CSE316 Microcontroller Project

Classroom Monitoring with RFID

by

Mahmudur Rahman Hera (1105001)

Amatur Rahman (1105003)

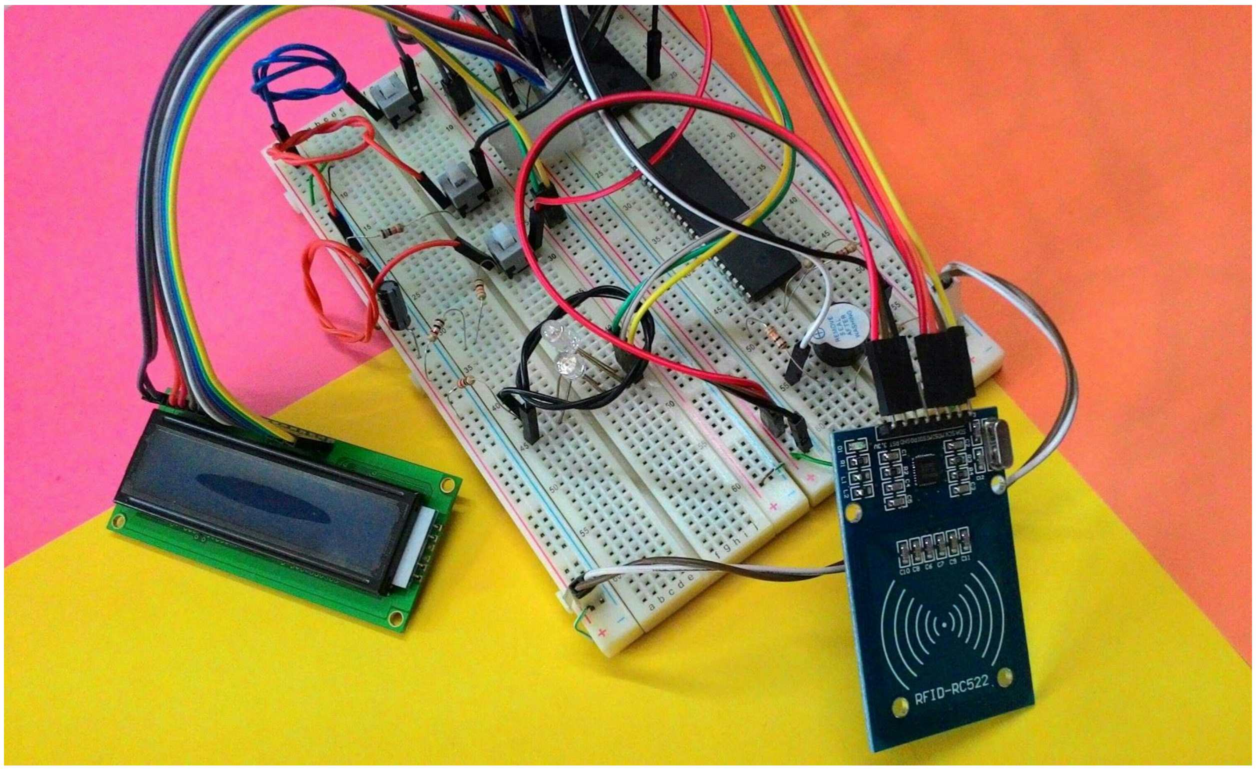


Table of Contents

Introduction

This project aims to develop a safe and secure monitoring system for the classroom environment. For this project we built a system using RFID sensor. RFID tag, another for the class administrator containing RFID reader monitoring the classroom. RFID reader captures the entering and leaving of a student in classroom and will keep count of how many students enter and leave. The count is shown on a LCD display on the administrative device. The attendance of students are captured using RFID tags. A student holds his ID card containing the RFID tag in front of the RFID reader, it reads the data and verifies it with the data stored in the microcontroller. If the data matches, then it displays a message on the LCD. If there is a mismatch, a message is displayed and it will buzz off the buzzer. The history of each day is saved in an internal database maintained by EEPROM. If a student attempts to leave or sneak out during the class, the RFID module will be able to detect it. RFID reader can record and verify the RFID tags in the classroom.

