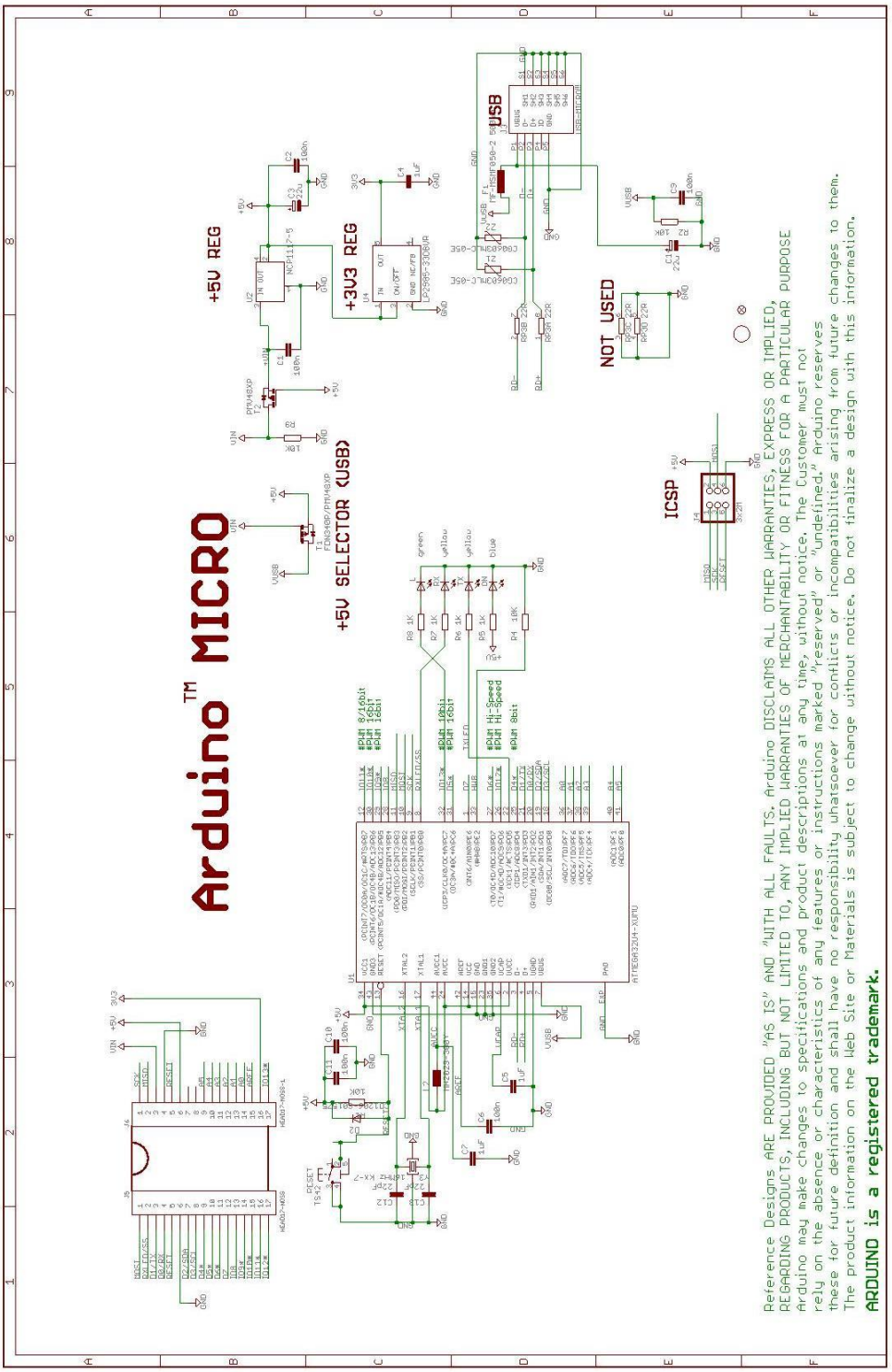


Figure 7.2 Arduino Micro: Internal Block Diagram

7.2 NFC/RFID module: The CR038A Mifare Reader/Writer is a device to read and write information from/to Mifare card, Classic 1K or 4K. The module reads NUID of NFC tag/NFC phone on command from Arduino Micro board. The module has its own data frame format which must be followed for any operation. It communicates via UART at default Baud rate of 19200.[6] Each microcontroller board has its dedicated NFC/RFID module

.

