

**Department of Electrical & Computer Engineering
University of Central Florida**

**C.L.A.I.M
Computerized Luggage and Information Messenger**

**Group 10
Ernest Jackman - Electrical Engineer
Adrian McGrath - Computer Engineer
Tomasz Pytel - Computer Engineer**

Fall 2015

Table of Contents

1 Executive Summary	1
2 Project Descriptions.....	3
2.1 Motivation	3
2.2 Objectives and Goals	4
2.3 Project Requirements and Specifications.....	5
2.3.1 General Requirements.....	7
2.3.2 Hardware Requirements and Specifications	8
2.3.3 Software Requirements and Specifications	9
2.4 Constraints and Standards.....	9
2.4.1 Standards.....	9
2.4.2 Hardware Constraints.....	13
2.4.3 Software Constraints	13
3 Research Related to Project Definition.....	17
3.1 Existing similar projects and products	17
3.2 Relevant Hardware Technologies	18
3.2.1 Microcontroller Decision	18
3.2.2 RFID Transceiver.....	19
3.2.3 Wireless Communication Technologies	20
3.2.4 RFID Tag	20
3.2.5 Power Supply	21
3.2.6 Display	21
3.3 Relevant Software Technologies.....	21

3.3.1 Development Languages	21
3.3.2 Integrated Development Environments	22
3.3.3 Maven	23
3.3.4 Operating Systems	23
3.3.5 Miscellaneous Software Tools	23
3.3.6 Miscellaneous Web Tools	24
3.4 Miscellaneous Technologies.....	25
3.4.1 Tag Housing	25
3.4.2 Mounting.....	26
3.4.3 Device Housing	27
4 Project Hardware and Software Design Details	29
4.1 Overall System Block Diagram	29
4.2 Hardware Design	30
4.2.1 HF RFID Tag.....	30
4.2.3 RFID Transceiver	31
4.2.4 Wi-Fi Integration	32
4.2.5 Bluetooth Integration.....	32
4.2.6 Master Central Processing Unit	34
4.2.7 Power Supply	35
4.2.8 Monitor Integration	35
4.3 Software Design.....	36
4.3.1 Microcontroller Unit Software.....	36
4.3.2 Java Language Coding Standards	36
4.3.3 Abstraction Software	41

4.3.4 Main CPU Software	42
4.3.5 Android Stick Software	47
4.4 Miscellaneous Design	50
4.4.1 Tag Housing Design	50
4.4.2 Mounting Design	51
4.4.3 Device Housing Design.....	53
5 Design Summary.....	54
5.1 Hardware Design Summary	54
5.1.1 Parts List	56
5.1.2 Schematic.....	58
5.1.3 PCB Layout	63
5.2 Software Design Summary	64
5.3 Miscellaneous Design Summary	66
5.3.1 Tag Housing.....	66
5.3.2 Carousel Conveyor Mounting.....	66
5.3.3 Board Housing.....	66
6 Prototype Construction	67
6.1 Parts Acquisition	67
6.2 Prototype Board.....	67
7 Project Prototype Testing	69
7.1 Hardware Testing Environment	70
7.2 Hardware Specific Testing	70
7.3 Software Testing Environments.....	71

7.3.1 Emulators	71
7.3.2 Mocked Airline Lookup Service.....	71
7.3.3 Mocked Passenger Notification Services.....	72
7.4 Software Specific Testing	72
7.4.1 Microcontroller Testing.....	72
7.4.2 Main CPU Testing	74
7.4.3 Android Stick Testing.....	77
7.6 Project Testing	79
7.6 Measurable Test Results	81
8 Administrative Content	84
8.1 Project Milestones	84
8.2 Budget & Financing.....	87
8.2.1 Expected Costs	87
8.2.2 Financing	87
8.3 Advisors.....	89
8.4 Facilities & Equipment	89
9 Project Summary	90
10 Appendix	91

1 Executive Summary

Airport terminals are typically crowded in baggage claim areas and lack a certain ease of use as the passengers stand around anxiously waiting for their luggage to come out to claim. This group's project would help alleviate that burden by attaching RFID tags to luggage and scanning them when they are loaded onto the baggage carousel. Once the owner of the luggage is identified, they are notified via either a text message or an airline mobile application. Passengers can be alerted that their bag is available on the carousel without them having to check every individual bag as it passes by. A television display mounted next to the baggage carousels would also be used to show a real-time table of luggage currently being unloaded and placed in the baggage claim area, as well as the bags already found on the carousel. This would result in a more orderly luggage retrieval process, compared to the current chaotic process.

There are no products on the market which address the specific situation our group is addressing, however some similar products do focus on improving the airport's baggage handling system accuracy rather than improving the passenger user experience. Our project chooses to instead focus on enhancing the passenger experience while simultaneously providing tracking possibilities for the airport.

The goal of the system would be to read data from RFID chips located on tags attached to the luggage to communicate with the airline. The information for identifying the owner of the bag would be found on the RFID chip. The passenger's airline would then be asked to notify the passenger that their bag has arrived. This would be accomplished by a RFID transceiver scanning the RFID tags, and the data would be sent to a microcontroller which would in turn send it over to another device over a wireless connection. This new device would then communicate with the airline's web services to request that they notify the appropriate owner of the bag. The notification of the owner would be handled by the airline service, and the received data would then be transmitted to a connected monitor. This monitor is wirelessly connected as well and does display the real-time luggage data to passengers waiting in the baggage claim area. The system would be robust enough to handle communication with different airlines to accurately identify the luggage owners. This system would be mounted so as to utilize the empty space above existing carousel systems. The system is powered by a plug to an outlet in the setup environment.

Shown in Figure 1.1 is the basic operation the group has designed as our solution. At Step 1, the luggage is loaded onto the conveyer track by the airline's ground crew. At Step 2, the luggage is scanned immediately before it travels into the baggage claim area. The scanned data is sent to the Master CPU, as illustrated by Step 3. After the luggage is scanned, the television display is updated to include the newly scanned bag, as illustrated by Step 4. After the television is displayed, the owner of the bag is notified by their airline via a text message or a mobile app

notification that their bag is available on the carousel, as illustrated by Step 5. Lastly, at Step 6, the passengers picks up their luggage in the baggage claim area.

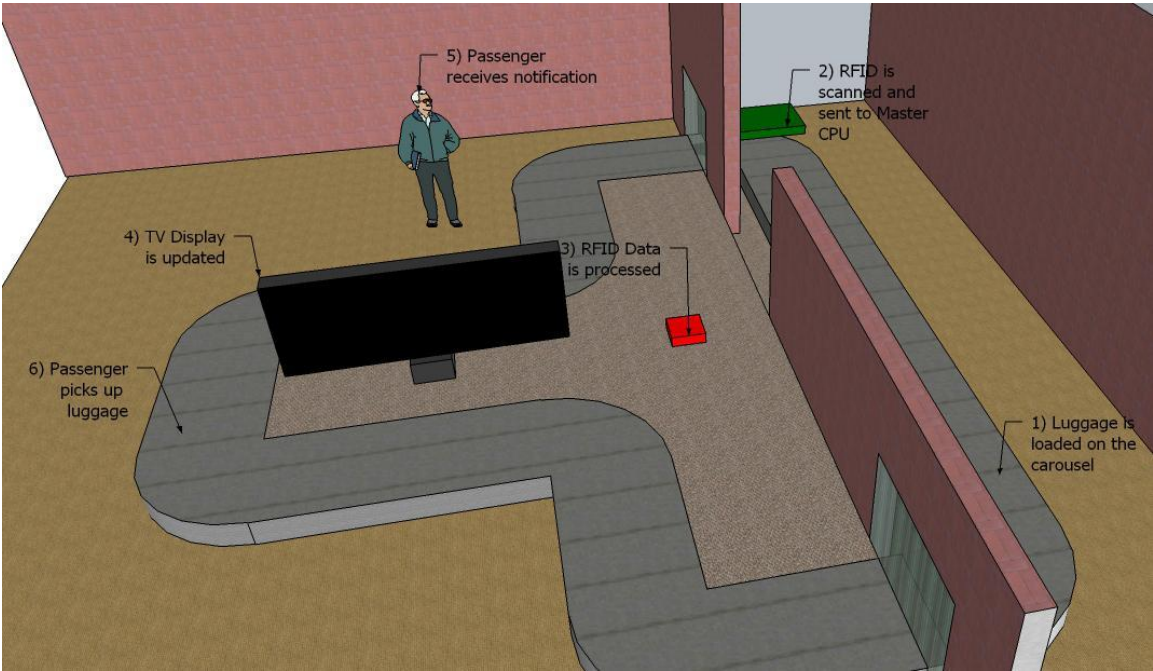


Figure 1.1 - Projected system mounting and process

2 Project Descriptions

2.1 Motivation

Noticing that the baggage claim system is one of the most crowded and unmanaged processes in modern airline travel, this senior design group decided to utilize RFID technology to find a solution to the problem. With the system in place, passengers would gain a certain ease of use in baggage claim areas when they receive a notification signaling for pick-up rather than crowding the airport carousels looking for their bags. They would be able to go about their business getting food, taking care of their children, making a call, etc., without worrying about whether their luggage has been offloaded yet. Not only would the passengers have less stress in the pick-up process, but the location of the bags could be determined if they were misplaced for some reason. Even though there are a few similar RFID systems being worked on, those systems are tailored more for the benefits of the airports and airlines rather than the benefits of the passengers.

By adding a new way of customer interaction and an up to date notification system, we provide a better experience at the airport for everyone. According to the Transportation.gov's May 2015 Air Travel Consumer Report, baggage is the 3rd highest category for consumer complaints [1]. As such, this device will lower and alleviate this kind of discrepancy. Providing this service for a severely outdated luggage claim system would further push innovation to the next level upon completion and yield positive results.

With the added benefit of creating a viable solution to this dilemma, the group was eager to learn about the different technologies and how they would work in conjunction with each other. Having known about the existence of RFID technology, the group was curious about how data was actually read from the RFID chips with the RFID Receivers. The Raspberry Pi was a tool some of the group members wanted to experiment with prior to the project, and found great use of making it relevant to the implementation. The group has had no experience working with wireless serial devices and was keen on how a wireless serial device piconet configuration can be implemented and how it would function. Building a microcontroller board and power supply was a great challenge to test the cumulative knowledge of the group's electrical engineers. The design process of a product from start to finish including the documentation, prototype and testing was a worthy learning experience for the group as well.

2.2 Objectives and Goals

The group's primary goal was to create a system that scans RFID tags mounted on luggage, identifies the owner, and notifies the owner while being robust enough to handle any number of airlines and work on many different carousel systems. This was accomplished by first researching the relevant technologies and then developing or utilizing them as needed.

The system was required be able to scan and decode rfid's accurately. This was accomplished by the power output of the RFID scanning unit and RFID tag size. These combined factors contributed to the overall range and accuracy of the system through use of either extended radiation range from the scanning antenna or larger receiving pad of the tag.

The system was required also perform with minimal delay. Proper communication of the serial data stream from the scanning unit to the main host controller. Since the main host unit was a primary wireless system, all data streams to the video display, airline, and RFID scanner were synchronized accordingly to minimize any delays inherent to the transmission of data along a serial line.

The system was required run on a continuous power supply. Due to the nature of an airport, regardless of whether the airport is regional or international, the system was expected to run for an entire day with little to no downtime. Some airports can have upwards of 150 flights at peak hours, which means at any time, our system needed to be operational.

A concern for the system was that it must comply with all health and safety regulations. The RFID scanner can have adjustable power output and needs to be adjusted accordingly to maintain a safe hazardous radiation pattern of less than 10 mW per centimeter squared area of radiation. Any more than this, and it fails government regulations for safe levels of hazardous exposure to human flesh. When presented in an industrial area, where those exposed are properly trained and warned, regulations are reduced to allow longer exposure amounts, so long as the area is properly labeled and employees informed.

This device also needed to comply with all environmental regulations. Radiation devices will not contain frequency above the visual light spectrum where ionization of the air may occur. This prevented harmful effects that may spark or burn objects passing under the radiation area. Additionally, no lead or lithium internal battery was used in this design that may cause fire or explosion. This device accepted any unwanted external voltage, and a proper fuse was in place for circuit protection.

The product is operational out of the box with some assembly to the mounting of the devices. Each module is self-contained in a box, with external plug connections as needed. This limited the installation process to mounting the scanning box and the processing unit to within range of each other for communication. An external

antenna to the scanning box was mounted to an overhang above the conveyer track, this allowed for a better scanning area, and provide some options for mounting of the device box, should the place of scanning not be near a power source. Distances between antenna and scanning unit, and scanning unit to processing unit is dependent on environmental conditions of interference and physical mounting constraints of the area the device is deployed in.

The cost of product was required to be reasonable, but the cost of production was highly dependent on the scanning device used, as higher frequency with better range and detection incur a larger upfront cost in chip/antenna design.

The system allowed free passage of all objects through the scanning area, while also being able to scan tags through thick objects. Based on the device chosen, the range and penetration of the scanner to detect tags was determined by power output, frequency, and local interference. Correct selection of a device was needed, so that it could scan far enough away to allow proper flow of objected underneath it. It also needed to not be overpowered or underpowered, to do the job it was designed to do. As underpowered would cause missed tag information, and overpowered could cause a health and safety hazard.

The system was required to operate with correct procedures for the access and use of personal information. The tag data was only collected and sent to the correct devices and authorities. The device was required to not randomly broadcast data or allow unwanted interception of unknown signals that could overwrite incoming data with false or unwanted information.