Watch our project video

Hardware Requirements

Hardware : **List of hardwares and cost**

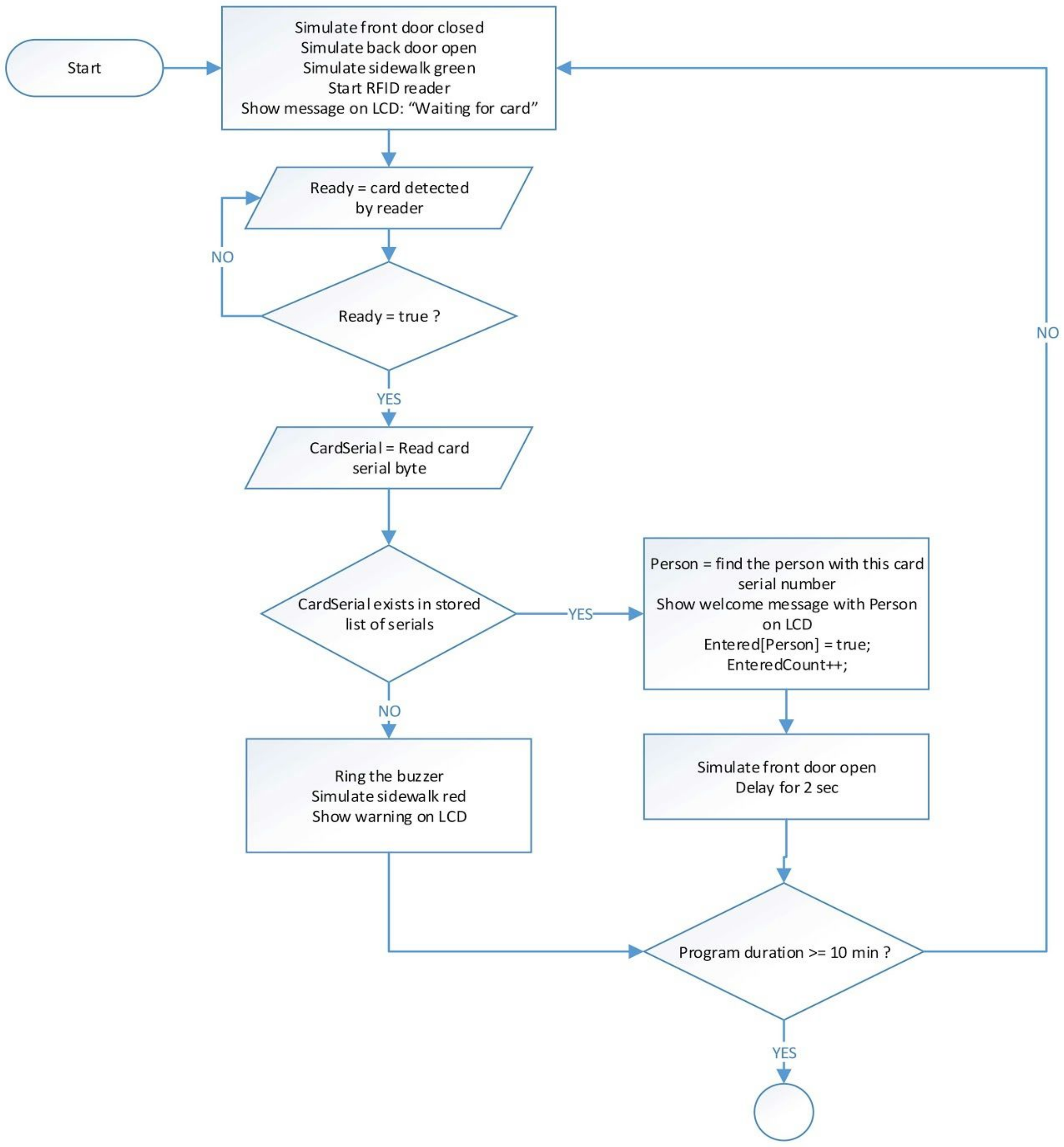
Equipments	**Cost (Tk)**
Atmega32 microcontroller	500
1 RFID Reader module (RC522)	500
4 RFID Tags (MiFare One)	Packed with Mifare RC522 module
16x2 LCD Alphanumeric Display	500
Buzzer	10
RGB LEDs	100
Resistors	10
Capacitors	10
2 SPDT Push Button Switches	20
1 Push Button	10
USB ISP 2.0 AVR Programmer	300