Figure 7.3 NFC/RFID module

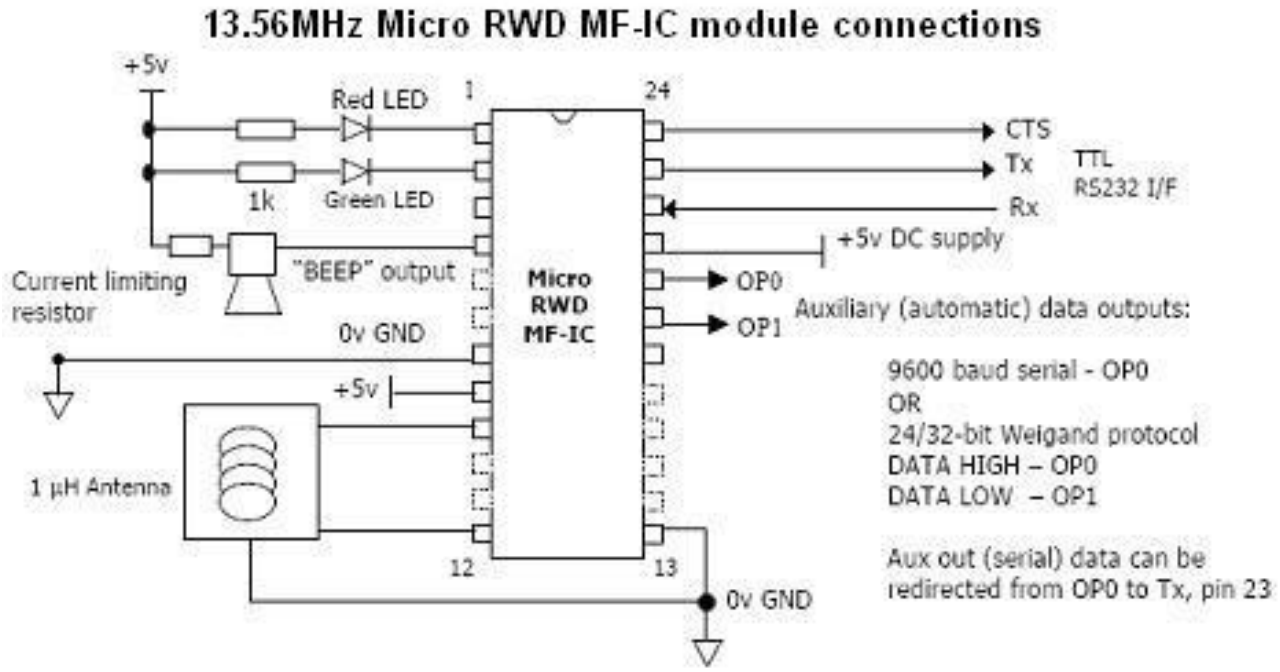


Figure 7.4 NFC Reader: Internal Block Diagram

7.3 MIFARE Classic 1K Tag: The MIFARE classic 1k tag, i.e. MF1S503x chip consists of a 1 kB EEPROM, RF interface and Digital Control Unit. Energy and data are transferred via an antenna consisting of a coil with a small number of turns which is directly connected to the MF1S503x. No further external components are necessary. This is the part of system which will be with the user.

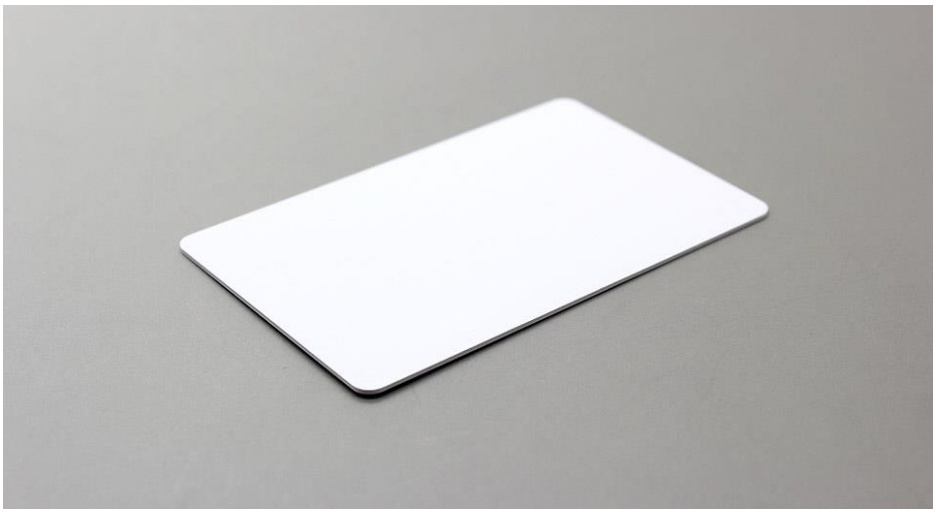


Figure 7.5: Mifare Classic 1k Tag

7.4 Central Server: The central server will store data and status of all users. The central server is connected to all the microcontrollers of the system.

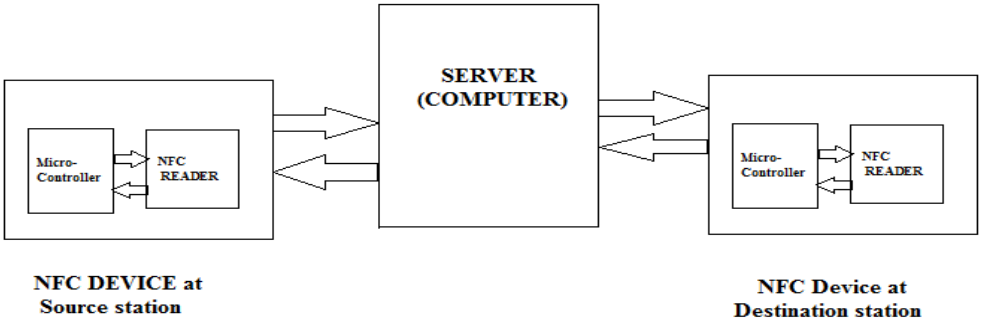


Fig. 7.6: Block Diagram of the proposed system

7.5 LCD

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

A Read/Write (R/W) pin that selects reading mode or writing mode

An Enable pin that enables writing to the registers

8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

There's also a display constrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

16-pin LCD, Pin 15 Led+ and Pin 16 is LED-

Pin No.	Name	Function
1	V_{ss}	Ground
2	V_{dd}	+ve supply
3	V_{ee}	Contrast
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Data bit 0
8	D1	Data bit 1
9	D2	Data bit 2
10	D3	Data bit 3
11	D4	Data bit 4
12	D5	Data bit 5
13	D6	Data bit 6
14	D7	Data bit 7

Source: Everyday Practical Electronics, 1997

Figure 7.7 LCD Pin Layout & Description

INTERFACE WITH ARDUINO

Hardware Required

Arduino Board

LCD Screen (compatible with Hitachi HD44780 driver)

pin headers to solder to the LCD display pins

10k Potentiometer

breadboard

hook-up wire

Circuit

Before wiring the LCD screen to your Arduino we suggest to solder a pin header strip to the 14 (or 16) pin count connector of the LCD screen, as you can see in the image above. To wire your LCD screen to your Arduino, connect the following pins:

LCD RS pin to digital pin 12

LCD Enable pin to digital pin 11

LCD D4 pin to digital pin 5

LCD D5 pin to digital pin 4

LCD D6 pin to digital pin 3

LCD D7 pin to digital pin 2

Additionally, wire a 10K pot to +5V and GND, with it's wiper (output) to LCD screens VO pin (pin3).