2.3 Project Requirements and Specifications

The group had to take into consideration several factors before the design process could begin. The Table 2.3.1 depicts the requirements that the project system needed to be tailored to.

<u>Required Specifications</u>		
Goals Met	Requirement	Justification
6	Must meet airport transmission standards	Regulation of frequency use and transmission of specific data from the RFID: ISO/IEC 18000-3 and ISO/IEC TR 24729-1:2008
1	RF Receiver must accurately scan the RFID chip 95% of the time	In order to ensure customer satisfaction
1,2	System must accurately identify who the luggage belongs to 95% of the time	So the system is able to contact the correct person
2,8	Must notify the correct recipient with a maximum delay of 60 seconds.	In order to ensure customer satisfaction
7,9	Must not exceed funds of \$2,000 to construct	So the system can sell for more than the cost of manufacturing.
3,8	Will use wall outlet to power scanners and microcontroller.	Must be able to run continuously.
4,5	Must not have any negative impact on the environment or personal	Do not want to cause any harm
10	Must meet size requirements to allow clearance for luggage	Clearance so all packages passing by the scanner goes unobstructed.
4,5	Must not emit any harmful radiation and is safe to handle	Do not want to cause any harm
11	Information the system transmits must be secure and not allow unauthorized access.	Secure information is required to prevent any unauthorized data retrieval

Table 2.3.1 – Requirements needed to be met by the project system

Goals and Objectives:

- 1) System was required to be able to scan and decode rfid's accurately.
- 2) System was required to perform with minimal delay.
- 3) System was required to run on a continuous power supply.
- 4) Required to comply with all health and safety regulations.
- 5) Required to comply with all environmental regulations.
- 6) Required to comply with all legislature and US laws and regulations as of 2015.
- 7) Operational out of the box with some assembly required.
- 8) Required to be able to operate 24/7 in an airport environment
- 9) Cost of product should be reasonable
- 10) Able to allow free passage of all objects through the scanning area.
- 11) Operate with correct procedures for the access and use of personal information

2.3.1 General Requirements

After identifying the base objectives and goals, the functionality of each component of the system was outlined.

Each RFID tag attached to luggage was able to be picked up by the RFID receiver. Due to the nature of the demonstration model system, and mounted receiver distance, the tag was scanned at the minimum distance of the high frequency system in order to be within the power transmission range of the system.

The RFID transceiver, the key component behind the RFID technology, was required to be able to scan a certain type or a variety of RFID tags for the data contents of each chip. Considering that the RFID chips can be passive or active, the group decided that having the RFID receiver configured to scan for passive chips would be the best option. RFID technology relies on the transmission of power to the RFID chips.

The microcontroller's primary function was required to include the need to receive and transmit the RFID tag data. The microcontroller received the RFID tag data from the RFID receiver, and awaited further instruction. The single board computer was required to send out a wireless serial device broadcast signal requesting the RFID tag data. After receiving the signal, the microcontroller was required to transmit the RFID tag data contents via a wireless serial device to the single board computer.

The single board computer was required to function as the central hub for the different components of the system. It was tasked with the handling of all data communications. It was required to utilize a wireless serial device to broadcast a data request signal to the microcontroller. After receiving the data on the same device, the single board computer was required to pass the data along to the mocked servers via Wi-Fi. The wireless serial device was required to then broadcast the new data to the Android Stick device.

The Android Stick needed to be able to receive data via a wireless serial device from the single board computer, prepare the data for display, and send the information to the TV to be displayed.

The mocked services simply needed to be able to receive data from the single board computer, identify the owner based on the RFID tag's contents, and notify the passenger who owns the scanned luggage. These services were required to be run by the airlines, and therefore the airlines are responsible for notifying the owner of the bag that the luggage has arrived on the carousel belt. It is up to the specific airline whether they wish to notify the passenger with a mobile application push notification, by sending a text message directly, or by using some other form of notification.

2.3.2 Hardware Requirements and Specifications

Required functions of the project include the ability to scan an RFID tag and wirelessly transmit this data to a receiving computer to process the information for display on a monitor and provide notification through text message. To accomplish this task, a radio frequency transceiver was required to read tag data. This data was then passed to a microcontroller unit, which was passed to the RFID information to a wireless broadcasting unit. The microcontroller also controlled the status and operation of the RFID scanner and wireless serial communication port.

Along with the communication node of the RFID scanner, a master node was needed for overall data extraction from each scanner; this master node was attached to the master unit, a single board computer. Further wireless communication was also required for connection with standard internet protocols on the computer side, in order to communicate with the host airline for passenger notification purposes. Other connections included video screen updates through hard wire or other wireless broadcast technology to provide visual confirmation of the tag data to the user.

Power requirements for this device were required to revolve around the need for a possible 24/7 operation with little to no downtime. It can be operated in both backroom industrial and front room commercial environments.

2.3.3 Software Requirements and Specifications

2.3.3.1 Microcontroller Specifications

The Microcontroller was required to be configured to send data via Bluetooth to the carousel's Single Board Computer.

2.3.3.2 Single Board Computer Specifications

The Single Board Computer was required to be configured to hold the host airport's IATA code, as well as the baggage carousel number the Single Board Computer is operating for. It was required to also be configured to have internet access, using an airport-provided wifi connection. The Single Board Computer was required to also be configured to receive incoming bluetooth connections from up to two Microcontroller Units.

2.3.3.3 Android Stick Specifications

The Android Stick was required to be configured to be connected to the Single Board Computer's Bluetooth access point. It was required to also be configured to run the specified software application in Single Application Mode.

2.4 Constraints and Standards

2.4.1 Standards

There were several standards that needed to be met by the project system. They are outlined in Table 2.4.1.

<i>Standard</i>	<i>Description</i>
ISO/IEC TR 24729-1:2008	Radio frequency identification for item management
ISO/IEC 18000-3:2010	13.56 Mhz
Title 47 - Chapter I - Subchapter A - Part 15 - Subpart C	Intentional Radiators
IEEE 802	local area and metropolitan area networks
IPC-2615	Printed Board Dimensions and Tolerances

Table 2.4.1 – Standards for the project system

Standard	Description
IPC-2612	Sectional Requirements for Electronic Diagramming Documentation
IPC-2221	Generic Standard on Printed Board Design
ISO 9000/9001	Quality Management
IPC-7351B	Generic Requirements for Surface Mount Design and Land Pattern Standards
IPC-4562	Metal Foil for Printed Wiring Applications
IEC 60695	Flammability Testing
IEC 60112	Insulation testing
ISO 527	plastic tensile properties
ISO 178	plastic flexural properties
ISO 8256	plastic impact strength
ISO 180	plastic Izod impact
ISO 179	plastic charpy impact

Table 2.4.1 *continued*– Standards for the project system

2.4.1.1 Performance

This device was required to be able to scan tags at a rate equal to conveyer belt speed. With an average speed of 90 feet per minute or half a meter a second, at least two to three tags per meter of track needed to be accurately sampled. A tag reading faster than one tag per second allowed for each tag attached to a luggage bag to be scanned as it rotates around the carousel.

This device also needed to communicate all tag data without drop of information when being wirelessly transmitting to each of the dependent substations for processing and display

2.4.1.2 Functionality

Required to scan RFID chip, transmit RFID chip data via query to airline database. The project was required to receive airline database information to display. This system should process information continuously, and show real-time data of recent scanned tag to a display.

2.4.1.3 Economics

Produced cheaply but with reliable and hardy parts that can last a long time, while still providing reliable consistency in detection and system stability. The project was required to be economically viable to implement in real world applications.

2.4.1.4 Energy

The primary power source was a five volt input, the small scanner chip was regulating its own 5 to 3.3 volts, and the Bluetooth sampled off a regulator chip. Maximum current draw of the scanning chip was 150 mA, with the Bluetooth requiring close to half the amount for continuous operation at 65 mA. Combined current draw from the system was required to then be at the most 215 mA. The microcontroller was required to be running off the internal regulator of the scanning chip. Since the msp430 in high frequency operation only drew up to 4 mA, the internal regulators 20mA draw limit was enough to support this device, and was within the max current draw of the primary input of 150mA.

2.4.1.5 Environmental

ABS plastics used in the construction of the circuit board project cases are known to release harmful chemicals when subjected to high temperatures. Under normal room temperatures no known harmful effects are released into the environment.

2.4.1.6 Health and Safety

Radiation from the scanning unit is regulated to 10 mW per cubic centimeters of exposed flesh to prevent radiation burn from energy propagation. Inhalation of toxic fumes from the project cases should they be exposed by high thermal temperature also provided a health hazard if directly inhaled while melting or on fire. Shock hazard can occur under certain conditions if any of the power lines become damaged or shorted.

2.4.1.7 Legal

The system queried the airline database but never directly access it. The system did not impede the security of data being transmitted or received. The project did not infringe on any copyright laws. The project did not breach any data security.

2.4.1.8 Maintainability

Ability to run 24/7 with little to no need for external contact. Minimum maintenance of dust removal, and proper maintenance of power cables.

2.4.1.9 Manufacturability

Due to the simple single board design, the scanning unit was able to be manufactured and installed into a simple case. This case could then be mounted into a multitude of positions, with the external antenna being the only device that needed to be close to the conveyer system. The single board computer for the main processing was produced by another company for creation and programming of the board. Each unit once manufactured was installed by mounting directly to an available surface near its respective peripheral.

2.4.1.10 Operational

It was required to operate within airport, commercial and industrial sections. Where the industrial section needed to be able to handle possible weather conditions if located close to, or directly in outside temperatures/conditions. In the commercial section, standard room temperature operation of the central processing unit or lower would be ideal for the operation of the single board computer.

Potential impacts on the device include extreme conditions of temperature, power surges, high impact forces, potential flooding dependent on conveyer positioning with ground floor.

2.4.1.11 Reliability and Availability

System was required to be reliable in its ability to scan every tag presented under the scanner and process the tag information in an accurate and timely manner. The display information was required to also be displayed and updated correctly. The system was required to send out a notification upon the luggage bag arrival.

2.4.1.12 Social and Cultural

Promptly notify passenger when their luggage was scanned and identified to provide better time management for the passengers. This allowed the passengers the freedom to buy gifts, talk with family, and perform other activities at their leisure instead of waiting at or crowding the conveyor carousel systems.

2.4.1.13 Usability

For physical mounting of the devices, each unit was required to be encased in an enclosure that can be mounted directly to a flat structure. Each device was wireless, and allows for placement in a variety of locations.

The system was required to be easy to assemble and get operational out of the box. Simple clamp/bolts, followed by plug and play operation of the software.

The software was required to be easily updatable with customer information to connect to their respective devices for external communication.

2.4.2 Hardware Constraints

Power out of the device was regulated by government standards as a health and safety factor. As such this device must be constrained while still able to scan through luggage without causing harm to workers nearby.

Placement of the scanning antenna was limited by airport construction and conveyor design, ability to allow luggage to pass freely underneath with no obstruction is a mandatory requirement.

2.4.3 Software Constraints

2.4.3.1 Compatibility Constraints

2.4.3.1.1 Java ME Embedded Constraints

The Java ME Embedded 8.1 Runtime Environment is a small subset of the Java Standard Edition. It does not contain nearly as much functionality as the Java Standard Edition, so knowing what functionality is not provided was important for development. It is also important to note that any third-party libraries which require the use of packages not available in Java ME Embedded 8.1 were incompatible for our system. It was necessary to use Java ME Embedded 8.1 instead of the Java Standard Edition because of the Single Board Computer's limited resources.

Java ME Embedded 8.1 comes with many of the basic packages from the Java Standard Edition. Most noticeably, all graphical packages are not available. They are not available because Java ME Embedded 8.1 is headless, so the group was unable to create a graphical user interface for the Single Board Computer. The package for input and output is greatly reduced, as Java ME Embedded only supports input and output with basic data streams.

Unlike Java Standard Edition, the language package is trimmed down and doesn't contain some classes necessary for reflection. The math package from the Java Standard Edition, which contains a few classes for managing numbers larger than the maximum values for integers and doubles are not available in Java ME Embedded. The net package, which contains many of the classes necessary for network communication is also gone. Our group needed to use the Oracle supplied HTTP Client API to gain access to the functionality needed to communicate with the host airline's web services.

The Remote Method Invocation package was not available, and the security package has extremely limited existing functionality. Depending on our security needs, the group needed to use one of Oracle's security APIs. The group also didn't have access to any of the sql, xml, or time packages. Several APIs exist for some of these functionalities, which the group needed to use.

2.4.3.1.2 Single Board Computer Configuration File

The Single Board Computer was in charge of knowing which Microcontroller Units and which Android Sticks it can communicate with. It also was in charge of knowing the security credentials it needs to access the airport's wireless network. This information was determined by a configuration file located in the Single Board Computer. This file needed to be configured by the customer for each individual Single Board Computer. The Single Board Computer program does not have user interface capabilities, and the Android Sticks the Single Board Computer were required to be configured before they could be used, the customer was constrained with how they are able to modify the configuration file. In order to modify the configuration file, the customer needed to remove the physical SD Card from the Single Board Computer and plug it into a compatible computer with a monitor and keyboard. The customer then needed to run the ConfigurationWizard file on the SD Card and go through the wizard to configure the various settings and devices.

The customer needed to configure the Bluetooth devices in the configuration file. In the wizard, the user should have entered the Bluetooth device UIDs found in the packaging for the individual scanner and Android Stick units.