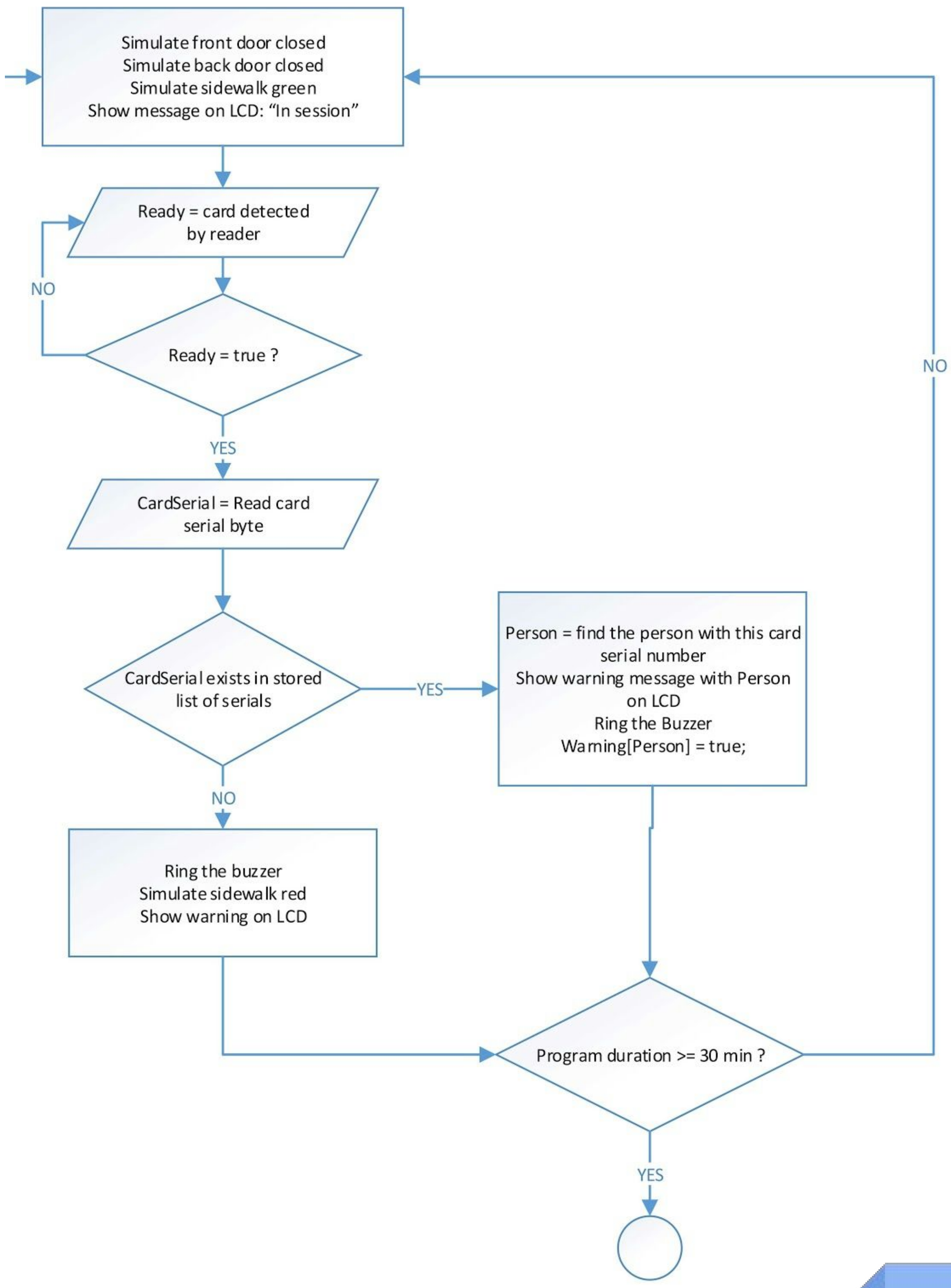
Software Requirements

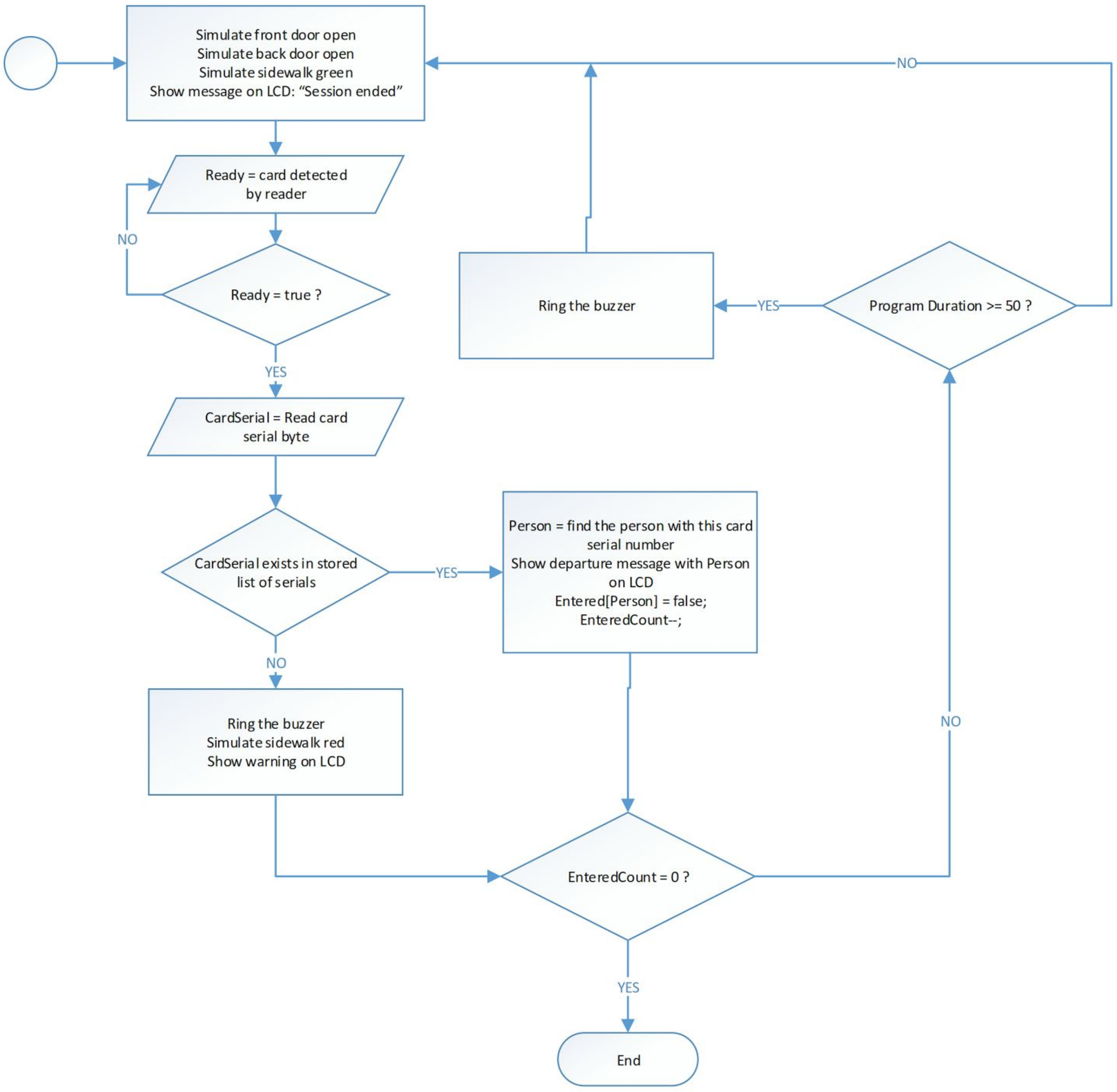
List of **Softwares used**

- ATmel Studio 7 (to compile .c code and build .hex and .eep file)
- eXtreme Burner - AVR (to load .hex and .eep file onto ATmega32)
- Proteus 8 Professional (for circuit design)

Flowchart


Start
Simulate front door closed
Simulate back door open
Simulate sidewalk green
Start RFID reader
Show message on LCD: "Waiting for card"
Ready = card detected by reader
Ready = true ?
NO
YES
CardSerial = Read card serial byte
CardSerial exists in stored list of serials
YES
NO
Person = find the person with this card serial number
Show welcome message with Person on LCD
Entered[Person] = true;
EnteredCount++;
Simulate front door open
Delay for 2 sec
Ring the buzzer
Simulate sidewalk red
Show warning on LCD
Program duration >= 10 min ?
NO
YES