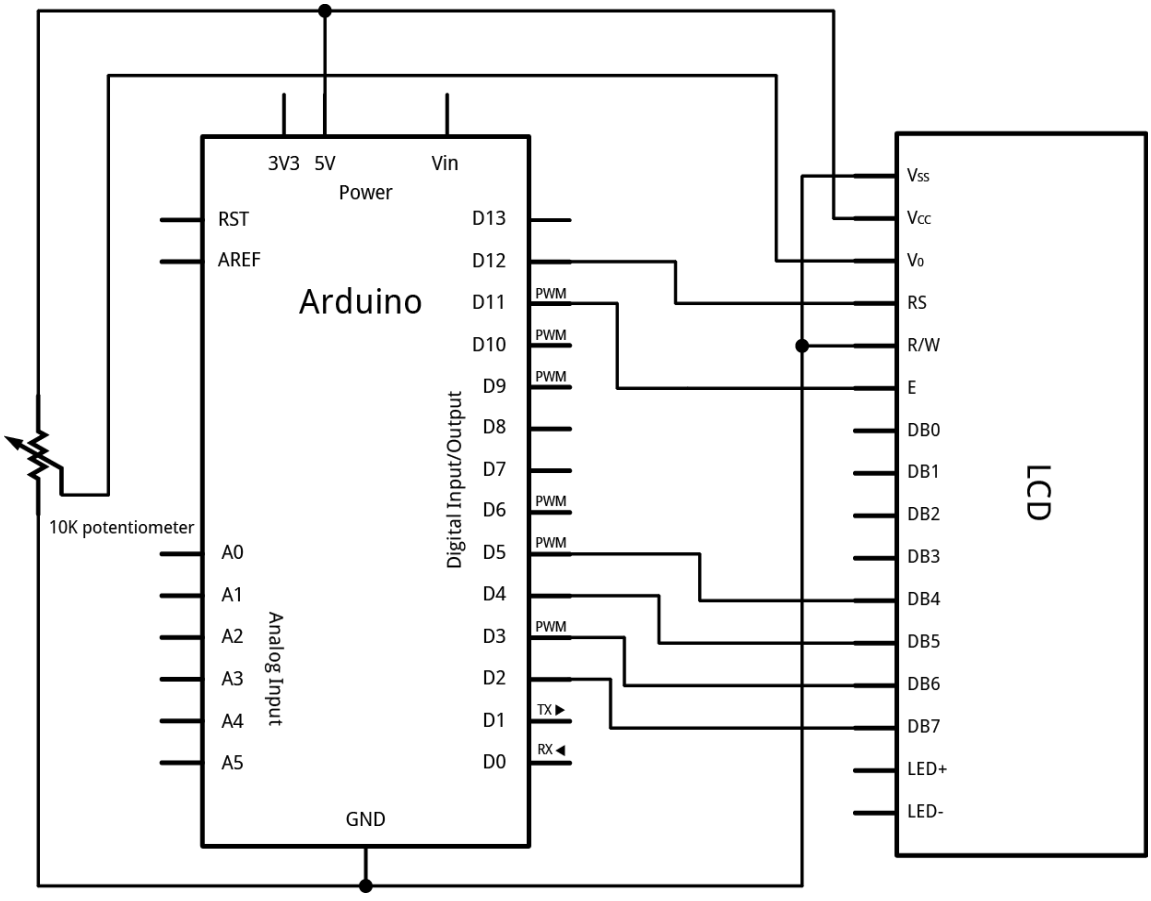


Figure 7.8: LCD Interfacing

7.6 SYSYEM DESCRIPTION

The project consists of an nfc based device which is kept at the station and an nfc tag which the user carries. The tag used by the user is the unique identification card of the user. The user taps the tag at the device kept at the station and the device reads the unique id, and send the information like the UID, the station code where the user boarded and The number of passenger, these details are sent to the server and the server maintains a record of the user , that the user is travelling. Once the user completes the journey and he want an exit from the station the user has to tap at the RFID device then only the exit doors open and the user can exit.

When the user taps the tag at the destination station based on the two stations the appropriate amount is deducted from the users account.

CHAPTER 8

ARDUINO SOFTWARE IMPLEMENTATION

8.1 Introduction:

The code for Arduino Micro is developed in Arduino IDE. Arduino IDE is an open source software to develop and upload code for Arduino boards. The environment is written in Java and based on Processing, avr-gcc, and other open source software.

