Abstract

The purpose of the RIFD attendance tracker is to allow teachers to track student's attendance without wasting valuable time taking roll. This is achieved through the use of RFID tags assigned to individual students that allows them to check in at the beginning of each class. The teacher would be provided access to the collected attendance data via an Android application for phone or tablet. The following is a systematic walkthrough of the waterfall process (Concept, Requirements, Design, Implementation, Testing, and Maintenance) of the prototype system developed with a conclusion and recommended improvements at the end.

Concept

The RFID IoT device will function as an easy way to swipe into class. Each swipe will be recorded and viewable by a teacher through an application on a mobile device.

Task 1

Pass a RFID tag by a RFID reader and decode the signal on the microcontroller.

Task 2

The decoded signal is checked against a local database (or registry).

Task 3

If the RFID checks as a valid ID, then a light will turn on indicating so.

Task 4

Record the history of the attempt so it can be accessed by a mobile device.

Task 5

Allow the list of valid IDs be editable and updatable.

Cost

The cost of materials must be under $105 for a prototype board. The engineering cost is ignored since I am a student doing it for educational purposes.

Table 1: Project Costs

Item	Cost
RFID Reader	< $15
Microcontroller	< $85
RFID Tags	< $5

Schedule

Date	Sub Task	Task #	Date Completed
2/7/17	Obtain Materials		2/15/17
2/9/17	Connect system and verify voltages are being read	Task 1	2/27/17
2/10/17	Decode Signal	Task 1	3/1/17
2/13/17	Functionality check against database	Task 2	3/6/17
2/15/17	Output LED indication	Task 3	3/1/17
2/22/17	Testing Reader		3/28/17
2/28/17	HTTP Post	Task 4	3/31/17
3/6/17	API Store and Read	Task 4	4/8/17
3/10/17	HTTP Get	Task 4	4/2/17
3/15/17	Test Database		4/5/17
3/22/17	RFID registered list update	Task 5	4/8/17
3/31/17	Filter histories stored in App	Task 5	4/8/17
4/3/17	Add, Remove, and Edit Valid Students	Task 5	4/8/17
4/5/17	Testing App		4/9/17
4/7/17	Finish Presentation		4/9/17
4/10/17	Project Presentation		4/9/17

Requirements

The RFID reader response timing is defined as a soft real-time system since missing deadlines will not be detrimental. However, the accurate reading of the RFID tags is crucial.

The RFID reader generates current once a RFID tag enters the EM field. The data is passed to the microcontroller (when talking about the microcontroller it includes the microprocessor). The microcontroller decodes data from the card reader and validates if it is a registered RFID. The microcontroller then posts the correct RFIDs as a history in a database. Correct RFIDs lights a LED indicating to the user that the RFID was valid and read. The database and history will then be accessible to the teacher from a mobile device. *Figure 1: HW & SW Interface Diagram* below shows the interfaces.

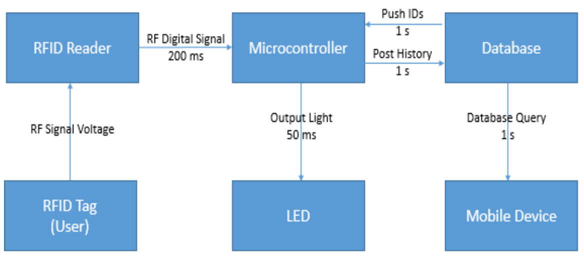


Figure 1: HW & SW Interface Diagram

RFID Read Requirements

The following are the requirements for the interactions between the RFID reader and microcontroller and the interaction between the microcontroller and the LED (*HW & SW Interface Diagram*).

Functional Requirements

The RFID tags must be read from 1 inch away or less by RFID reader.

There are three swipe error scenarios. The first is a false negative. A false negative is when a registered user scans their card and it is read as a negative. In this scenario, additional reads would be required to register a student's entrance or exit. False negatives are undesirable, though they are not detrimental. However, false positives and incorrect positives are less ideal scenarios. A false positive is when a card that shouldn't be accepted is accepted. An incorrect positive is when a card that should be accepted is accepted as the wrong card. This will be demonstrated during tasks 1-3.