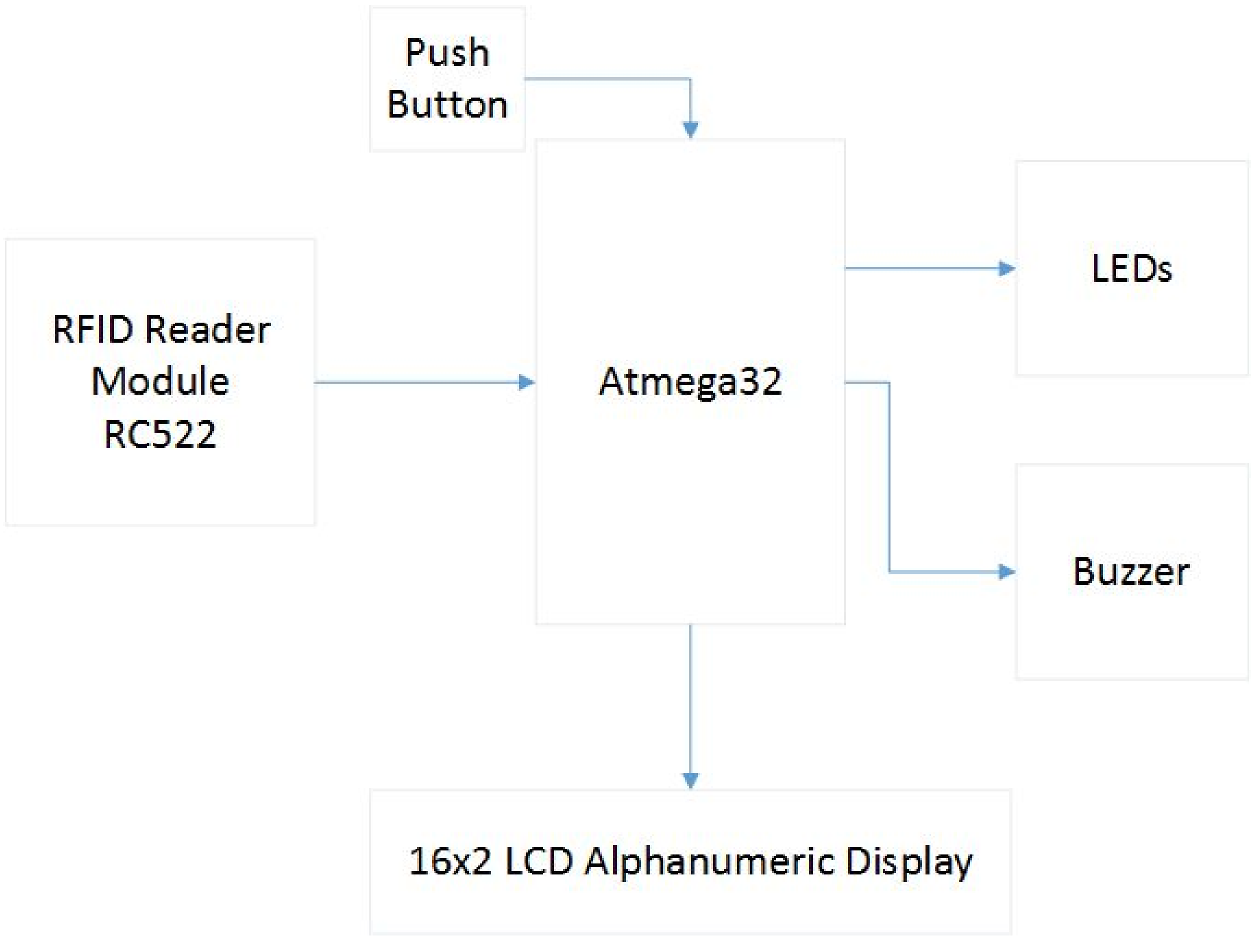

Simulate front door closed
Simulate back door closed
Simulate sidewalk green
Show message on LCD: "In session"
Ready = card detected by reader
NO
Ready = true ?
YES
CardSerial = Read card serial byte
CardSerial exists in stored list of serials
YES
Person = find the person with this card serial number
Show warning message with Person on LCD
Ring the Buzzer
Warning[Person] = true;
NO
Ring the buzzer
Simulate sidewalk red
Show warning on LCD
Program duration >= 30 min ?
NO
YES



Simulate front door open
Simulate back door open
Simulate sidewalk green
Show message on LCD: "Session ended"
Ready = card detected by reader
NO
Ready = true ?
YES
CardSerial = Read card serial byte
CardSerial exists in stored list of serials
YES
Person = find the person with this card serial number
Show departure message with Person on LCD
Entered[Person] = false;
EnteredCount--;
NO
Ring the buzzer
Simulate sidewalk red
Show warning on LCD
EnteredCount = 0 ?
YES
End
NO
Program Duration >= 50 ?
YES
Ring the buzzer
NO