8.2 Code Flow:

The microcontroller commands NFC/RFID module to continuously scan for NFC tag/NFC phone in vicinity. If the NFC tag/NFC phone is present in range, the microcontroller reads its NUID through NFC/RFID module. The NUID (Number Unique Identifier) is sent to central server with a header and some other data. This data includes details about station, type of point (entry, exit or recharge), number of passengers (only at entry point) and recharge value (only at recharge point). The central server acts according to data received. It updates the user account and sends a positive acknowledgement and number of passengers (only at exit point). If any error occurs, the central server will send negative acknowledgement. Based on acknowledgement received microcontroller gives or denies access to user. The user is identified by the NFC device and the device asks the user to enter the number of passengers who want to gain access into the station. The device sends the NUID, the station code and the number of passengers to the server. Based on this data the necessary changes are made in the users account in the server. Now the system (server) knows that the user is travelling. When the user alights at the destination station and taps the NFC tag/NFC enabled mobile phone at the exit NFC device, the device sends the NUID and the station code to the server. Based on the data provided by the server, the device knows that the user is trying to exit the station and the number of passengers who were given access to the station. Based on this information appropriate amount is deducted from the users account. The device commands the turnstile gates to allow exit to the passengers.

8.3 Flow Chart:

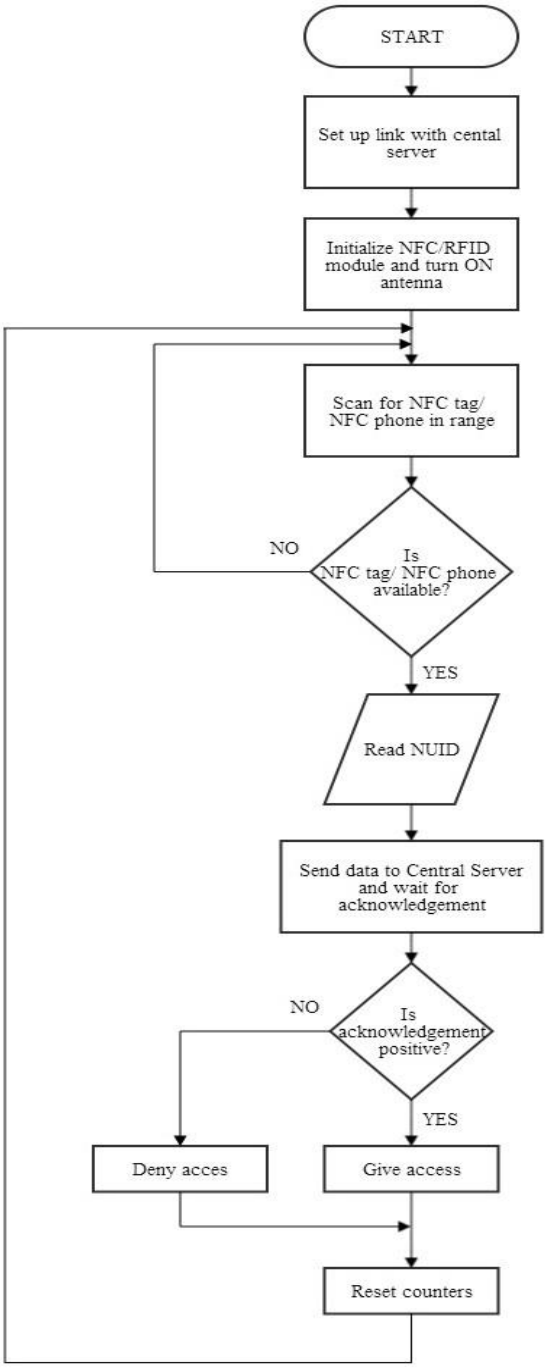


Figure 8.1 Flow chart of Arduino Micro

8.4 Code Explanation:

1) Initialize controller for the given station and for one of the modes amongst entry, exit, recharge

2) Wait for server to send a response

3) When server is connected and ready, initialize the NFC reader/writer

4) After reader/writer is initialized turn on its Antenna

5) Now, wait till reception of card number

6) When, card number is received and if controller is initialized as entry module send card number, station number, number of passengers to server. If server allows access then give access else no access.

7) If controller is initialized as exit module, send card number to sever. If server allows then grant exit.

8) If controller is initialized in recharge mode. Give card number and recharge amount to the server

9) Go back to 5

CHAPTER 9

SERVER SOFTWARE IMPLEMENTATION

9.1 Introduction:

In this project we have used “processing” software to emulate a server on a computer which will be connected to the Arduino using a USB port. The server is responsible for maintaining and processing all information for any given user in the system. The job of the server is explained in this chapter.