Non-Functional Requirements

- Each user has unique RFID tag
- Only one transaction per second to avoid duplicate swipes in one swipe
- RFIDs may be added and removed from authenticated list
- RFID tags must follow MIFARE protocol [1].
  - Operating frequency: 13.56 MHz
  - Data transfer: 106 kbit/s
  - Data integrity: 16-bit CRC, parity, bit coding, bit counting
  - 7 Byte UID or 4 Byte NUID
- NUID (4 Byte at 106 kbit/s) read 80-100 ms
- SPI communication between RFID reader and microcontroller

Table 2: RFID Read Accuracy Requirements

Error Type	# of Errors per 100 swipes
RFID signal sent 200 ms	≤ 3
False Negative	≤ 3
False Positive	0
Incorrect Positive	0
LED not indicating	0

Post/Get Server

These are the interactions between the database as shown in the *HW & SW Interface Diagram*. The database interacts with both the microcontroller and the mobile device.

Functional Requirements

The total execution and what the user sees on the application is dependent on the data exchange speeds of the environment. Most users understand that occasionally a low Wi-Fi signal can prevent an instantaneous update. The speed of the updates and exchanges between the database and controller are not crucial to the system. The same is true with the database and mobile device. As seen in the HW & SW Interface Diagram, there are more relaxed requirements. The intention is to meet these requirements, but if they aren't met, there won't be any serious consequences.

The teacher should be able to filter swipe history by student, date, and date ranges.

Non-Functional Requirements

- The microcontroller must connect to the database via Wi-Fi.
- SQL must be used as the database language.
- Database must be accessible by a mobile device.
- The app must be an android app for the mobile device.

Table 3: Post/Get Timing Requirements

Error Type	# of missed deadlines per 100 swipes
Post History in 1 s	≤ 3
Push ID in 1 s	≤ 3
Mobile Device Query 1 s	≤ 3

Design

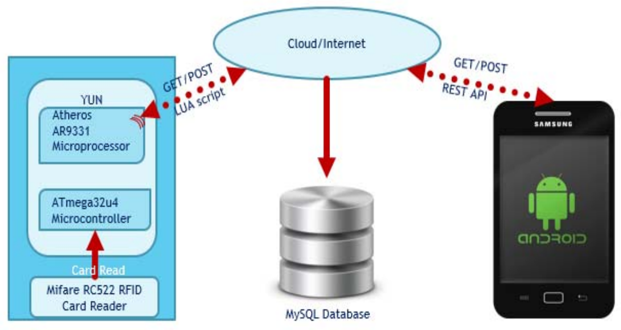


Figure 2: Design Overview

Hardware and Material

Total Cost: $80.90

Microcontroller Board

[2] *Arduino YUN ($69.95)*

- ATmega32u4 Microcontroller
- Atheros AR9331 Microprocessor
- Input Voltage 5V
- 20 Digital I/O (7 PWM outputs)
- Flash memory: 32KB (Arduino)/16MB (Linux)
- 16 MHz Clock Speed
- Wi-Fi and Ethernet

[3] RFID Reader w/RFID Tags

Mifare RC522 Card Read ($10.95)

- Operating current: 13—26mA/DC 3.3V
- Idle current: 10-13mA/ DC 3.3V
- Sleep current: <80uA
- Peak current: <30mA
- Operating frequency: 13.56MHz
- Data transfer rate: Maximum 10Mbit/s
- Read distance of 1.2-2 inches
- 3 key ring FOBs
- 5 RFID cards

SW Structure

The software is partitioned into what functions are needed on the microcontroller, android application, and the database. The interfaces between them are also defined and can be seen in *Figure 3: SW Module Partitioning* below.