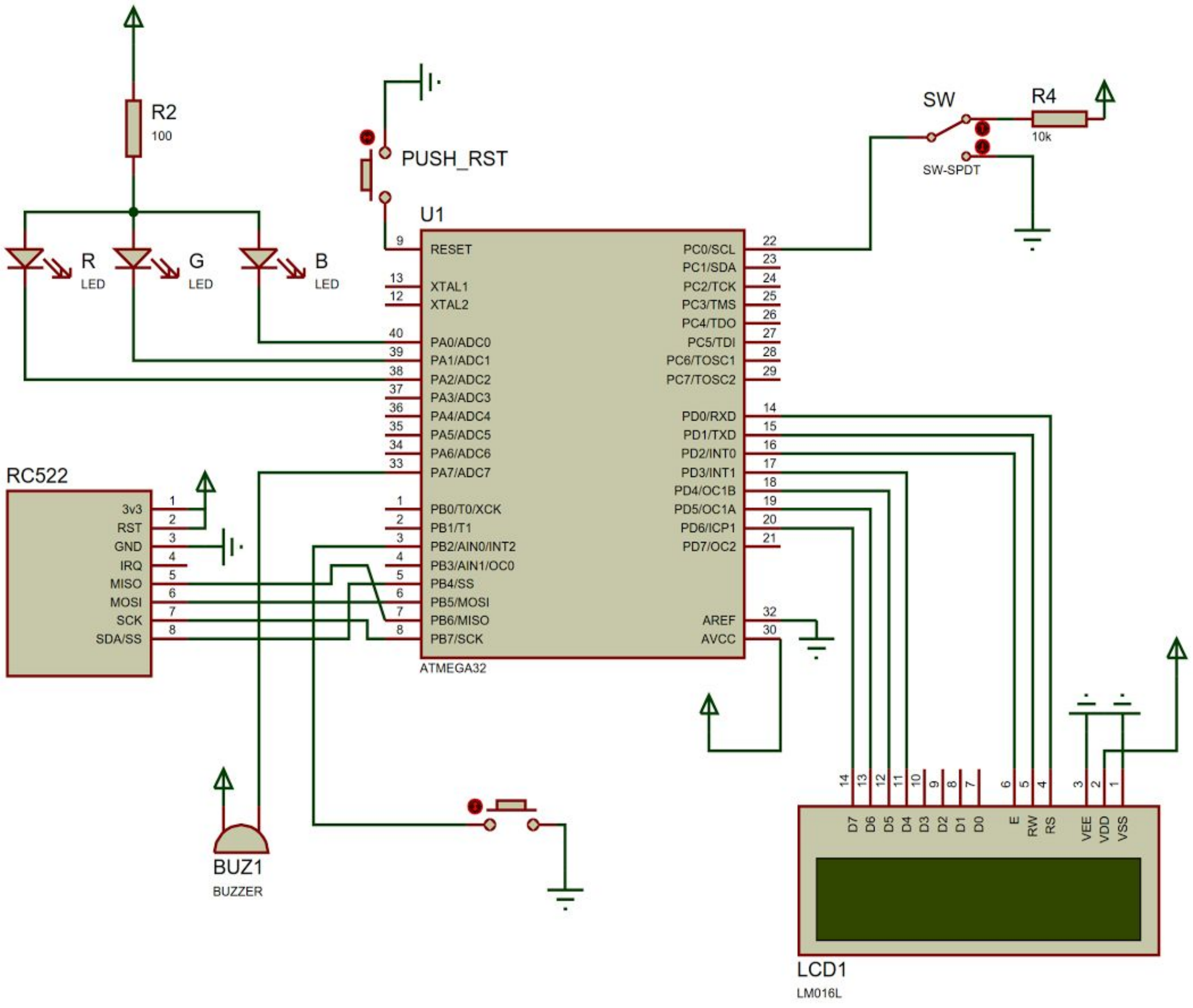
Block Diagram

showing **Input and Output**



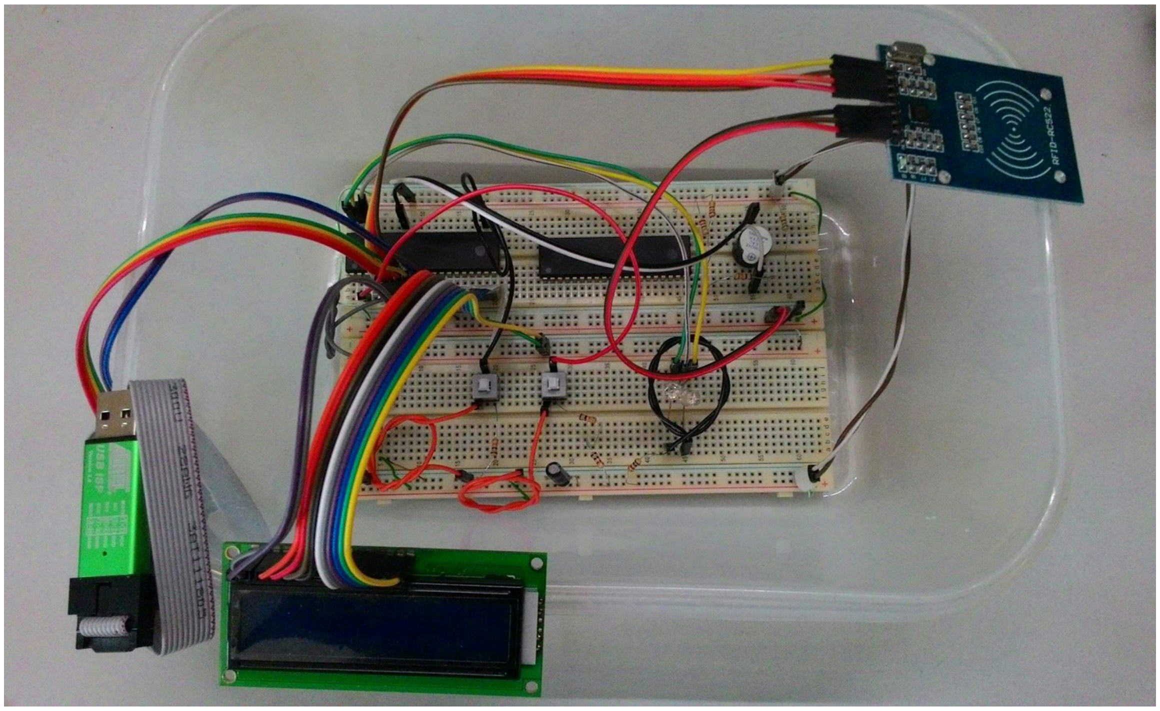
Circuit Diagram

showing **Complete Connection**



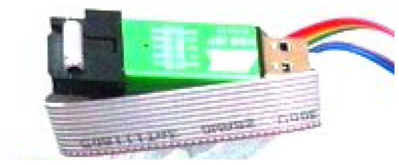
Actual Circuit

Snapshot **of the working circuit.**



USB ISP Programmer

This was used to burn the .hex and .eep file into the Atmega32 flash memory and EEPROM. It is powered off of 5V USB bus. Our whole project is powered using its USB port power supply.