9.2 Flow Chart:

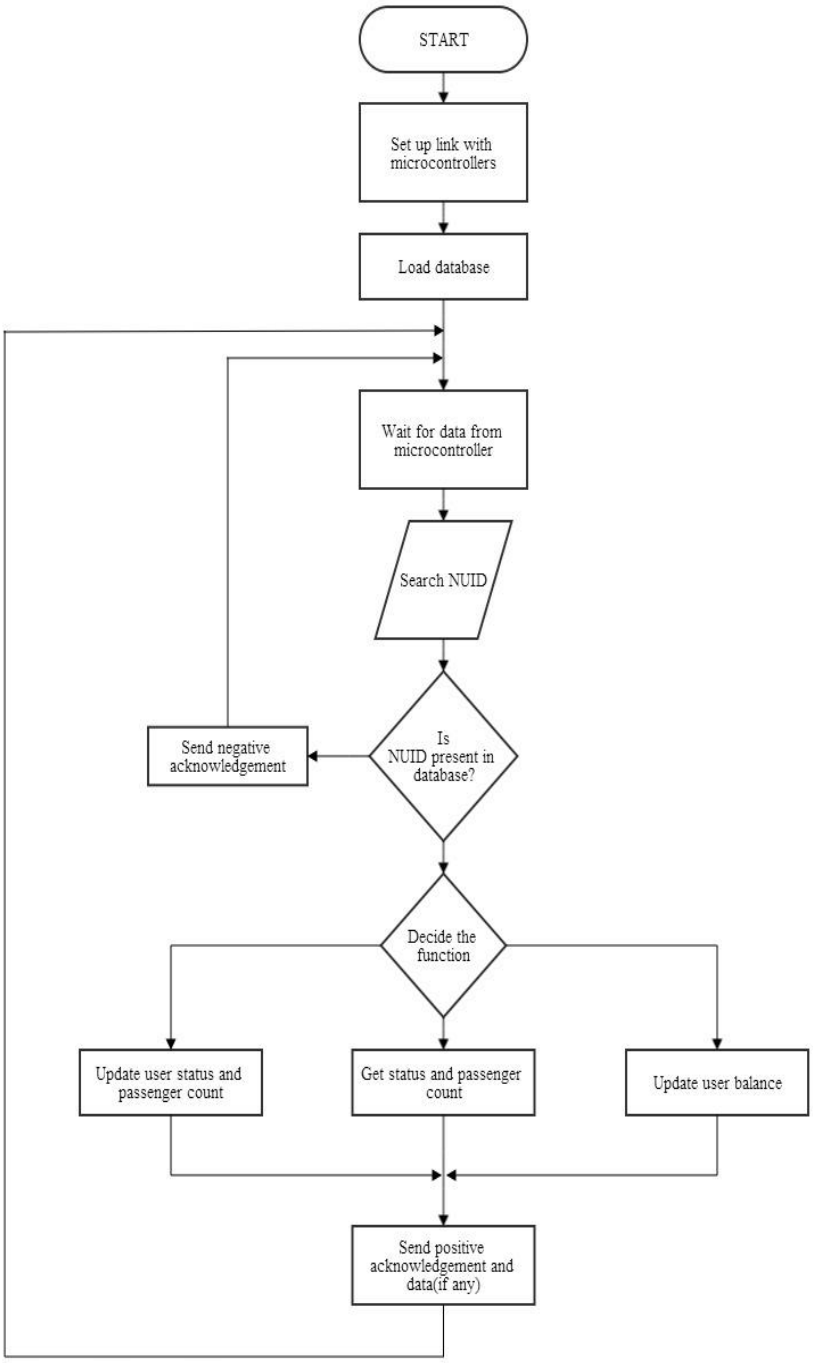


Figure 9.1 Flow Chart of Server Implementation

9.3 Code Explanation:

1) Set-up link with mirco-controller and Load database

2) Wait till data is received from micro-controller

3) Search the User ID received in the database, if not found then send negative acknowledgement

4) Depending on the information in database and data received, decide if user should be given access and make appropriate balance changes.

5) Go back to 2

CHAPTER 10

RESULTS AND DISCUSSIONS

10.1 ADVANTAGES OF PROPOSED SYSTEM

The NFC based ticketing system is a very convenient way of gaining access to the Metrorail system, it saves a lot of user's time which was wasted in getting a ticket as the access to the station is provided at the touch of a tap..The NFC based system is a secure way of getting the ticket as the range of operation of the device is just a few centimetres, no unauthorized person can hack the system. Moreover, even users with NFC enabled phones can use this system with complete security.[8] The hacker needs to be very close to the device if he has to gain unauthorized access. This adds more security to the system.

Unlike Bluetooth and other wireless systems, the data transmission link is established and at the same time the data is transmitted, so the user does not have to wait for long time for the data exchange to take place. The NFC cards can be used again and again. This saves a lot of paper which was previously used to print tickets. The NFC tags are available at very less cost and hence can be provided at nominal price by the Metrorail system authority and NFC enabled phones are not required by the user to use the NFC based ticketing system.

10.2 FUTURE SCOPE

The device's use can also be extended to do NFC based transaction at various places like colleges, buses, amusement parks, corporate offices, industries, etc. Thus requiring less paper work and saving the users valuable time. This can be done easily by just changing the program written in the server. This makes the system versatile.

10.3LIMITATIONS

The Initial set up cost of the proposed system will be higher as compared to traditional ticketing system. If NFC Tag is lost or stolen it can be misused. Also, the tag may not be read if the tag is not aligned properly at the NFC device or if the tag is not tapped properly at the device.

10.4 CONCLUSIONS

Although, the proposed system will have a high set up cost, in the long run it will be beneficial for the Indian Metrorail. Since the maintenance cost is very less due to absence of human ticket vendors, this ticketing system will quickly recover its set up cost. Also it will be very convenient for the passengers and solve the problem of making the passengers stand in large queues and thus increase the overall efficiency of Metrorail travel. Moreover, this system has future scope of being capable of extended to general transactions as stated in the advantages which will make it even more useful.

Appendix A

Paper Publication

International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 4 (April 2014), PP.20-24

Design of an Automatic Fare Collection System Using Near Field Communication with Focus on Indian Metrorail

Ajay Gore[1], Nirvedh Meshram[2], Sumit Gadi [3], Rahul Raghatate[4]
[1,2,3,4] (Electrical Engineering Department, Veermata Jijabai Technological Institute,Mumbai, India)

Abstract:- Traditional ticketing system for railways involves serpentine queues for passengers and requires constant human processing. Attempts to fix this problem include coupon books for passengers and more recently smart cards for standard railways. In standard railway, the purpose of the smartcard is to automatically print a ticket for the passenger. However, if a smart card system is to be implemented for Metrorail the purpose of this smartcard will be to give the passenger access to the station by commanding the turnstile to rotate(or a flap gate to open) and then again enable the passenger to exit at the destination station. Building on this idea, we propose a more advanced automatic ticketing system using Near Field Communication keeping in mind the specific needs and requirements of Metrorail systems in India and also the convenience of the passengers. The prototype of this system is implemented and described in this paper.