2.4.3.2 Single Board Computer Operating System Constraints

The Single Board Computer was required to run an operating system capable of utilizing the Java Embedded Micro Edition 8.1 Runtime Environment. Ideally, the Single Board Computer should run, at minimum, a version of the Embedded Linux series.

2.4.3.3 Customer Required Software Implementations

2.4.3.3.1 Overview

The customers for the group's product (i.e. Airlines) were expected to implement some web services on their systems. The customers were responsible for notifying the passengers that their bags are available. By allowing the customer to implement this functionality on their own, the customer was able to choose which method of notifying the passengers they feel would be best. Some customers may choose to notify the passengers via a text message to their cell phone, while other customers may prefer to implement the notifications in their existing mobile applications.

2.4.3.3.2 Implementation Requirements

The implementation of the notification system was required to be a RESTful web service which accepts json payloads. An example of the incoming payload can be found in Figure 2.4.1 below.

```
{
  "identification": "bagID",
  "flight": "flightID",
  "arrivalAirport": "airportIATA",
  "carousel": "carouselNumber",
  "time": "scanTime"
}
```

Figure 2.4.1 - Incoming json payload example

The *identification* value contained the baggage identification information which was provided to the system when the bag was originally checked in. The formatting for the *identification* value depended on what the customer originally provided. The *flight* value contained the flight identification information which was provided to the system the bag was originally checked in.

The formatting for the *flight* value depended on what the customer originally provided. The *arrivalAirport* value contained the arrival airport for the bag which was configured on the carousel's CPU. The formatting for the *arrivalAirport* value was the three-letter IATA code for the arrival airport. The *carousel* value contained

the number for the carousel the bag was scanned on, which was configured on the carousel's CPU. The formatting for the *carousel* value depended on what the customer configured for the CPU. The *time* value contained the UTC time for when the bag was scanned. The formatting for the *time* value was the combined date and time ISO 8601 extended format.

3 Research Related to Project Definition

3.1 Existing similar projects and products

Even though RFID technology stems from the 1980s, it has not been as widely or prominently utilized in modern times as one would expect. There are several noteworthy products that are similar to this group's project solution. However, even though there are a few similar systems being worked on, those systems are tailored more for the benefits of the airports and airlines rather than the benefits of the passengers. This group planned to focus on a solution more suited for the passenger's benefit.

The Hong Kong International Airport (HKG) has started using only RFID Baggage tags as early as 2009 for all of the bags that leave the airport every day. This RFID system has been proven to yield an improved handling capacity compared to the standard paper tag bar code system used in place today [2]. These RFID tags are encoded with unique ID numbers, 3 letter IATA code describing the destination airport, as well as the date and flight number. This specific ID number links to the back end baggage handling systems the passenger's information.

Air France-KLM launched an eTag in 2014 which included two e-ink displays. Partnered with the eTrack system which makes use of GSM, GPS, and Bluetooth technology enables the luggage to be easily tracked by any smartphone. This system was designed to benefit the airlines as well as the passengers and allows linking of different luggage bags to an account which can then be used for a streamlined check-in process [3].

A company in the United Kingdom by the name of ReboundTAG has created permanent RFID luggage tags for airlines [4]. This tag has two microchips, where one tag is used for identification, which remains permanent and can be used to help automate the check-in process. The other one is re-writable so that airlines can write new flight information on each trip. This tag can also be tracked with Worldtracer to locate the luggage and identify owner of its whereabouts. This is a pretty similar product to the group's approach, however it includes an additional tag that houses the passenger's personal information. For security reasons the group decided to limit the tags to one microchip and be re-writable so they could be reused as a smarter economical option.

Vanguard ID Systems has created a different permanent RFID luggage tag called the ViewTag [5]. A prominent feature of this tag is its functionality with the owner's cellphone, in which the location of the luggage can be checked using the tag's QR code or NFC embedded module housed in the tag itself. The ViewTag utilizes Bluetooth and an RFID reader located on the loading belt of an airplane. After scanning the tag, the luggage owner receives a confirmation message, resulting in updates of the location on-board flights, connecting flights, and final destinations.

Qantas, Australia's largest airline, has developed the Next Generation Check-In program, or NGCI for short, which uses RFID technology to simplify the check-in process and essentially allows for self-service [6]. Using the Q Card, passengers may check-in simply by touching it against a Q Card Reader, after which they receive a confirmation message and the Q Card becomes their boarding pass. The Q Bag Tag is simply an alternative to the normal paper tags and has the usual data and barcode on it.

3.2 Relevant Hardware Technologies

A microcontroller was required to convey information from the RFID scanner to the wireless transmitter for external processing of the data. This chip passed information and handled all status and command issues for the control and error handling of the RFID scanner and wireless communication.

A RFID transceiver was used for powering the passive tags and receiving their stored data. A wireless communication chip was used for transmitting data gathered by the RFID transceiver to the base master unit.

A single board computer was used as the master unit for controlling the slave scanner units containing the RFID scanner, controller, and wireless communication device. A WiFi transceiver was used for relaying decoded information on the master unit to external sites for text message service. Another wireless unit was used for updating an external monitor with decoded information for user display.

3.2.1 Microcontroller Decision

The microcontroller needed to have the ability to pass data from the RFID scanner to the wireless communication chip. This was accomplished through either a parallel or serial communication between the chip and the device it is going to communicate with.

On top of the transfer of the data from one device to the next, the microcontroller also needed to be able to process any error and status messages flagged by the RFID scanner of the wireless serial device. Most of these functions can be accomplished through the use of LED lights that can be attached to the microcontroller and programmed to indicate status values.

It also needed to accept commands being sent to the serial device and properly send these commands to the RFID scanner to control scan power, on/off status, and any other status and setting requested by the master unit. This needed to be handled by the communication set up by the devices for bidirectional communication.

3.2.2 RFID Transceiver

The RFID reader/writer was required to have enough power to scan through luggage and provide proper power out to achieve the needed minimum transmit power to the RFID tag.

The larger the RFID scanner is, in regards to power out, the longer the range of the system. This is due to degradation of the radio signal through free space, limiting the scanning distance. Another factor is the frequency of the transceiver: the lower the frequency, the longer the wavelength and the better the penetration through surfaces, but at a reduced peak power per cycle. So a shorter wavelength offer more power per cycle, but the higher frequency wavelength will suffer less penetration through thicker materials.

An example of an UHF RFID scanner is the ThingMagic board as shown in Figure 3.2.1, where a manufacturer has constructed and tested with specified antennas to be properly tuned and regulated by government standards for the long range radiation for tag scanning.

The size and power of the RFID transceiver will be dependent on the location and any obstruction it will need to scan through.



Figure 3.2.1 ThingMagic Embedded RFID Scanner

3.2.3 Wireless Communication Technologies

Two types of communication were required for this project, a serial communication and a WiFi communication. The primary communication was serial data transmission from the RFID scanning unit to the master unit, and another serial communication line from the master unit to the monitor. The other type of communication was for integration to a standard WiFi router for internet or local database connection to transmit needed information for the text message notification system.

3.2.4 RFID Tag

The RFID tag to be used in the system needed to be a passive RFID tag programmed with related information provided by the airline for display and text message notification. The size of the tag determined the range in which it may be powered from. The larger the tag, the more surface area the antenna has. This larger antenna provided a larger surface area for transmitting RF power across the coils of the antenna powering up the RFID chip. Once powered, the antenna was then used to broadcast the encoded data stored within the chip to the receiving unit.

Example tags come in many shapes and sizes, as shown below in Figure 3.2.1. The tags can come in a multitude of sizes of square and rectangular shapes, along with circle shapes to match whatever object it is to be attached to.

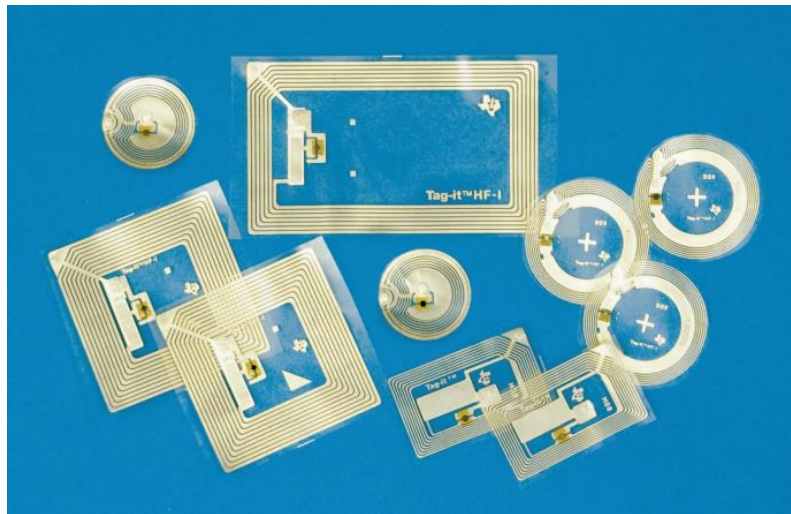


Figure 3.2.1 - RFID Tags
“Reprinted with permission from Texas Instruments”

3.2.5 Power Supply

The power supply needed to provide enough power to run the transmission side of the RFID scanner. Depending on the RFID scanner used, this can be upwards of five watts of continuous power out on high end devices, to a few milliwatts on smaller devices. This meant that if the system was to be operated on a 24/7 basis, the required power for transmission of power to the RFID tags would require a constant power supply. The master central processing unit also needed to be operating multiple wireless transceivers. It was not provided much downtime in its operation while the system was on, as the master unit gave commands to the slave scanners and gathered information to be processed from them.

3.2.6 Display

A monitor or television compatible with the current technology in wireless video transmission, either internally, via a network connection to a server, or externally, through video codec devices. The display needs to be updated with real time data provided by the master unit on each tag scanned. This display needs to be large enough to accommodate font sizes where all personnel viewing it will be able to see the displayed information from a distance, and sturdy enough to be present in a commercial environment.

3.3 Relevant Software Technologies

3.3.1 Development Languages

3.3.1.1 C

The C language is basic enough for controlling data flow and managing status information. For this project, the C language was ideal for use in the RFID scanning unit.

This program was used in the microcontroller units. It was developed to be able to take the raw data from the RFID Receiver and send it via the wireless serial device to the master central processing unit. The program also needed to accept data from the serial communication device in order to control the operation of the RFID scanning chip, and send status information back to the processing unit.

3.3.1.2 Java

While the C language will work on an embedded system of small size, the large amount of data handling and the need for the communication synchronization of multiple devices at the same time required the use of a more robust operating

system and programming language. The Java Programming Language is able to be tailored to run specific devices while allowing a broad range of programming to send the data between the interconnected devices. The devices included the master central processing unit, and the wireless broadcast unit for updating a monitor with tag information.

3.3.1.3 PHP

The mocked web services, which were used for testing and demoing, were developed in PHP for simplicity. PHP is a server-side programming language, and one of the most commonly used languages for implementing quick, basic, and session-less RESTful webservices. While PHP would most likely not be robust enough for a commercial implementation, it was more than adequate for the needs of the group's testing environment.

3.3.2 Integrated Development Environments

3.3.2.1 Eclipse IDE

The Eclipse IDE was used as the group's Integrated Development Environment for the Java Software, the Android Software, and the PHP Software. This Integrated Development Environment allowed the group to develop our large range of languages on a single platform. Eclipse also allowed the group to run emulators and virtual machines directly within the IDE. Eclipse allowed the group's developers to configure coding standards, which could be shared by exporting and importing various configuration files, and it had full support for the group's source control needs. Eclipse has extremely detailed support for Maven and the Android SDK, making it an ideal candidate for the group's needs. Since it is such a commonly used IDE, finding tutorials and assistance for most portions of the project was easy to do.

3.3.2.2 Code Composer Studio

Code composer offers support across all Texas Instrument devices, and is available for free or limited use for most all of their device chips. Using this software allowed for not only the writing but step by step debug control of each relative debug board provided by the company.

3.3.3 Maven

Apache Maven was used as the build automation tool for the Master CPU's software, as well as the software for the wireless broadcasting of monitor information. Maven allowed the group to manage library dependencies easily between our developers, because of its standardized configurations and because it automatically downloads the specified versions of dependent libraries for our developers from the Maven 2 Central Repository. These automated features greatly reduced potential mistakes our developers may have made during builds and source control. The build tools within Maven also allowed the group to configure our unit tests to run prior to build completion. If any of our unit tests in the system failed as a result of source code changes during the build, the build failed and we were notified of what tests failed. Configuring all of our unit tests to automatically run as a precondition for build success meant we could greatly reduce any breaking bugs which may arise due to source code changes.

3.3.4 Operating Systems

Having never worked with linux based boards before, the group had a wide variety of operating systems to choose from to utilize. The top few that were up for consideration for their respective abilities were all Linux based.

The OpenELEC operating system and RaspBMC are tailored to have straight out of the box media center operating system adaptations. Since the group did not require this functionality for the project system, these two choices were promptly discarded immediately.

The Pidora operating system has been known to have some glaring issues and even with several versions of bug fixes, there was still some negative feedback on the system. In order to avoid any potential future problems that at first glance may not have seemed relevant, the group chose to also abandon this as an operating system choice for the Raspberry Pi.