Description of Modules & Used Libraries

Describing **all the hardware parts and their connections.**

RFID-RC522

Mifare RC522 is the high integrated RFID card reader which works on non-contact 13.56mhz communication. RC522 use the advanced modulation system. This module uses 3.3V power supply, and communicates directly with any Atmega32 MCU by connecting through SPI protocol, which ensure high reading distance. We used the following library to interface with the RC522 board.
https://github.com/asif-mahmud/MIFARE-RFID-with-AVR/blob/master/avr-rfid-library-1.0.0.tar.gz

This library is for Atmega48, so we had to do some tweaks to make it work for Atmega32. Pin and port definitions were changed to adapt it to Atmega32.

The reader RFID-RC522 interface via SPI with the microcontroller. It only uses a subset of that library: to read the UID. Uses the SPI port of Atmega32 (port B). All the necessary codes to interface RC522 with Atmega32 are available via `my_header.h` along with the code for LCD interfacing. The main workflow of our project can be traced from `main.c` filed under the folder *"source code"*. Documentations are inlined with the code.

SPI:

Serial Peripheral Interface (SPI) is an interface that enables the serial (which means one bit at a time) exchange of data between Atmega32 (master) and RFID Reader RC522 (slave). It operates in full duplex mode, meaning that data can be transferred in both directions at the same time. It automatically detects the card close to the antenna area, and generate an interrupt signal to Atmega32. The MFRC522 acts as a slave during SPI communication. The SPI clock signal SCK must be generated by the master. Data communication from the master to the slave uses the MOSI line. The MISO line is used to send data from the MFRC522 to the master. Data bytes on both MOSI and MISO lines are sent with the MSB first. Data on both MOSI and MISO lines must be stable on the rising edge of the clock and can be changed on the falling edge. Data is provided by the MFRC522 on the falling clock edge and is stable during the rising clock edge.

- ***SPI read data***

Reading data using SPI requires the byte order shown in Table to be used. It is possible to read out up to n-data bytes. The first byte sent defines both the mode and the address. Remark: The MSB must be sent first.

Table. MOSI and MISO byte order

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	address 1	address 2	...	address n	0
MISO	X	data 0	data 1	...	data n - 1	data n

X = Do not care.

- ***SPI write data***

To write data to the MFRC522 using SPI requires the byte order shown in Table. It is possible to write up to n data bytes by only sending one address byte. The first send byte defines both the mode and the address byte.

Table. MOSI and MISO byte order

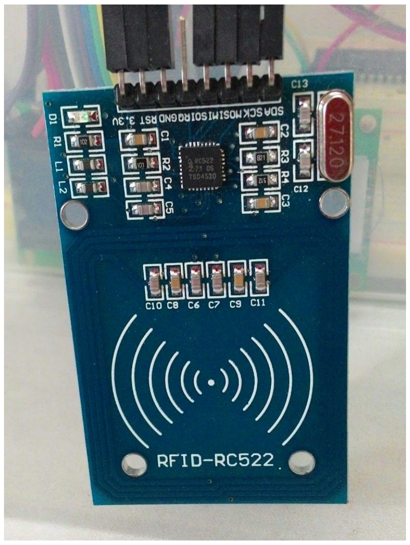
Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	data 0	data 1	...	data n - 1	data n
MISO	X	X	X	...	X	X

X = Do not care.

- ***SPI address byte***

The MSB of the first byte defines the mode used. To read data from the MFRC522 the MSB is set to logic 1. To write data to the MFRC522 the MSB must be set to logic 0. Bits to 1 define the address and the LSB is set to logic 0.

RC522 Connections



Connections from board to Atmega32

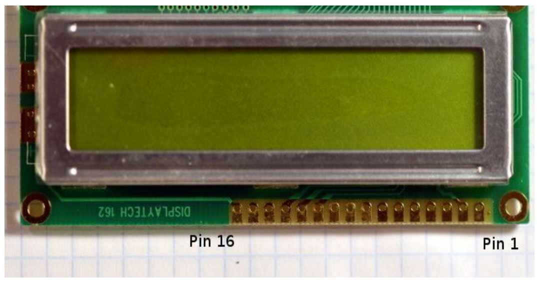
- **SDA** *(-CS)* to **PB4** (Pin 5 on Atmega32)
- **SCK** to **SCK (PB7)** (Pin 8 on Atmega32)
- **MOSI** to **MOSI (PB5)** (Pin 6 on Atmega32)
- **MISO** to **MISO (PB6)** (Pin 7 on Atmega32)
- **IRQ** (not connected)
- **GND** to GND
- **RST** *(-Reset)* to GND (as we didn't keep any method to individually reset RFID reader)
- **3.3V** (PIN1) to **4.3V**

RFID Mifare Tags

The RFID reader sensor RC522 receives information from RFID Mifare tags.These tags are sold in the form of card or keychain. In this project, a keychain tag (blue) and a card tag (white one) was used, as shown in the picture to the right. These tags contain EEPROM with 1 Kb of available space for data where a identification number is saved.