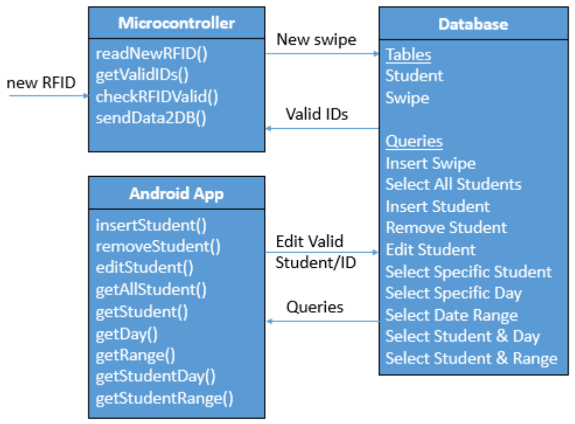


Figure 3: SW Module Partitioning

The following data flow diagram in *Figure 4* aids in the programming of the RFID read flow. The flow goes from the swipe coming in as a byte until it is sent to the database.

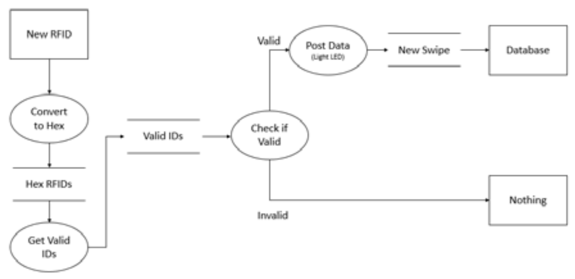


Figure 4: RFID Read Data Flow Diagram

The mobile application is to be developed with the structure defined in *Figure 5* and *Figure 6*. There should be a view for the history of swipes and a view to go in and edit the valid students with associated RFIDs.

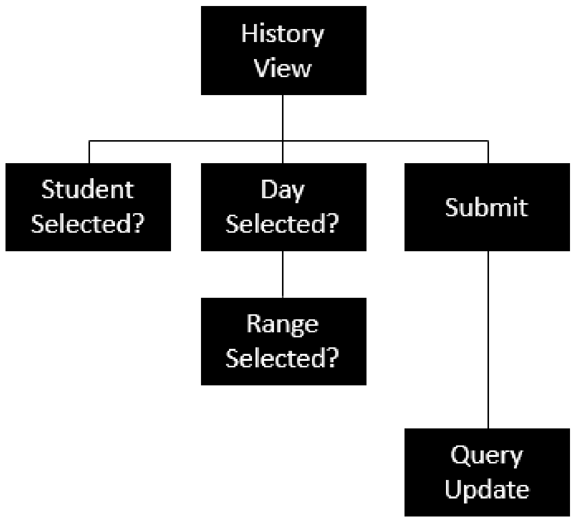


Figure 5: History View Structure Chart

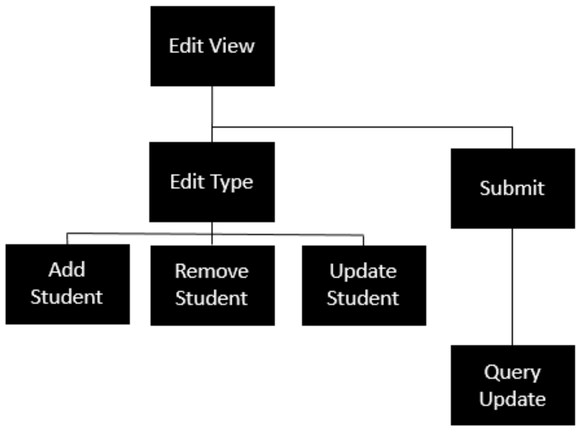


Figure 6: Edit View Structure Chart

Test Cases

The test vectors and results are shown together in the Test section.

Three test vectors for *Table 2: RFID Read Accuracy Requirements* are in *Table 4*, *Table 5*, and *Table 6*. Each test vector includes testing for the signals sent before the 200 ms deadline. The design of the test vectors is to test more robustly for false positives and incorrect positives. In each test vector, Two RFIDs are valid and the rest are invalid RFIDs. The test checks to see if any of the invalid RFIDs read as positives or the valid RFIDs are misread as invalid. The pairs of valid RFIDs for the test vectors be the edge case, RFIDs with matching last two bytes of RFID, and RFIDs with matching first and last byte of RFIDs.

There are three test vectors for *Table 3: Post/Get Timing Requirements*. The *Post RFID Test* vector checks for how many deadlines are missed for 100 posts, *Table 8* tests for how many deadlines are missed for 100 pushed IDs, and *Table 9* tests for how many deadlines are missed for 100 queries.