Having narrowed down the flavors of Linux by a margin, the two remaining options that supported our features were ArchLinux and Raspbian. These two operating systems are pretty similar, however one major difference is that ArchLinux boots directly into the command line, while Raspbian has some graphical interface. Many tutorials to projects about functionality and operation have plenty of support with the Raspbian operating system that would bring an additional benefit. The specific operating system to be used was Raspbian Wheezy as it has been confirmed on multiple sources to work with the Raspberry Pi single board computer with the group's planned implementation.

3.3.5 Miscellaneous Software Tools

3.3.5.1 PuTTY

PuTTY was used to interface with the group's Master CPU via a serial connection. PuTTY allowed the group to use Secure Shell to communicate with our Master

CPU via a USB/TTY cable connected between our development computers and our Master CPU. By interfacing with our Master CPU this way, the group was able to easily transfer the program files to the device, as well as configure the device according to our needs.

3.3.5.2 Eagle

Easily Applicable Graphical Layout Editor was used to help design and plan out the necessary hardware components, including the schematic diagrams for the individual components and the printed circuit board layout. This software allowed detailed placement and reverse operation between the schematic and printed circuit board layout for full control of part placement and proper wiring connections.

3.3.5.3 Umlet

Umlet was used to help design and plan out the necessary software components including the block diagrams, UML diagrams, and flowcharts. Umlet provided the group with simple, yet robust, control over the designs and layouts for our many diagrams and flowcharts. Umlet also had the added benefit of being a plug-in for Eclipse, our chosen Integrated Development Environment.

3.3.5.4 Google SketchUp

Google SketchUp was used to help design and plan out the necessary mounting components, and to help illustrate the process of the system's functionality as a final working product. Google SketchUp allowed us to create three dimensional models of the baggage claim area, so the group could figure out how to correctly mount our various devices, as well as help the group sort out some of the logistics in a way two dimensional images would be insufficient.

3.3.6 Miscellaneous Web Tools

3.3.6.1 Repository

Assembla was used to host our software repositories for source control. This repository allowed the group to perform Subversion for version control, which allowed independent work on individual portions of the system and merged them all together without creating file conflicts. Assembla allows a free tier Subversion repository, which is ideal for the group's needs.

3.3.6.2 Issue Tracking And Agile Manager

Yodiz was used to host our issue tracking and agile management. This website allowed us to create several agile sprints and add as many tickets as we would like to each sprint. The group was able to plan the development of our system

several months in advance. Group members were able to create bug tickets when necessary, and all of the status information for the project was easily viewed on Yodiz. Yodiz's free tier plan allows for a maximum of three users, which is ideal for our group of three engineers.

3.3.6.3 Mocked Web Services Domain

LivingBucket.com is the web domain which was used as the host for our mocked web services. This domain allowed us to upload the PHP scripts which served as our fake airline lookup system and our various fake airlines. This domain was required to be interacted with directly by our Master CPU. The LivingBucket domain was chosen because it is currently owned by one of the group's members and has guaranteed uptime, so the group did not need to find a free domain which could have the potential for uptime issues throughout development and demoing.

3.4 Miscellaneous Technologies

3.4.1 Tag Housing

There were several simple methods of attaching tags to luggage bags that the group looked into. When first researching RFID chips and related devices, the low frequency chips were found to be used most commonly for door locks. The "tags" for these were just simple door key cards the size of a credit card, though keychain variations also exist. Other possible options for tag housing included a laminated or plastic sleeve similar to current information tags some passengers put on their luggage bags.

However, in order to relate to the luggage bag scenario, the group looked into more conventional means currently being used at airports in modern times. Existing systems use a sticker tag with text information and a barcode printed on the paper strip. Using a commercial printer device with RFID embedded in the printer roll, these types of tags can be created with the RFID chip printed onto the paper strip. These tags can be mass produced with the most economical savings compared to the other options, considering the scale of the intended product to be used. A printer unit can be used like from the Zebra RFID Company, the tags contain a RFID chip and printed to work the same way as normal luggage tags, maintaining the older system of baggage claim tracking for any systems still in place that need it. A picture of an RFID printer distributing a ribbon of printed paper with embedded tags within it is shown in figure 3.4.1.



Figure 3.4.1 RFID Printer

3.4.2 Mounting

The scanning device and master unit are mounted onto any flat surface that ensured it is out of the way of the local operation of devices.

For the scanning device, the antenna needed to be set up in a way that it pointed towards the tags that needed to be scanned, while not obstructing the flow of objects on the conveyer system. A simple L bracket next to each other assisted in the creation of a platform for the antenna. The antenna was connected to the scanning unit that is mounted along the wall a couple feet from the antenna itself. Both these units were required to be connected through a coax cable.

The master processing unit, placed in the main passenger area, is mounted out of the way, in a location close to the scanning unit for proper communication of its data. The television also needed to be mounted within the vicinity so the master processing unit can communicate all video information of the tag data to it.

A fully mounted system would look like Figure 3.4.1 below, indicating the placement of the RFID antenna, RFID scanning box, Main processing unit, and Display. Since each system is modular and wireless, other than the need for power, each unit can be smartly places in any location within its designated contact range with its other units.

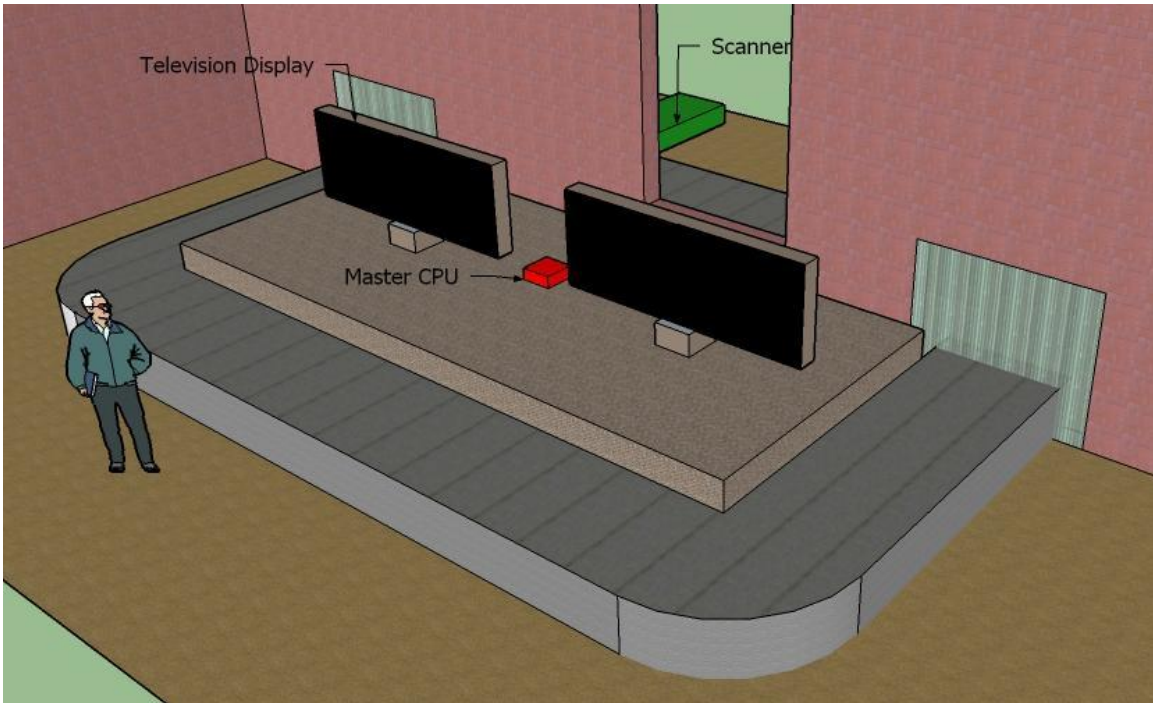


Figure 3.4.1 - System Layout

3.4.3 Device Housing

The main devices of the system, the scanning unit and the master unit, were both required to be encased in some type of protective housing, most likely plastic. The scanning unit casing needed to allow heat dissipation but also be sturdy and rugged enough to withstand industrial location environments, as it was required to be mounted in the back of the carousel system. It required holes to allow for connection to power as well as the external antenna attachment. A primitive project case can be seen below in Figure 3.4.2 that can be used as a base. The manual drilling of required holes was necessary to allow mounting of a PCB board within, as well as any additional components the project demands. These housing cases were available in a large quantity of dimensions and styles and the best fit was chosen for the project.



Figure 3.4.2 – Scanning unit enclosure.

The master unit needed to have a more aesthetic housing since it was to be located in a commercial area. Its ruggedness did not need to be as hardy as the scanning unit, but needed to be able to be mounted at multiple positioning to accommodate a location that is near a monitor for data broadcasting. Figure 3.4.3 below shows one of many available cases designed for a prototyping single board computer, which come in a wide array of colors and plastics.



Figure 3.4.3 – Master unit enclosure

4 Project Hardware and Software Design Details

4.1 Overall System Block Diagram

The overall system, shown in Figure 4.1.1, is operated from a Master CPU which controls each peripheral device. The Master CPU handles device communication, monitor updating, and sending information to the host airlines. The host airlines are in charge of notifying the passengers that their bag has arrived on the carousel. It also commands the RFID scanning unit to turn on or off depending on the status of the system. Interconnections between each device to the master CPU is handled by a Bluetooth serial communication device.

For the CPU, the Raspberry Pi functions as the high end service to manage the notification system of text, monitor, and airline. Communication of these services are handled through the use of the Bluetooth, a standard WiFi, and wireless video transmission through an Android system.

For the scanning unit, the RF transceiver is handled by the TRF7970 microchip, and controlled by the MSP430 microcontroller. Utilizing a HF frequency antenna, hard connected with a SMA and UFL connection to the PCB, to extend the scanning range of the device.

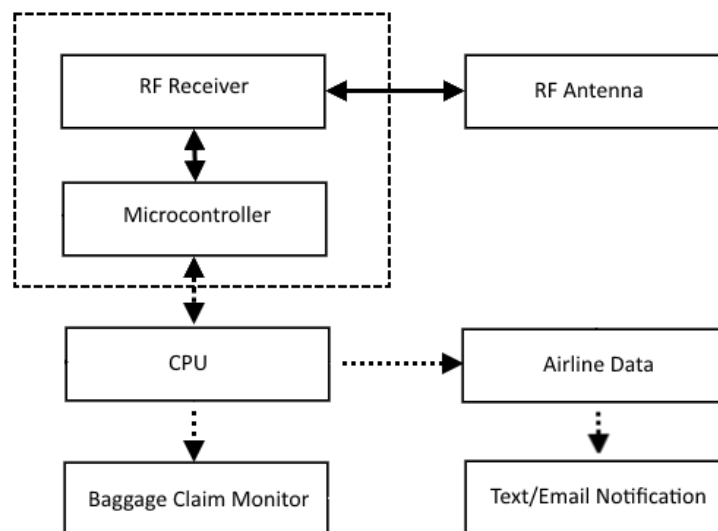


Figure 4.1.1 - System Block Diagram

4.2 Hardware Design

4.2.1 HF RFID Tag

RI-I02-112A-03, as shown below in Figure 4.2.1, is a large rectangle Tag-it™ HF-I Plus Transponder Inlay by Texas Instruments. It is a global ISO 18000-3 compliant, ISO/IEC 15693, passive 13.56 Mhz tag, containing 2kbit of user programmable memory with factory installed Application Family Identifier (AFI) and Data Storage Format For this project the MSP430G2553, a dip style chip was used for ease of programing and testing, it can run on 3.3V source and contains a Universal Serial Communications Interface (USCI). This serial communication can be split into two serial sections of portA and portB. Each port can support Universal Asynchronous Receiver Transmitter (UART), Serial Peripheral Interface (SPI), and Inter-Integrated Circuit (I2C). Since both ports can only support a single SPI or I2C at one time, the microcontroller is set up with a single UART to the Bluetooth serial communication device, shown below in Figure 4.2.2, and SPI to the RFID scanner, shown below in Figure 4.2.3.

The MSP430 also controls the command lines of the RFID scanner and monitors status information, relaying this information back to the master unit to allow remote operation of the RFID scanner. For future productions, the MSP430F type chip would offer surface mount, and space saving features. It can be implemented as a final production chip as it contains the same basic operational pins for serial communication.

For status information and tracking of program operations, a LED light was attached to a spare general input/output pins for troubleshooting purposes. This light was used in order to indicate when a tag is scanned and/or a heartbeat indicator to make sure the program is not stuck in an internal loop.