Keywords:- Ticketing system, human processing, smartcard, Metrorail, automatic ticketing, Near Field Communication, RFID, automatic fare collection

I. INTRODUCTION

NFC(Near Field Communication) technology enables simple and safe two-way interactions between electronic devices, allowing consumers to perform contactless transactions, access digital content, and connect electronic devices with a single touch.[1] (NFCIP-1) Standard specifies the interface and protocol for simple wireless communication between close coupled devices. These Near Field Communication (NFC) devices communicate with bit rates of 106, 212, and 424 kbit/s.[2]

Another option is to use a Bluetooth for communication. However, Bluetooth based applications have the additional procedure of requiring the users to pair their phones, which will be time consuming and additional burden for the system server. With NFC it is possible to skip this step. Also, the users have the dual option of using NFC tags or NFC enabled mobile phones. If NFC enabled mobile phone is used it acts as a NFC tag emulator.

Some important conditions to be met are NFC Data Exchange Format (NDEF) [2],[3], Record Type Definition (RTD) [4], and Type Name Format (TNF) a for the operation of all NFC applications. They show the type of an NFC message, record, and format, respectively.

The proposed system consists of an NFC based device which is kept at the station and an NFC tag/ NFC enabled mobile phone which the user carries, the tag/mobile phone used by the user has a unique identification number of the user. The system also allows more than one passenger to pass through the turnstile via one tag/mobile phone if such a request is made.

II. METHODOLOGY

The methodology consist of hardware interfacing of the system and the software implementation

A. Hardware Description and Interfacing

The hardware used in the prototype consists of an Arduino Micro board, a NFC/RFID 13.56MHz module, Mifare/transponder tag and a computer which functions as a central server of the ticketing system.

1) Microcontroller Board: The Arduino Micro is a microcontroller board based on the ATmega32u4, developed in conjunction with Adafruit. It has 20 digital input/output pins, a 16 MHz crystal oscillator, a micro USB connection, an ICSP header, and a reset button.[5] The UART port of Arduino Micro is used to communicate with NFC/RFID reader module. And the USB2.0 port is used to communicate with the computer based server. Both communications use speed of 19200 baud/sec. The board will take passenger count for each user and give control input for turnstile. All entry, exit and recharge point have a microcontroller board.

2) NFC/RFID module: The CR038A Mifare Reader/Writer is a device to read and write information from/to Mifare card, Classic 1K or 4K. The module reads NUID of NFC tag/NFC phone on command from Arduino Micro board. The module has its own data frame format which must be followed for any operation. It communicates via UART at default Baud rate of 19200.[6] Each microcontroller board has its dedicated NFC/RFID module.

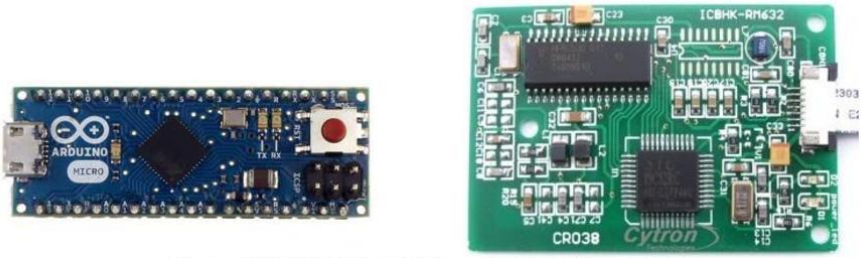


Fig. 1: ARDUINO BOARD [5] and NFC/RFID module [6]

3) MIFARE Classic 1K Tag: The MIFARE classic 1k tag, i.e. MF1S503x chip consists of a 1 kB EEPROM, RF interface and Digital Control Unit. Energy and data are transferred via an antenna consisting of a coil with a small number of turns which is directly connected to the MF1S503x. No further external components are necessary. This is the part of system which will be with the user.

4) Central Server: The central server will store data and status of all users. The central server is connected to all the microcontrollers of the system.

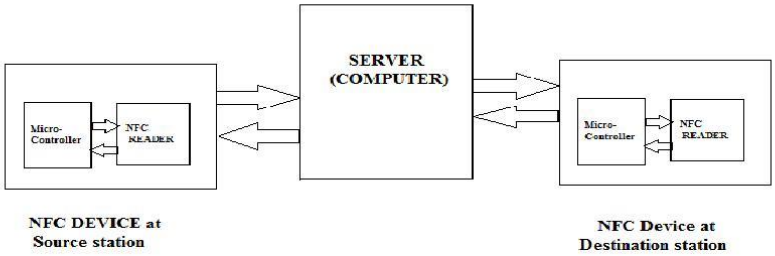


Fig. 2: Block Diagram of the proposed system

B. Software implementation

The system consists of two types of processing units which interact with each other. One is the microcontroller board at each entry, exit and recharge point which controls NFC/RFID module, turnstile, number of passengers. The other is central server which maintains and updates all user accounts, sends required data to all microcontroller boards.

1) Microcontroller Software: The code for Arduino Micro is developed in Arduino IDE. Arduino IDE is an open source software to develop and upload code for Arduino boards. The environment is written in Java and based on Processing, avr-gcc, and other open source software.