We programmed the white card to be a valid card, and the blue keychain containing the tag to be marked as unrecognized.

16x2 Alphanumeric LCD Display



The LCD has 16 connector pins. Connections are:

- LCD 1 *(GND)* to GND
- LCD 2 *(VCC)* to 5V
- LCD 3 *(contrast)* to GND
- LCD 4 *(RS)* to PD0 (Pin 14 on Atmega32)
- LCD 5 *(R/-W)* to PD1 (Pin 15 on Atmega32)
- LCD 6 *(Clock or Enable)* to PD2 (Pin 16 on Atmega32)
- LCD 7, 8, 9, 10 are *not connected*
- LCD 11 *(Data 4)* to PC0 (Pin 17 on Atmega32)
- LCD 12 *(Data 5)* to PC1 (Pin 18 on Atmega32)
- LCD 13 *(Data 6)* to PC2 (Pin 19 on Atmega32)
- LCD 14 *(Data 7)* to PC3 (Pin 20 on Atmega32)
- LCD 15 (to VCC) and LCD 16 (to GND) are for background light.

RGB LED

An RGB LED with a common anode is controlled.

Output bits are:

- **Common Anode** to **5V** via 1K-ohm resistor
- **RED cathode** to **PA2** (Pin 38 on Atmega8)
- **GREEN cathode** to **PA1** (Pin 39 on Atmega8)
- **BLUE cathode** to **PA0** (Pin 40 on Atmega8)

Buzzer for Tone Generation

To generate beep tones upon warning and success, connect

- One pin of **Buzzer** to **PA7**
- The other pin to GND

Problems Faced

Describing **the practical issues and observations made while putting the project together.**

- The library for LCD module we used initially had code for 20x4 LCD display, which we changed to adapt it to 16x2 display.
- In the library for RFID reader, code for Atmega48 was provided, which we had to change to Atmega32 code.
- At first the RFID module was not detecting the tags, although the RFID reader was initialized correctly. After changing the voltage at pin1 (3v3) of the reader several times, we found that keeping the voltage at that pin slightly higher than 3.5 volt makes it detect the MiFare white card. Keeping the voltage in the range of 4 to 4.2 volts made both the blue tag and the white tag be detected smoothly by the reader.
    - This change of voltage was done by adding resistors and applying voltage divider rule to create desired voltage level from main 5V power supply of Atmega32.
- While trying to set up the EEPROM, initially we didn't set any initial value to our EEPROM variables, so our program was inconsistently showing garbage value in the stored variables. To fix this we observed that, upon compilation of the program, GCC outputs two Intel-HEX format files. One is .HEX which contains the program data and is loaded into the AVR's memory, and the other is a .EEP file. The .EEP file contains the default EEPROM values, which we later loaded into AVR via programmer's EEPROM programming functions separately. As we were burning the .HEX only, the variables were not properly initialized.
    - To set a default EEPROM value in GCC, we simply assigned a value to our EEMEM variable, like this: `uint8_t EEMEM SomeVariable = 0;`
- We also faced problems while working with interrupts. In our project, we used INT2 (interrupt 2) by connecting a push button to PIN3. This presented us with the "switch bounce" problem, the push button 'bounced' as in giving multiple on/off actions. For us, each press of the switch was causing the ISR to be called twice.
    - To solve this we added delay at start of the ISR routine : `_delay_ms(500);`
    - And cleared the flag : `GIFR |= (1<<INTF2);`

Acknowledgements

These are the sites **that helped us to set up the project.**

For basic interfacing with ATmega32

- http://maxembedded.com/
- http://www.avrfreaks.net/
- http://avrprojects.info/
- http://extremeelectronics.co.in/

For making RFID Reader work

- https://github.com/miguelbalboa/rfid
- https://github.com/asif-mahmud/MIFARE-RFID-with-AVR

That's it!

Apart from helping us gain insights into the functions and myriads of possibilities RFID coupled with ATmega32 microcontroller, this system will be beneficial in keeping track of children in a small classroom environment. It can also serve as an automatic attendance system. If we can we hope to add more functionalities to the project and try to implement in our own classrooms! We hope that this small effort of us will inspire other electronics hobbyists and students alike to work with RFID. The complete source code is provided with the report with documentation. We encourage all to be creative and to extend the project by incorporating their own creative ideas.

Hope you have fun exploring the project!

December 23, 2015