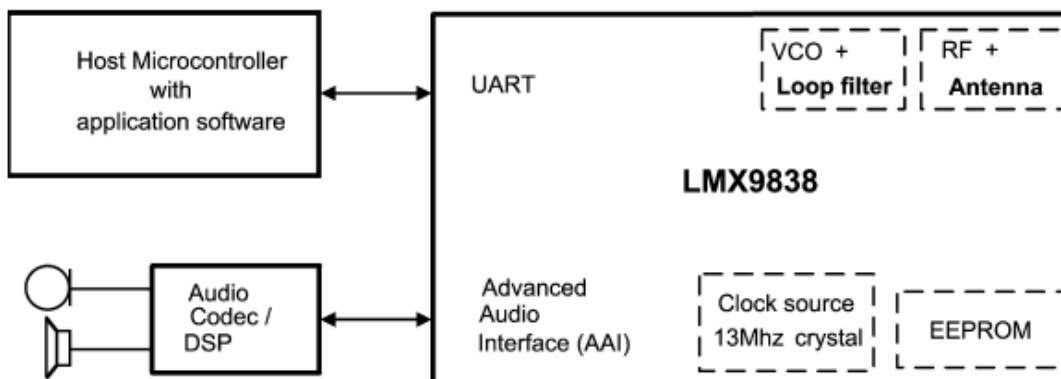


Figure 4.2.2 MSP430 to LMX9838 UART Block Diagram
"Reprinted with permission from Texas Instruments"

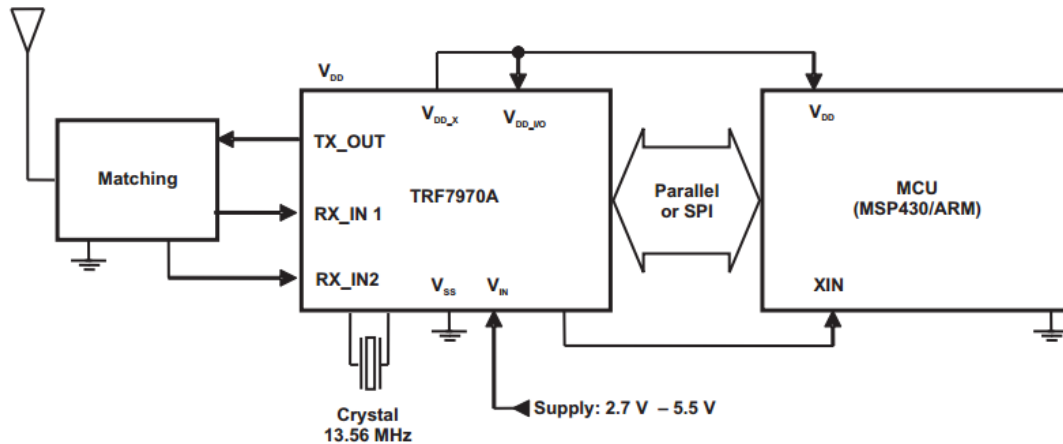


Figure 4.2.3 MSP430 to TRF7970A SPI Block Diagram
 “Reprinted with permission from Texas Instruments”

4.2.3 RFID Transceiver

TRF7970A, shown below in Figure 4.2.4, is a multi protocol integrated chip for encoding and decoding of ISO 18092, 21481, 15693, 18000-3, 14443A/B, and FeliCa protocols. It operates at a minimum of 2.7V to 5.5V for full power operations. The power out can be set to 100 mW or 200 mW on the transmission side, where the later requires a minimum of 5V input to achieve. In this design the 5V input, so a 200mW output was picked for maximum scanning range. The chip also includes an internal voltage regulator with a 20 mA max current draw and supplies power directly to the MSP430 microcontroller for power operation on a 2.7V to 3.4V range. The RFID scanner is able to support SPI or an 8-bit parallel output. For this design the SPI connection was used to the MSP430 for command and control instructions. To implement the SPI connection I/O connection of the parallel pins 0-7 as indicated below in Figure 4.2.4, are adjusted to indicate SPI mode. This is done by providing a high voltage on pins 1 and 2, and a ground on pin 0. During the initial startup of the RFID chip, the SPI protocols is selected, providing serial data flow out of I/O pin 6 and 7, and flow control off I/O 4.

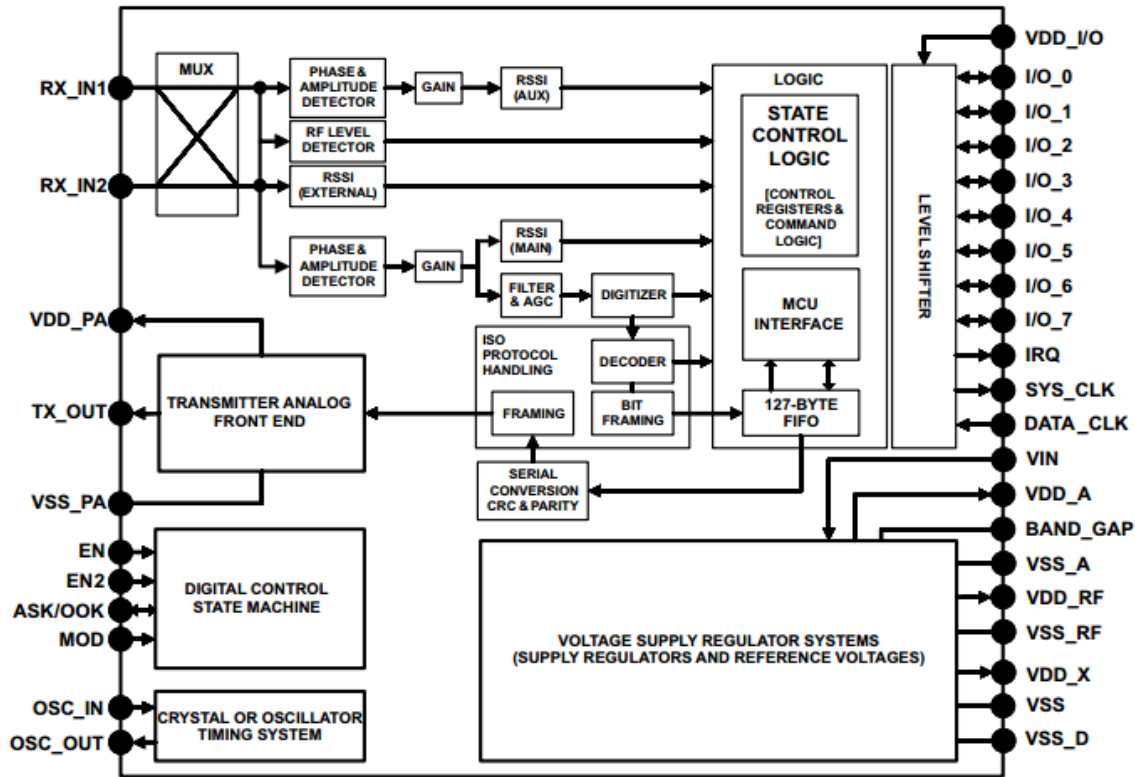


Figure 4.2.4 TRF7970
 “Reprinted with permission from Texas Instruments”

4.2.4 Wi-Fi Integration

Edimax EQ-7811Un Wi-Fi USB adapter is a 802.11b/g/n standard wifi transceiver with up to 128 bit WEP,WPA1/2 encryption. It is a small USB connected device with a small size of 7.1x14.9x18.5mm attached to the master unit for connection to the internet, or local network, in order to facilitate the notification of members when tag data is extracted and passenger notification is required to be sent.

4.2.5 Bluetooth Integration

TRENDnet Micro-Bluetooth TBW-106UB v2, shown below in Figure 4.2.5, is a small USB connected device of size 20x12x5mm, using Bluetooth 4.0 protocols, it is used to communicate and control the RFID scanner by sending data to and from the scanning unit for monitoring and controlling the on status of the device. The Bluetooth adapter can be set up as a master node and is able to control multiple scanners if the system requires a larger system installed.

The Bluetooth device on the scanning unit is the LMX9838 serial port module,

shown below in Figure 4.2.6, it is a Bluetooth 2.0 protocol device, and communicates with the MSP430 through the use of UART. It is an integrated chip with internal antenna. This device is used as a slave device to the master node for the retrieval of the data stream from the RFID scanner. It is connected to the MSP430 through the UART connection for data gathering from the RFID scanner. Further status controls are controlled by the MSP430 along with status LEDs attached to the general purpose input/output pins of the Bluetooth device to indicate transmission and reception of data through the device.



Figure 4.2.5 - Micro-Bluetooth and WiFi Adaptor

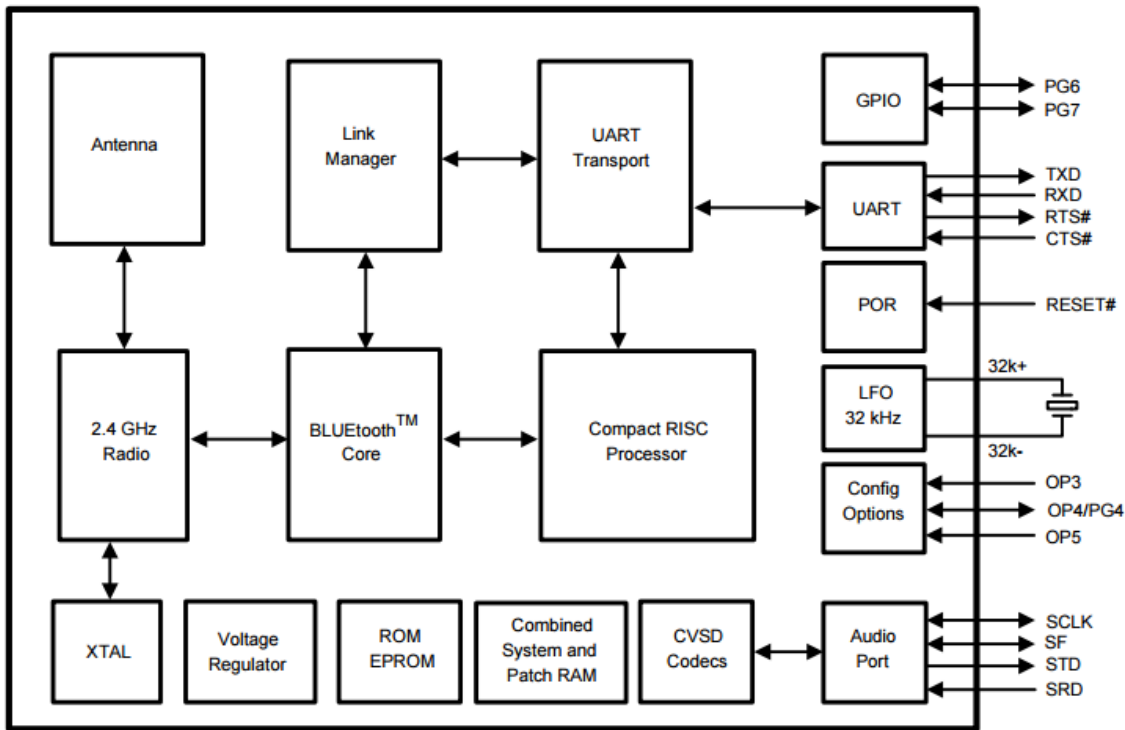


Figure 4.2.6 - LMX9838 serial port module
 “Reprinted with permission from Texas Instruments”

4.2.6 Master Central Processing Unit

Raspberry Pi 2 Model B is a single board computer using a 900MHz quad-core ARM Cortex-A7 processor and is used as the master unit for command and control of the slave RFID scanner, as well as a data extraction device for all the tag information being provided by the scanning unit. All data extracted from an activated tag, is used to update a video display and provide a message to a text messaging service. The Raspberry Pi 2 Model B has 1GB of RAM, four universal serial bus ports, forty general-purpose input/output ports, a full high definition multimedia interface port, an Ethernet port, a combined 3.5mm audio jack and composite video port, a camera interface, a display interface, a micro secure digital card slot, and a VideoCore IV 3D graphics core. The Raspberry Pi 2 Model B was chosen because of its large range of capability, its compatibility with the group’s chosen development language, and its competitive cost. The group felt confident in the Raspberry Pi 2 Model B’s ability to perform all of the tasks it will be assigned to perform for the success of the project.

4.2.7 Power Supply

Generic 12v power plug with matching power plug adapter was used in final design of the PCB layout and demonstration build. The power supply is able to provide enough current to support both the bluetooth, and NF RFID chip. The 12 volt plug is regulated down to an operational voltage of 5V. This 5V is further regulated to 3.3V for Bluetooth chip internal operations. The NF RFID chip utilized the 5V directly, and the MSP430 runs off the internal regulator of the NF RFID chip that outputs 3.3V as well.

4.2.8 Monitor Integration

Andoer MK808B Plus Android 4.4 HDMI TV stick is a wireless device for displaying video to a television screen using an android based operating system. This system is used to provide a graphical interface for the tag data being presented. Once each tags data is transmitted to the master unit, and decoded, the information provided is sent to the monitor device so that an informational screen can present all current tag data values for display. As tag information is updated this information can be added or removed from the display in real time.

The monitor itself is an HDMI based television to allow the android stick to transmit its information to the video receiver. Size and resolution is based on a case to case basis with the airline installing the service. Depending on distance and font size being presented to the consumer by the system, the display monitor must increase in size to be seen at a distance. The monitor has the ability to be mounted to a wall or pole within the baggage area, and rugged enough to run in a busy environment with many people moving about causing potential unwanted vibration or damage. The MK808B Plus Android Stick can be seen in Figure 4.2.7.



Figure 4.2.7 - The MK808B Plus Android Stick

4.3 Software Design

4.3.1 Microcontroller Unit Software

The microcontroller was programmed in C and controls the identification of the tag being scanned through its manufacturer identification number present in each of the tags as they are made. The base program was provided by TI specifically for their RFID chip, and was modified for the MSP4302553 controller.

Programming of the software and flashing of the chip was accomplished using the Code Composer Studio designed to work with the Texas Instrument line of chips.

During startup of the Scanning Unit, initialization of the device chips started. The Bluetooth chip powered up, defaulted into Idle Automatic and awaited connection requests. Connection requests were generated by the Master CPU when requesting data from the scanning unit.

Next, the TRF7970 powered up along with the MSP430. As the MSP430 initializes, it set up the serial communications TX and RX pins for connection with the Bluetooth module, and MISO/MOSI for the TRF7970. The serial communication pins were established after this initialization, and data was sent to the TRF7970 to initialize its starting mode.