2) Central Server Software: The central server is implemented on computer and code is developed on Processing Development Environment(PDE). Processing is an open source language for development of codes on different platforms for different purposes. Default Java mode is utilised for developing the code in PDE. [7]

3) Code Flow: The microcontroller commands NFC/RFID module to continuously scan for NFC tag/NFC phone in vicinity. If the NFC tag/NFC phone is present in range, the microcontroller reads its NUID through NFC/RFID module. The NUID (Number Unique Identifier) is sent to central server with a header and some other data. This data includes details about station, type of point (entry, exit or recharge), number of passengers (only at entry point) and recharge value (only at recharge point). The central server acts according to data received. It updates the user account and sends a positive acknowledgement and number of passengers (only at exit point). If any error occurs, the central server will send negative acknowledgement. Based on acknowledgement received microcontroller gives or denies access to user.

The user is identified by the NFC device and the device asks the user to enter the number of passengers who want to gain access into the station. The device sends the NUID, the station code and the number of passengers to the server. Based on this data the necessary changes are made in the users account in the server. Now the system (server) knows that the user is travelling.

When the user alights at the destination station and taps the NFC tag/NFC enabled mobile phone at the exit NFC device, the device sends the NUID and the station code to the server.

Based on the data provided by the server, the device knows that the user is trying to exit the station and the number of passengers who were given access to the station.

Based on this information appropriate amount is deducted from the users account. The device commands the turnstile gates to allow exit to the passengers.

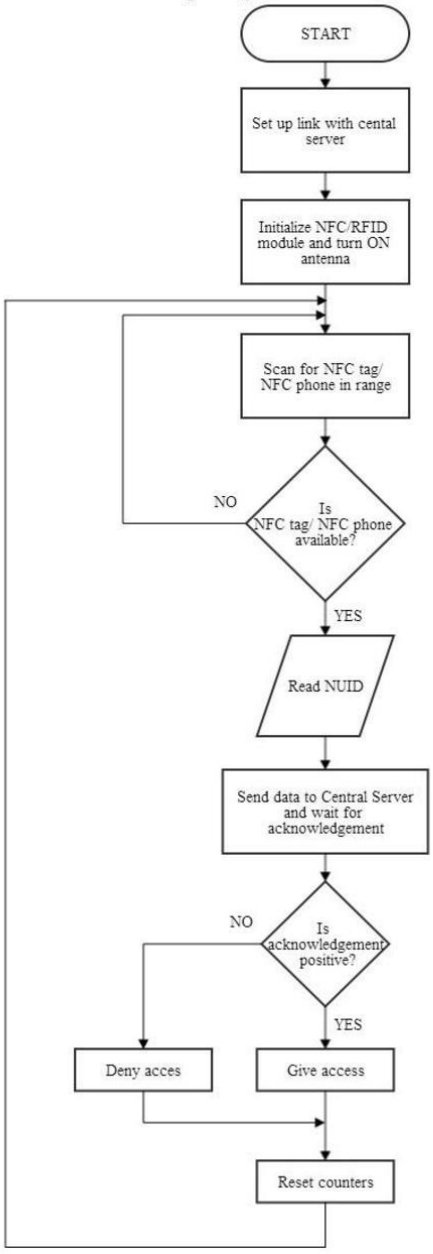


Fig. 3: Code flow at Microcontroller

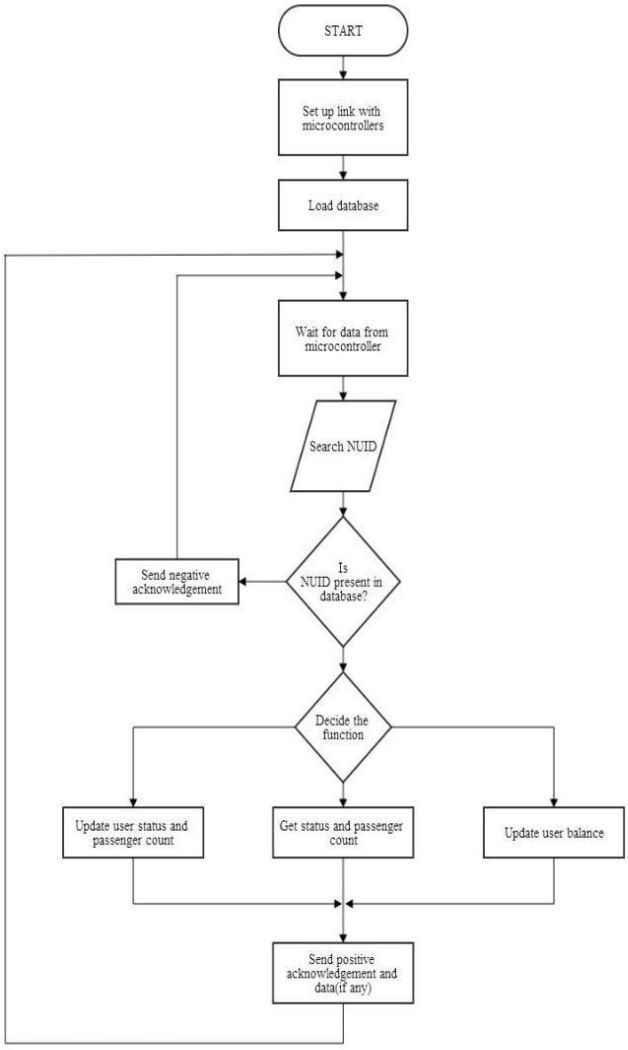


Fig. 4: Code flow at Server

III. ADVANTAGES OF PROPOSED SYSTEM

The NFC based ticketing system is a very convenient way of gaining access to the Metrorail system, it saves a lot of user's time which was wasted in getting a ticket as the access to the station is provided at the touch of a tap..The NFC based system is a secure way of getting the ticket as the range of operation of the device is just a few centimetres, no unauthorized person can hack the system. Moreover, even users with NFC enabled phones can use this system with complete security.[8] The hacker needs to be very close to the device if he has to gain unauthorized access.