Implementation

This section explains how the system is integrated with both the hardware and software, where the code can be found, what is contained in each file of code, and how testing was automated.

Integration

The Yun is powered by 5 V/1 A micro USB. The Yun then supplies power and ground to the RFID reader. In addition to the power and ground lines, the SPI serial communication requires MOSI (Master Output Slave Input), MISO, SS (Slave Select), SCK (System Clock), and Reset lines between the reader and microcontroller.

The Yun board communicates with the database through the microprocessor. The microcontroller sends a command to the microprocessor to run [4]lua scripts that post-swipe data to the database and get the valid IDs from the database.

The Android app uses an API to access and edit the tables in the database [5].

Code

The code used for implementation of this project can be found on GitHub [6]. The “RFIDtoDB.ino” is a sketch for the microcontroller. The sketch uses Miguel Balboa’s library [7]. The “.lua” files are scripts that connect to the database and query the database. The folder “RFIDCam” contains the Android Studio project that is for the app.

Automate Testing

RFID Read Test 1, RFID Read Test 2, & RFID Read Test 3

During testing, a timestamp was inserted into the code once a RFID was recognized and another timestamp before the data was sent. This allowed the response time of the signals being sent to be checked.

Post RFID Test

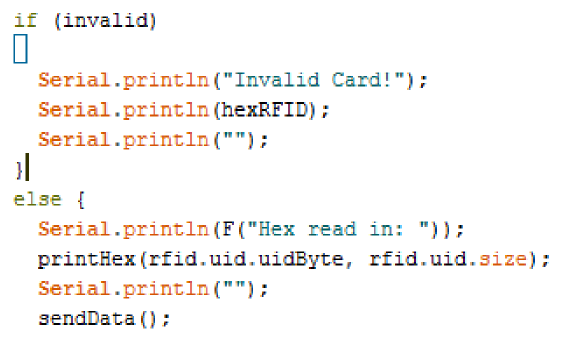
To check the response time, the timestamp on the code was checked right before the data was sent and compared to the timestamp on the database when it was received.

Push RFID Test & Database Query Test

Similarly to the Post RFID Test, a timestamp on the android app was compared to the timestamp on the database of when it was received

Testing

When testing began, it became apparent that the else-statement was left out of if-statement. Consequently, the system would check and show it was invalid but then it would continue to send the data anyways.



```
if (invalid)
{
  Serial.println("Invalid Card!");
  Serial.println(hexRFID);
  Serial.println("");
}
else {
  Serial.println(F("Hex read in: "));
  printHex(rfid.uid.uidByte, rfid.uid.size);
  Serial.println("");
  sendData();
```

After eliminating that defect, tests on the test vectors were able to be continued.

Results

The most important requirement of the system was the accuracy of the RFIDs identifying as the correct RFID. False positives, incorrect positives, or false negative errors were never observed for a misread RFID. Occasionally, some deadlines did not meet the response time requirements for the signal being sent, the post to the database, and getting valid IDs for the database. This was not too frequent to be alarming. It could have been caused by not having the RFIDs close enough. A contraption could be made to make sure that the RFID are passed within the required inch but that would require more time and money to validate something non-vital.

Table 4: RFID Read Test 1

RFID	# Swipes	Valid	Sent	FN	FP	IP
2370DF2B	20	Y	0	0	na	na
4443E12B	10	n	0	na	0	0
4E890785	10	n	2	na	0	0
7132E02B	10	n	0	na	0	0
85F56F3B	10	n	1	na	0	0
8CA0DF2B	10	n	1	na	0	0
DB87DF2B	10	n	1	na	0	0
DCF1715B	20	y	0	0	na	na
TOTAL	100	na	5	0	0	0

*FN = False Negative, FP=False Positive, IP = Incorrect Positive, and na = Not applicable

Table 5: RFID Read Test 2

RFID	# Swipes	Valid	Sent	FN	FP	IP
2370DF2B	20	y	0	0	na	na
4443E12B	20	y	1	0	na	na
4E890785	10	n	0	na	0	0
7132E02B	10	n	0	na	0	0
85F56F3B	10	n	0	na	0	0
8CA0DF2B	10	n	0	na	0	0
DB87DF2B	10	n	0	na	0	0
DCF1715B	10	n	0	na	0	0
TOTAL	100	na	1	0	0	0