With the TRF7970 initialized the IRQ bit was cleared and the tag scanning began. Setting the transmit power out flag within the TRF7970, the chip output a signal to power up any tags within range. A single tag's UID data was read from the tag. After a delay of about 6 milliseconds, the chip powered down transmission, and set the IRQ line. If a tag was detected, its UID information was temporarily stored in the MSP430's memory through the SPI connection, and offloaded through the MSP430's UART to the Bluetooth chip.

The Bluetooth chip in its Idle Automatic mode awaited a connection with the Master CPU. At the time the connection is established through the Bluetooth protocols, the UID data present on the UART input lines was transmitted for processing..

4.3.2 Java Language Coding Standards

The group's developers were required to adhere to a strict and detailed set of coding standards for the java portions of the system. These coding standard rules have been accepted by all members who were tasked with java development for the project. These coding standards were configured within each developer's Integrated Development Environment, and the IDE was set to automatically adjust the code to adhere to the coding standards as much as possible after the developer

has saved their work. Adhering to strict Coding Standards was important because our group was using a Subversion Repository to manage code. If the formatting and style was different for each developer, merging source code changes could become extremely problematic.

4.3.2.1 Tabbing Policy

The group's policy for *tabbing* was fairly standard. The tab characters were not allowed. All tabbing was performed by using four spaces instead of the tab character. Wrapped lines were tabbed over using an additional 8 spaces. Class bodies, class members, enum declarations, method bodies, block bodies, switch bodies, case bodies, and 'break' statements were all indented over. Nested indentation was always used to visually signify a change in code scope. These tabbing standards were in place to ensure a universally readable format for all developers in the group.

4.3.2.2 Brackets and Parentheses Policy

The group also had a fairly standard policy when it came to *brackets and parentheses*. Opening brackets were not located on a different line, but instead located at the end of the block's control flow line. Brackets were also always used for bodies, regardless of the size of the body. Even if the body had only one line of code in it, the body was wrapped in brackets, despite the fact that the brackets are not computationally necessary. This standard was in place to prevent accidental bugs which could have arisen from code not being located within the scope the developer believes it is in. Closing brackets were almost always located on their own line, with the exception being cases where a flow control operator was proceeding. For example, for else-if statements and for do-while statements. Parentheses were only used when necessary. This policy existed to simplify the appearance of mathematical equations and other code segments which require parentheses.

4.3.2.3 White Space Policy

The group's *white space* policies were designed to optimize the readability of the code. White Spaces should have been located before opening parentheses for conditional and flow control statements. This policy did not apply to method calls, where no space between the method name and the method's arguments should exist. White Space should also exist before opening brackets under all circumstances. White Spaces should be located after all commas and colons. White Spaces should also exist before colons, except when the colon is being used to signify a label, in which no white space should exist between the label name and the colon. In the rare situation where code may follow a closing bracket on the same line as the closing bracket, such as for an else-if statement, white space should exist between the closing bracket and the start of the preceding code. White Space should also exist before and after all operators, with the exception of

increment and decrement operators, where no spaces should exist before or after them.

4.3.2.4 Shorthand Policy

The group's *shorthand* policies were designed to allow shorthand coding while also limiting the abuse of shorthand coding, which may have lead to code which was difficult to maintain and debug. The *for-each loop* statements were allowed only in situations where an index was not required, and only in situations where the iterator was not modified in any way within the loop. This policy existed because while retrieving the index of an item within a *for-each loop* was doable, it required additional computation which would otherwise be unnecessary if a standard *for loop* was used instead. Additionally, this policy prevented the modification of the iterator, or any data structures being iterated, within the loop because these modifications threw a *Concurrent Modification Exception* and resulted in the program crashing. The *ternary operator* statements were allowed only in situations where the results of the conditional statements were basic, and only when there were at most two outcomes to the conditional statement. This policy existed because complex or lengthy results in ternary operator statements could have become cumbersome and could have result in code which was difficult to maintain and difficult to debug. Additionally, this policy prevented ternary operators from being chained together to create three or more possible outcomes for the line of code which had the ternary operators. This restriction was put in place because chaining ternary operators could have also created code which was difficult to maintain and difficult to debug.

4.3.2.5 Blank Lines Policy

The group's policy for *blank lines* was very basic. It is important to note that comments are not considered blank lines. No blank lines should exist before the package declaration. A single blank line should exist after the package declaration, before the import group, after the import group, between the class declarations, before the first declaration in the class, before the field declarations, after the field declarations, and between class methods.

4.3.2.6 New Lines and Line Wrapping Policy

The group's policy for *new lines and line wrapping* was also fairly straightforward. A new line should have be added for all empty bodies. A new line should have existed after annotations. A new line should exist after enum values. A new line should exist after opening brackets, with the exception of when an array is being directly initialized. No new lines should exist for else statements, catch statements, finally statements, or while statements. Additionally, each line of code should wrap after a linewidth of 200.

4.3.2.7 Class Fields Policy

The group's policy for *class fields* was very extensive, because the group wanted to make sure the code was easy to maintain and debug. Our first policy for class fields was to restrict complex types to their most reasonably abstract type. An example of this was declaring a variable as a *java.util.List*, instead of a *java.util.ArrayList*. The variable may have been initialized as a *java.util.ArrayList* later in the class, however the class variable must have been restricted to a *java.util.List*. This policy existed because declaring the complex variables like this allowed for more maintainable code in the event where the initialization type needed to be changed for some reason. The "most reasonably abstract type" was determined by which methods the complex type needed access to during development. Our second policy for class fields was to restrict the naming convention of normal variables to be formatted in camel case. Camel Case states that the field's name should lead with a lowercase letter, and if the field's name contains multiple words, the words should be pushed together without spaces and the leading character for each word, excluding the first word, should be capitalized. The naming convention for constant fields was different, however. Constant fields were fully capitalized, with underscores separating words in the case where a constant field's name had multiple words.

Our third policy for class fields was the order in which they were declared in the class. The fields were first ordered by their specialty type, with any constants coming first, followed by static fields, followed by final fields, with normal fields coming in last. The fields were then ordered based on visibility, where private fields were first, protected fields were in the middle, and public fields were last. Lastly, the fields were ordered based on alphabetical order, where 'A' is first and 'Z' is last. Our fourth policy for class fields required that the fields should always be accessed within the class by using the 'this' qualifier. This policy existed to prevent any bugs which may have arisen from accidentally hiding fields and variables. This policy also existed to make it easier to read the code, as the 'this' qualifier quickly told the developer that the field has class scope. Our fifth policy for class fields was that all class fields should be restricted to the private scope if at all possible. Access to the fields from outside of the class should be restricted to the usage of getters and setters. This policy existed to prevent regression bugs from creeping into the system from code changes. If a developer needed to perform some sort of check on a field prior to allowing an external class to retrieve the value of the field, having the access to that field restricted to only getter methods meant the developer will not have to track down every single time the field was directly accessed outside of the class. This policy had enormous positive impacts on code maintainability in the long run. Our final policy for class fields was that all unused private fields should be deleted. This policy existed to prevent the existence of unnecessary fields and unnecessary memory usage. If the field was not used by the class, it did not belong in the class.

4.3.2.8 Class Methods Policy

The group's policy for *class methods* closely resembled our class field's policy. Our first policy for class methods was the order in which the methods should be organized. The methods were first ordered based on their visibility, with public methods coming first, protected methods in the middle, and private methods at the end. The methods were then ordered in alphabetical order, where 'A' is first and 'Z' is last. Lastly, the methods were ordered by ascending number of parameters, where the least number of parameters come first and the most number of parameters come last. The order of constructors in the class was a bit different. If any of the constructors in the class were public, all the constructors should have been located before the class methods. If none of the constructors in the class were public, all of the constructors should have been located after the class methods. From there, the constructors were ordered in the same fashion mentioned earlier for the class methods.

4.3.2.9 Imports Policy

The group's policy for *imports* was fairly basic. All imports should have been explicitly declared. The usage of importing entire packages with the asterisk wildcard was not allowed under any circumstances. All unused imports were removed. These policies existed to prevent the unnecessary importing of classes which may result in excessive memory usage. Additionally, the imports should have been ordered in alphabetical order. This policy existed to improve the maintainability of the code.

4.3.2.10 Comments Policy

The group's policy for *comments* was very strict. Our group views comments were a critical part in maintaining code in a shared environment, so we considered it extremely important that our code was correctly commented in a way which can be understood by all of our developers. First of all, our Integrated Development Environment's automatic comment formatting was enabled. This tool was important because it allowed us to standardize our commenting format. No comments exceeded a linewidth of 200. This policy existed to ensure that comments do not expand beyond the linewidth of the code. Block comments should have been restricted only to comments spanning more than a single line. For comments which were only a single line or less in length, line comments should be used instead. Additionally, line comments were restricted to be used only for comments which were a single line or less in length. Embedded comments should have only been used for complex operations and they should have been used responsibly. Leaving a comment for simple lines of code was unnecessary and cumbersome, so we did not allow it. JavaDoc comments were extremely useful as documentation, both due to their standardized nature and due to their integration with our Integrated Development Environment. JavaDoc comments should have

been provided for all classes, class fields, and class methods, regardless of their visibility and/or purpose. Additionally, JavaDoc comments should include descriptions for all parameters and return types.

4.3.2.11 General Coding Policy

The group also had many *general coding* policies. First, inner classes were only used when no other option was possible. Inner classes were commonly used excessively or when better alternatives were available, and this practice could have led to code which was difficult to maintain, which is why we were restricting them. All code should be set to be automatically formatted when the file is saved. This policy existed to ensure our policies are adhered to as strictly as possible. The annotation '@Override' should be used whenever applicable. This annotation signified when a method was inherited, which was extremely useful when reading the code and attempting to debug it. Any unused local variables should have been removed. These variables took up unnecessary memory and usually included unnecessary additional computations. The 'TODO' tag should have been used to take note of all unfinished code. The 'FIXME' tag should have been used to take note of all unresolved bugs in codes. The 'XXX' tag should have been used for any other important notes. These tags were being used because of their integration with our Integrated Development Environment, such that we could easily locate them and make the necessary adjustments.

4.3.3 Abstraction Software

4.3.3.1 Model Objects

To simplify our software design for our Main CPU and Android Stick devices, our group planned on sharing some object functionality by creating an abstraction library. The abstraction library was required to contain shared functionality for the Main CPU and Android Stick model classes to be built off of. The UML diagram for this library can be found below, in Figure 4.3.1.

The *AbstractBaggage* object was the first object in our abstraction library. This object contained fields that both *Baggage* Objects in the Main CPU and Android Stick share. Specifically, this class contained the bag's airline IATA code, the flight the bag was on, and the passenger's display name. Additionally, this class contained the associated getter and setter methods for each field. The scope of the fields in this class were protected, instead of private, so they were easily accessible by the inheriting baggage classes.

The *AbstractServiceResponse<T>* object was a generic response object for the various service classes in our system. The class contained a boolean where *true* indicates the service operated successfully, and *false* indicates the service was unable to successfully complete its operation. Classes which inherit from this class

were expected to specify a result type for the T parameter. Additionally, this class contained a getter and a setter for an unspecified *result* variable of type T . The scope of the success variable was protected, instead of private, so the variable could be easily accessed by the inheriting service response classes.

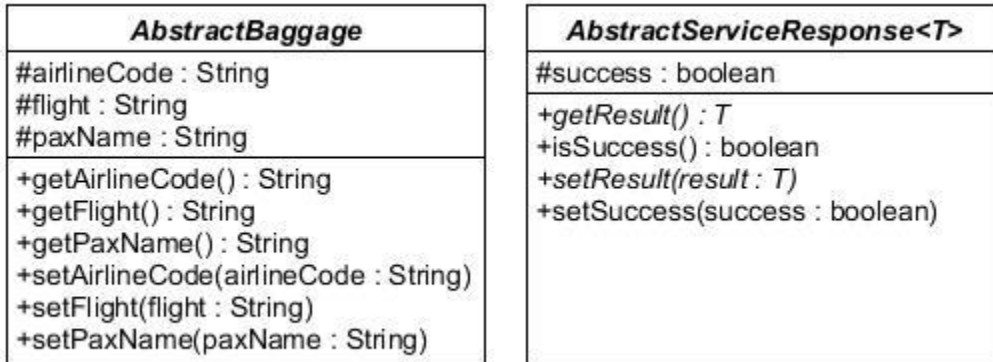


Figure 4.3.1 - UML diagram Model Objects

4.3.4 Main CPU Software

The program running on the CPU was required to be using a subset of Java known as Java ME Embedded. Java ME Embedded is a miniaturized version of the Java programming language designed for small embedded headless devices. The software for the CPU was required to be compiled and packaged using the Java ME Embedded 8.1 Development Kit, so the compatible Java ME Embedded 8.1 Runtime Environment should be installed onto the CPU.

4.3.4.1 Required Libraries

4.3.4.1.1 JUnit 4

JUnit 4 was required to be used for creating and maintaining our automatic java unit tests. JUnit was chosen because our developers had experience using it and because it is the most popular unit testing software available for java. JUnit is so popular that Maven contains specialized functionality for making the JUnit and Maven operate collectively. The usage of JUnit means we were required to be able to easily create and manage our various unit tests in an automated fashion.