Unlike Bluetooth and other wireless systems, the data transmission link is established and at the same time the data is transmitted, so the user does not have to wait for long time for the data exchange to take place. The NFC cards can be used again and again. This saves a lot of paper which was previously used to print tickets. The NFC tags are available at very less cost and hence can be provided at nominal price by the Metrorail system authority and NFC enabled phones are not required by the user to use the NFC based ticketing system.

The device's use can also be extended to do NFC based transaction at various places like colleges, buses, amusement parks, corporate offices, industries, etc. Thus requiring less paper work and saving the users valuable time.

IV. LIMITATIONS

The Initial set up cost of the proposed system will be higher as compared to traditional ticketing system. If NFC Tag is lost or stolen it can be misused. Also, the tag may not be read if the tag is not aligned properly at the NFC device or if the tag is not tapped properly at the device.

V. CONCLUSIONS

Although, the proposed system will have a high set up cost, in the long run it will be beneficial for the Indian Metrorail. Since the maintenance cost is very less due to absence of human ticket vendors, this ticketing system will quickly recover its set up cost. Also it will be very convenient for the passengers and solve the problem of making the passengers stand in large queues and thus increase the overall efficiency of Metrorail travel. Moreover, this system has future scope of being capable of extended to general transactions as stated in the advantages which will make it even more useful.

ACKNOWLEDGMENT

Dr. R. D. Daruwala, Professor, Electrical Engineering Department, Veermata Jijabai Technological Institute

REFERENCES

[1]. NFC Forum
[2]. Near Field Communication - Interface and Protocol (NFCIP-1) ECMA-340 3rd Edition / June 2013
[3]. NFC Data Exchange Format, NFC Forum Technical Specification, NDEF 1.0, July 2006.
[4]. NFC Record Type Definition, NFC Forum Technical
[5]. Arduino Website (http://arduino.cc/en/Main/arduinoBoardMicro)
[6]. Cryton Technologies MIFARE Reader CR038 RFID-ICRW-CR038 User manual V1.0 Jan 2013
[7]. Processing 2 Environment IDE (http://processing.org/reference/environment/)
[8]. A Smart Card Alliance Contactless and Mobile Payments , Council White Paper ,Publication Date: May 2009 , Publication Number: CPMC-09001

Acceptance Letter

Dear Author,

We are pleased to inform you that the above research paper has been Accepted for publication in **International Journal of Engineering Research and Development.** Your paper will be published in next Issue. Your work will be subject to the publication policies of the journal, available at www.ijerd.com

It would publish online as well as print versions.

Ref. id	**33131**
Title of Paper	**Design of an Automatic Fare Collection System Using Near Field Communication with Focus on Indian Metrorail**
Author(s)	**Ajay Gore, Nirvedh Meshram, Sumit Gadi, Rahul Raghatate**

Reviewers Comments:

1. **Quality of Manuscript is good**.
2. **Consolidated Decision: Accepted for publication**

Let me know if you have any further queries. We always look forward to hearing from you.

IJERD indexing partner & libraries: http://www.ijerd.com/indexing.html

IJERD payment option: http://www.ijerd.com/payment%20option.html

With Regards,
Editor-in-Chief
International Journal of Engineering Research and Development(IJERD)
e-ISSN : 2278-067X, p-ISSN : 2278-800X
E-mail: ijerd.editor@gmail.com
URL: www.ijerd.com

BIBLIOGRAPHY

1) William Stallings, Data and Computer Communications 8th edition 2006

2) David E. Simon, An Embedded Software Primer

3) V. Daniel Hunt, RFID- A guide to radio frequency Identification

4) Processing Creative Coding and Generative Art in Processing 2; Ira Greenberg, Dianna Xu, Deepak Kumar

5) Processing: a programming handbook for visual designers and artists; Casey Reas, Ben Fry

REFRENCES

[1] NFC Forum

[2] Near Field Communication - Interface and Protocol (NFCIP-1) ECMA-340 3rd Edition / June 2013

[3] NFC Data Exchange Format, NFC Forum Technical Specification, NDEF 1.0, July 2006.

[4] NFC Record Type Definition, NFC Forum Technical

[5] Arduino Website and Forum (http://arduino.cc/en/Main/arduinoBoardMicro)

[6] Cryton Technologies MIFARE Reader CR038 RFID-ICRW-CR038 User manual V1.0 Jan 2013

[7] Processing 2 Environment IDE (http://processing.org/reference/environment/)

[8] A Smart Card Alliance Contactless and Mobile Payments , Council White Paper ,Publication Date: May 2009 , Publication Number: CPMC-09001

ACKNOWLEDGEMENTS

Planning, designing and implementing this project has been an enriching experience and yet highly enjoyable for each of us. So before presenting our report we would like to first thank the individuals concerned for the success of our project.

We are highly grateful to the help and support offered to us by Prof. R. D. Daruwala, V.J.T.I., under whose able mentorship we have implemented our project. Without his guidance and support it would not have been possible for us to implement our project and achieve the results observed in the give span of time.

We would also like to thank all our colleagues for their help, suggestions and moral support offered to us.

Through this project we leant the importance of team work, sharing of knowledge and exchange of ideas. With every mistake, a valuable lesson was learnt and conceptual and practical knowledge was enhanced. We learnt the value of not giving up due to initial unsatisfactory outcomes and having the urge to always strive for perfection and better results and being open to each other's suggestions to achieve improvement.