Table 6: RFID Read Test 3

RFID	# Swipes	Valid	Sent	FN	FP	IP
2370DF2B	10	n	0	na	0	0
4443E12B	10	n	1	na	0	0
4E890785	10	n	0	na	0	0
7132E02B	10	n	0	na	0	0
85F56F3B	20	y	0	0	na	na
8CA0DF2B	20	y	1	0	na	na
DB87DF2B	10	n	0	na	0	0
DCF1715B	10	n	0	na	0	0
TOTAL	100	na	2	0	0	0

Table 7: Post RFID Test

RFID	# Swipes	# of missed post deadlines
2370DF2B	13	0
4443E12B	12	1
4E890785	12	0
7132E02B	13	0
85F56F3B	13	0
8CA0DF2B	12	2
DB87DF2B	12	0
DCF1715B	13	1
TOTAL	100	4

Table 8: Push RFID Test

Added RFID	Removed RFID	# of missed push deadlines
50	50	5

Table 9: Database Query Test

Query Type	# Queries	# of missed query deadlines
All Students	17	0
Specific Student	17	0
Specific Day	17	0
Date Range	17	0
Student & Day	16	0
Student and range	16	1
TOTAL	100	1

Deployment & Maintenance

If the prototype (proof-of-concept) was used as the product being deployed it would require wiring between the RFID reader and the microcontroller. Software deployment on the board would require saving the lua script files on the root folder of the microprocessor. Software for the microcontroller would also need to be downloaded. Lastly, the Wi-Fi of the Yun would need to be configured to the network it is being implemented on.

Customer support would include general technical support, software glitches, defects in the hardware, and wiring issues between the microcontroller and RFID reader. The first two could be addressed remotely. A new version of the software could be deployed and downloaded onto the board via Wi-Fi. Defects in the hardware of the RFID reader would require replacing and rewiring. That possibly could be done over the phone depending on the customer. However, a defect in the microcontroller would require replacing, rewiring, and reconfiguring everything.

Conclusion & Improvements

Designing a single board that integrates the RFID reader and microcontroller is recommended to improve the product before deployment. It would reduce cost per unit and require less maintenance and support later on. Integrating the boards would reduce the cost of wiring and minimize the need of customer support for wiring issues. The Yun board contains more than what is necessary for the system. A simpler microcontroller and OS would still be able to meet the requirements. A custom designed board could take more time but reduced costs, complexity, and less maintenance would result. In addition, software deployment on the board would be in a single location and less complex. The only reason that it may not be feasible to integrate together would be due to EMC issues. A negative from integrating the boards would be an increase in cost per unit repaired if there was a hardware defect because the whole board would need to be replaced every time. Unlike the prototype, where if the RFID reader was defective you would only need to replace the reader.

References

[1] NXP, "MIFARE® Classic 1K - Mainstream contactless smart card IC for fast and easy solution development|NXP," [Online]. [Accessed 20 2 2017].

[2] "Arduino YUN," Arduino, 2015. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardYun. [Accessed 15 2 2017].

[3] "MFRC522 Datasheet," NXP, 27 4 2016. [Online]. Available: https://www.nxp.com/documents/data_sheet/MFRC522.pdf. [Accessed 25 02 2017].

[4] "Save Data to MySQL," Dragino, 2016. [Online]. Available: http://wiki.dragino.com/index.php?title=Save_Data_to_MySQL. [Accessed 5 4 2017].

[5] "SimpleWebAPI," Android Authority, 2015. [Online]. Available: https://github.com/obaro/SimpleWebAPI. [Accessed 5 4 2017].

[6] C. Leeper, "GitHub," 15 04 2017. [Online]. Available: https://github.com/cleep55/RFIDtoDB/blob/master/RFIDtoDB.ino. [Accessed 19 04 2017].

[7] M. Balboa, "GitHub," 2017. [Online]. Available: https://github.com/miguelbalboa/rfid/blob/master/UNLICENSE. [Accessed 3 3 2017].