4.3.4.1.2 Mockito

Mockito was required to be used alongside JUnit for the sake of mocking functionality in order to isolate unit test cases. Mockito is a widely used library for unit testing, especially when using JUnit. Mockito was chosen because our developers had experience using it and because of its popularity. The usage of Mockito meant we were able to write more isolated and specific unit tests without having to worry about unrelated code impacting the results of the unit tests.

4.3.4.1.3 Bluecove

Bluecove was required to be used for the Bluetooth capabilities on the CPU. The CPU needed to act as the master of a piconet, so we needed to find a java library with sufficient support for our needs. Bluecove was chosen because it is one of the few libraries we could find that implement the JSR-82 Bluetooth Application Programming Interface. JSR-82 implementation was important to us because it is the current Bluetooth minimum standards specifications for the Java Micro Edition subset. The usage of Bluecove meant we only needed to configure an existing library for our Bluetooth needs in our CPU, instead of implementing our own library for Bluetooth functionality.

4.3.4.2 Model Objects

The data model for this system was fairly basic. The UML diagram for the Model Objects in our Main CPU can be found below in Figure 4.3.2.

Baggage information sent to our Main CPU was used to construct *Baggage* objects. A *Baggage* object inherits from the *AbstractBaggage* class and contains the bag's airline IATA code, the bag's identification information, the passenger's display name, the flight the bag was on, the airport the bag was scanned at, the carousel number the bag was scanned at, the time the bag was first scanned, and a value indicating when the bag should timeout.

All of the *Baggage* objects are stored in a synchronized linkedlist hashmap named *BaggageCache*. The *BaggageCache* map acts as an ordered data cache, where the key is the bag's airline IATA code concatenated with the bag's passenger identification information and the value is the *Baggage* object. The objects are ordered by the time they were first scanned, with the most recently scanned bag being the last object. Objects are removed from the *BaggageCache* once their timeout value has been passed.

The *MasterController* is a singleton object which holds, and is tasked with managing, the *BaggageCache*. The *MasterController* also determines what data to send to the airline hosts for passenger notification, updates the *Baggage* objects timeout values, and determines when the monitor devices should be updated. The *MasterController* also removes objects from the cache if their timeout value has been reached. The *MasterController* contains a hashmap of airline endpoint paths, which it is tasked with managing. The hashmap's key is the airline's IATA code, while the value is the airline's notification endpoint path. This hashmap is cleared after an hour of inactivity.

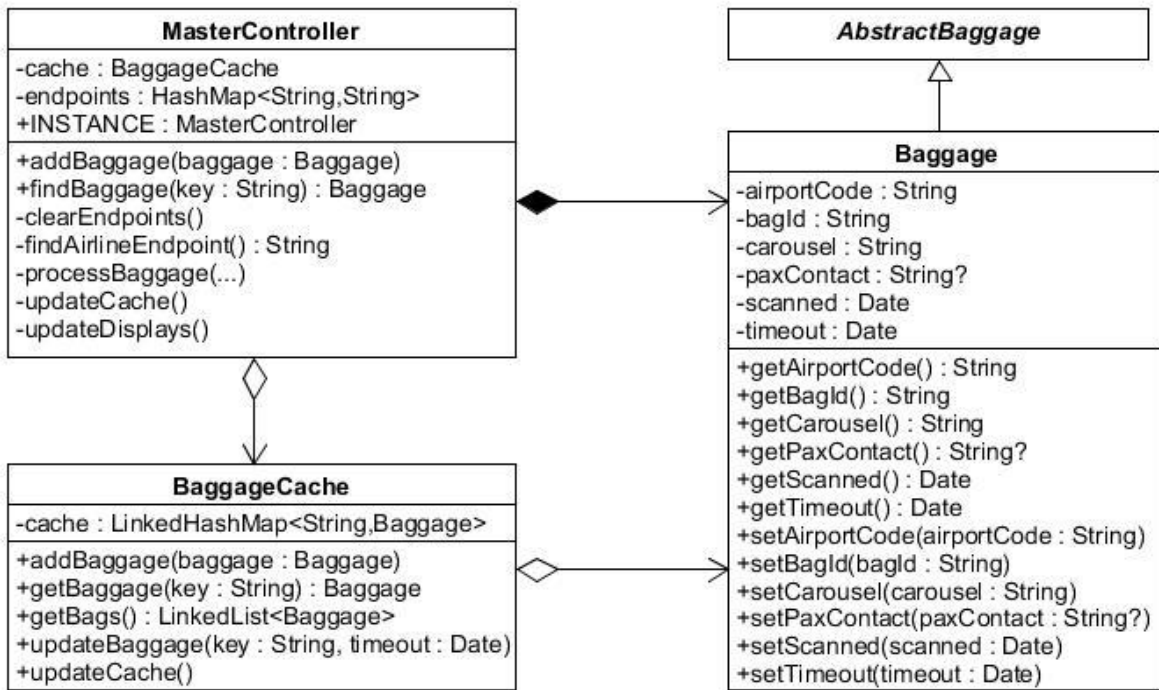


Figure 4.3.2 - UML diagram Model Objects

4.3.4.3 Services

The Main CPU was required to do several input and output tasks, so it contained several different services. The UML diagram for the Main CPU's Services can be found below in Figure 4.3.3.

The baggage tag data first arrives in the Main CPU via a Bluetooth connection. The *BaggageTagService* exists to manage the Bluetooth connections, retrieve the incoming data, and convert the incoming data into *Baggage* objects. By creating an instance of a *BaggageResponse* object, the newly created *Baggage* object is passed to the *MasterController* to be added to the *BaggageCache*. The *BaggageResponse* object inherits from the *AbstractServiceResponse* class and contains a boolean value indicating whether or not the service method was successful, as well as the created *Baggage* object.

The *AirlineEndpointService* exists to query the airline endpoint webservice and retrieve the notification endpoint paths for a given bag's airline. This service returns an instance of an *EndpointResponse* object. The *EndpointResponse* object inherits from the *AbstractServiceResponse* class and contains a boolean value indicating whether or not the service method was successful, as well as the found endpoint path in the form of a String.

The *AirlineNotificationService* exists to pass information to the airline's web service endpoint for the purpose of notifying the passenger that their bag can be found on the carousel. It is up to the airline's discretion for how they want to notify the passenger - either by text message or a mobile app's push notification. This service does not listen for a response.

The *DisplayUpdateService* exists to push new data to all of the listening Android Sticks in the baggage claim area. This service does not listen for a response.

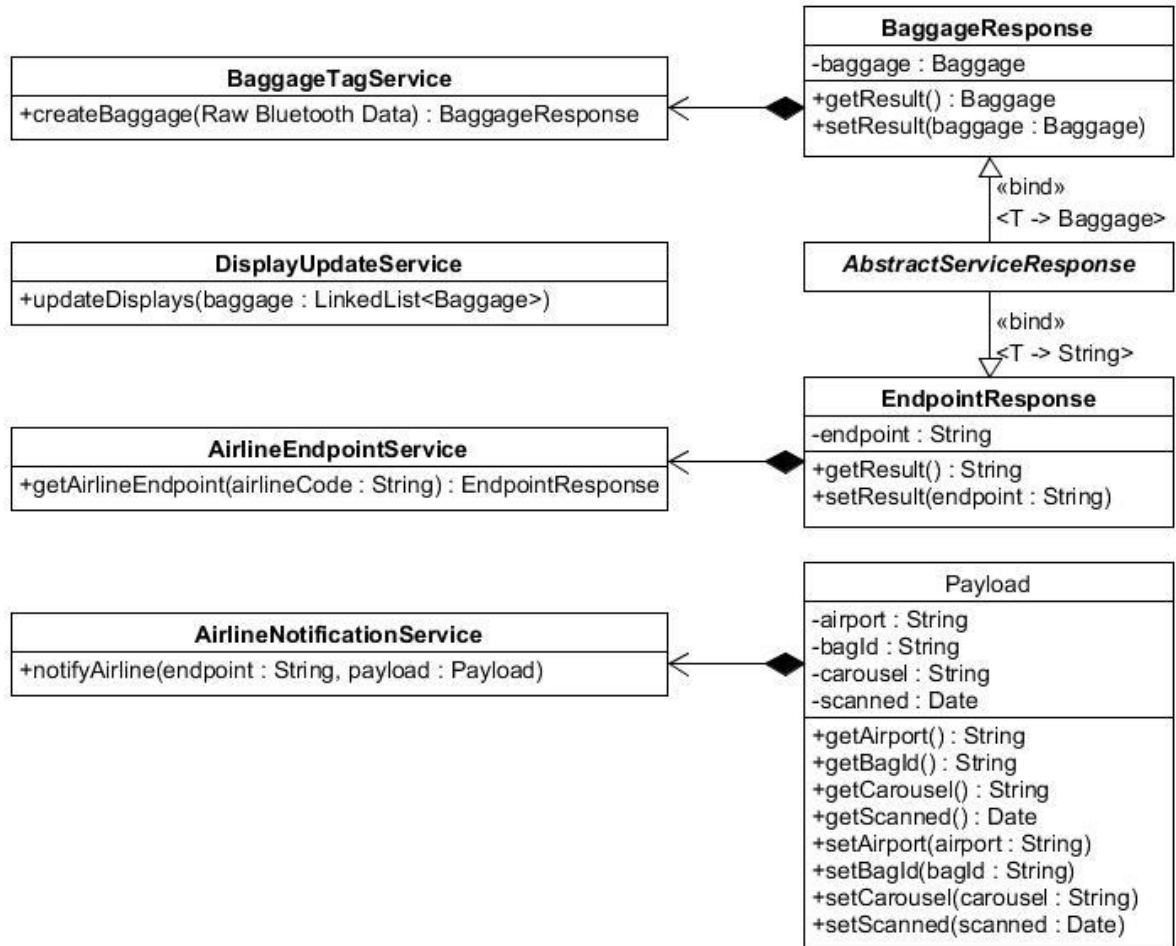


Figure 4.3.3 - UML diagram Main CPU Services

4.3.4.4 Flow Chart

There were two main processes for the Main CPU to perform. The first process was its primary functionality, which was receiving incoming baggage tag data, processing it, and triggering updates to the display, as well as notifying airlines that new baggage has arrived, when necessary. A flow diagram outlining this process can be found below in Figure 4.3.4.

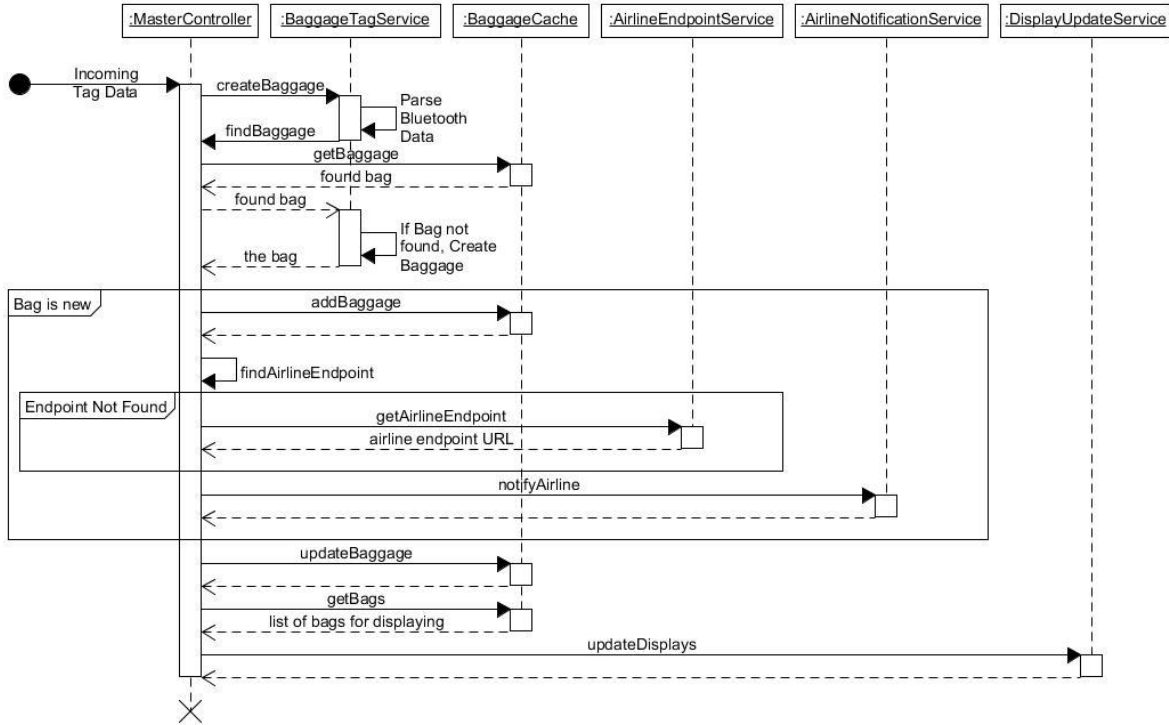
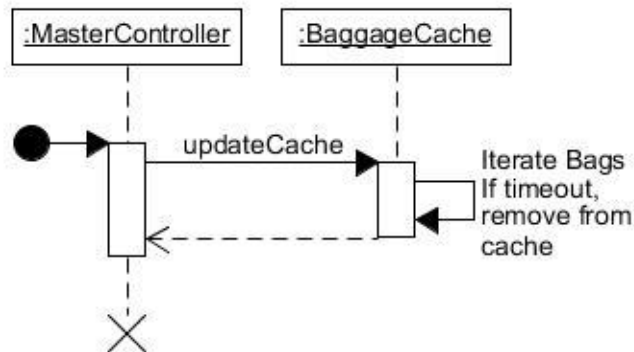


Figure 4.3.4 - Flow Diagram

The second process the Main CPU had to perform is keeping the cache up to date. A secondary process runs in the background and periodically checks the timeout intervals of the bags found in the *BaggageCache*. Since the *Baggage* Objects in the *BaggageCache* are ordered such that the earliest scanned bag is first, the cache only needed to be iterated until it finds an object which has not timed out. A flow diagram outlining this process can be found below in Figure 4.3.5.

Figure 4.3.5 - Flow Diagram



4.3.5 Android Stick Software

The Android Operating System was used on our Android Stick. The specific version of the Android Operating System running on the Android Stick is Version 4.4 KitKat. This version allowed us to use the most up-to-date enhancements for the Android Stick Bluetooth support, allowing us to update the television displays quickly, and with more accuracy. Version 4.4 KitKat also allowed us to use the most up-to-date enhancements to the Android Operating System family's user interface, which also allowed us to create the most user friendly display for the passengers waiting to claim their bags.

The program running on the Android Stick was developed using the Java Standard Edition 7 Development Kit. The program was built as an Android Application with the assistance of the Android Development Kit and Maven. Version 7 was chosen instead of Version 8 because the Android Development Kit's most recent compatibility with the Java Standard Development Kit uses Version 7.

4.3.5.1 Model Objects and Services

The data model for the Android Stick contains only two classes, and there is only one service class. The UML diagram for the Android Stick classes can be found below in Figure 4.3.6.

The *Baggage* object inherits from the *AbstractBaggage* class and contains the bag's airline IATA code, the passenger's display name, and the flight the bag was on. The *BaggageCache* is a singleton array wrapper containing all of the *Baggage* objects to be displayed on the monitors.

The *UpdateDataService* exists to retrieve incoming data from the CPU. This service deletes the existing data found in the *BaggageCache* and replaces it entirely with the incoming data. This service is strictly for receiving incoming data, so it does not send any data.

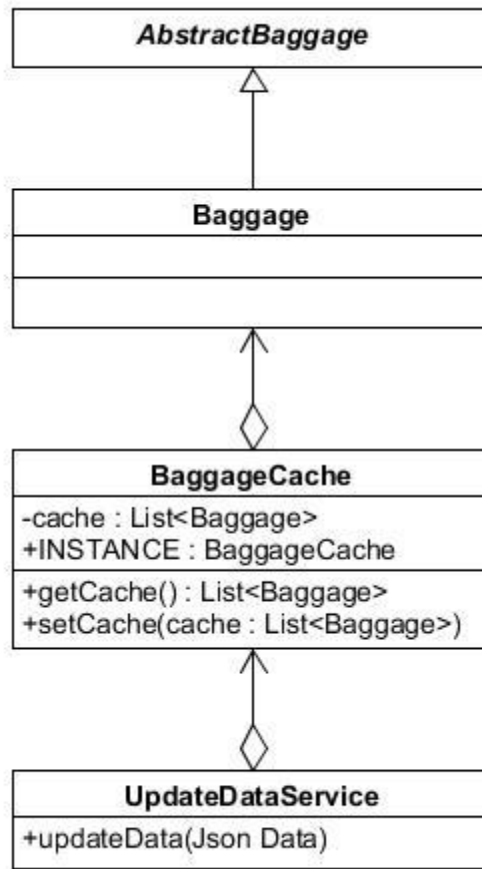


Figure 4.3.6 - UML diagram Android Stick

4.3.5.3 Views

The Android application on the Android Stick contains a single view. This view shows a basic table of the data found in the *BaggageCache*.

4.3.5.4 Flow Chart

The Android Stick only has one primary process. When data is received by the Android Stick from Bluetooth, the internal data cache is replaced with the incoming data and the display view is updated. The flow diagram for this process can be found below in Figure 4.3.7.

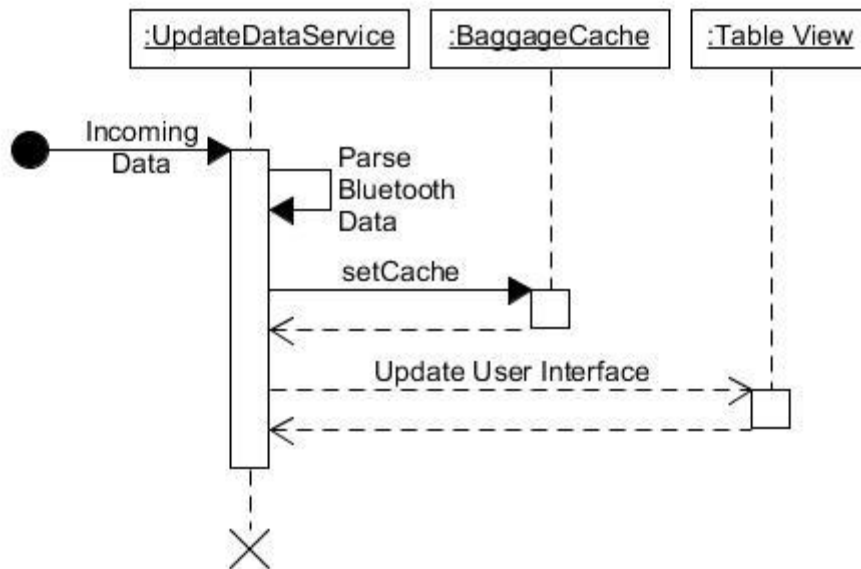


Figure 4.3.7 - Flow diagram

4.3.5.5 Modes

This Json file, as seen in Figure 4.3.8, contains a list of bags from the latest update with the display information specified by the airlines. Each line depicts one bag object with the corresponding IATA airline code, the flight number, and the passenger name. After the file is parsed, the android stick updates its cache and displays the new information accordingly.

```

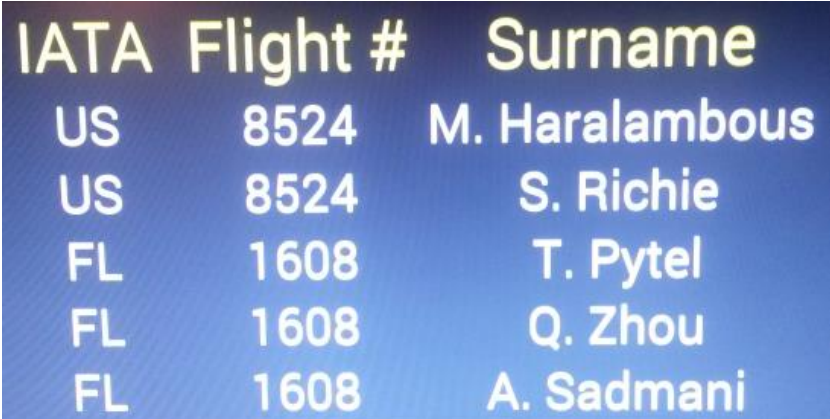
{"Bags:"
{
  {"airline":"FL","flight":"744","paxName":"J. Smith"},
  ...
}
}
  
```

Figure 4.3.8 - Json data contents

The monitor requirement consists of any HDMI based television to allow the android stick to transmit its information to the video receiver. Taking size into consideration, the resolution size is only required to be high enough to accommodate the font size needed for displaying tag information at a distance.

The Mk808b Plus Android Stick receives the data information from the Raspberry Pi via the built-in Bluetooth receiver. Depending on the command contents being sent from the Raspberry Pi, the Android Stick will either proceed to cache the new information it and send it to the monitor to display, or enter standby mode. Standby mode displays advertisements on the screen, and returns to displaying luggage data after the next prompt from the Raspberry Pi. This mode is also turned on

when the system is idling when not in use. Figure 4.3.9 depicts the data being displayed on the monitor at the airport carousel.



IATA	Flight #	Surname
US	8524	M. Haralambous
US	8524	S. Richie
FL	1608	T. Pytel
FL	1608	Q. Zhou
FL	1608	A. Sadmani

Figure 4.3.9 - Data contents in Display Mode.

The data displayed on the monitor is dependent on what the airlines choose to display, and would be defined for passengers before they opt into the service. The group went with displaying the IATA airline code, the flight number, and the passenger surname accompanied with the first initial of their first name.

During the standby mode of the display feature, any existing advertisements are displayed on the screen until the system is updated with passenger information again. Simply adding this additional feature ensures that the monitors are still being used for advertisement time as well as displaying the C.L.A.I.M. information for the extra service for the passengers. This form of notification coupled with the mobile app push notification allows passengers freedom to partake in other activities while they wait for their bags to appear.

4.4 Miscellaneous Design

4.4.1 Tag Housing Design

For economic reasons and ease of use in mass production, the printed sticker RFID tag was chosen for the tag housing design. The group ordered some RFID sticker tags from a known manufacturer with already written data information required for the project. This design choice removes any extra components for the tag housing, as the RFID chips to be printed can easily be mounted in a similar fashion to that of existing barcode systems used in modern airports. This also allows for removal of any unnecessary parts that could interfere with the RFID scanner from picking up the RFID chip signals.

The RFID tag can be placed on a paper reel with printing paper for traditional barcode printed information. The sticker adhesive would then be wrapped around the luggage handle. For testing purposes, the transponder used is the one shown in Figure 4.2.1, where it is mounted onto the surface of a test object while attached

using a form of tape or sticker in order to simulate the interference of a thin paper film over the tags antenna.

4.4.2 Mounting Design

4.4.2.1 Flat Conveyor System

Different airports have different conveyor carousel systems for baggage claim, which require different mounting positions in order to accomplish the functionality of the project. There are a variety of things to consider for the placement of the system; the most important being that it should be out of the way as much as possible while still being in the ideal position to meet the objectives.

For the flat single-level conveyor systems, the scanner should be mounted to the wall with a “L” bracket. In this conveyor system, there is a single belt which begins in the backroom where the airline’s ground crew load the bags. The belt travels outside to the baggage claim area, and then travels back into the backroom to form a large, continuous loop. The external antenna should be mounted along the bracket in such a fashion that it will hang over the conveyor track in order to scan all objects that pass below it. The Master CPU can be located either in the backroom or in a discreet location in the baggage claim area where the public would be unable to access it. The Master CPU would need to be close enough to the Microcontroller units and the Android Sticks attached to the TV Displays, in order to be able to communicate effectively with them. Figure 4.4.1 below depicts how the system would be mounted.

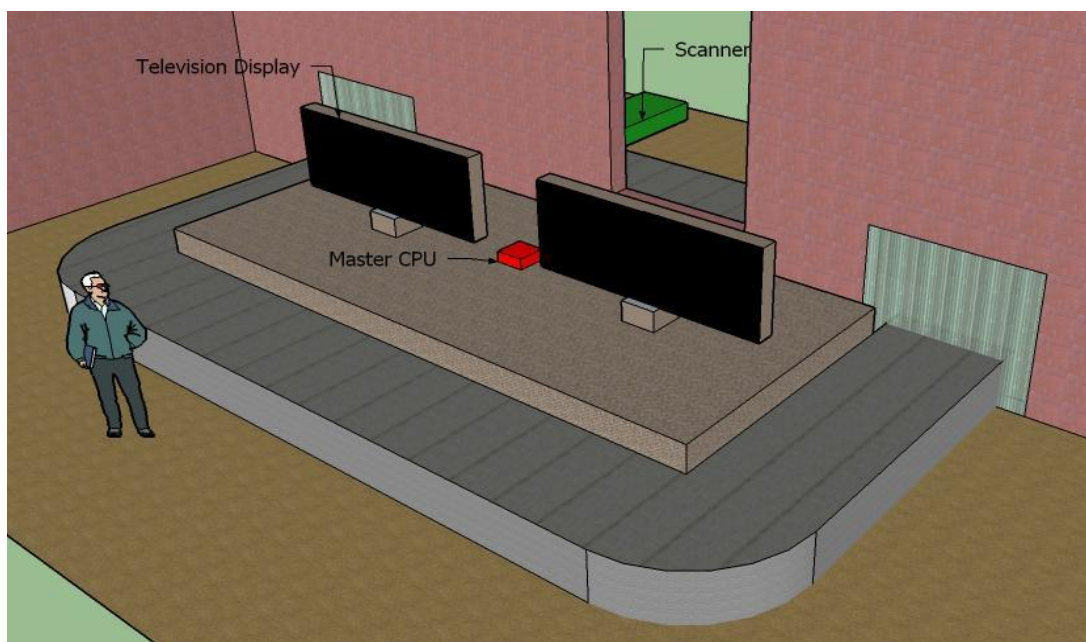


Figure 4.1.1 - Example of mounting configuration for a flat conveyor system

4.4.2.2 Sloped Conveyor System

For the sloped multi-level conveyor systems, the scanner should be integrated into the already existing bumper system. This will allow each passing luggage bag item to be scanned as it enters the track system. The Master CPU can be located either in the backroom or in a discreet location in the baggage claim area where the public would be unable to access it. The Master CPU would need to be close enough to the Microcontroller units and the Android Sticks attached to the TV Displays, in order to be able to communicate effectively with them. Figure 4.4.2 below depicts how the system would be mounted.

One important differentiating factor between the sloped conveyor system and the flat conveyor system scenarios is that the sloped conveyor system requires an additional microcontroller unit to scan luggage on the outside carousel. This is due to the luggage carousel not looping around the backroom, resulting in luggage items needing to be scanned on the separate track to update the TV display information correctly.

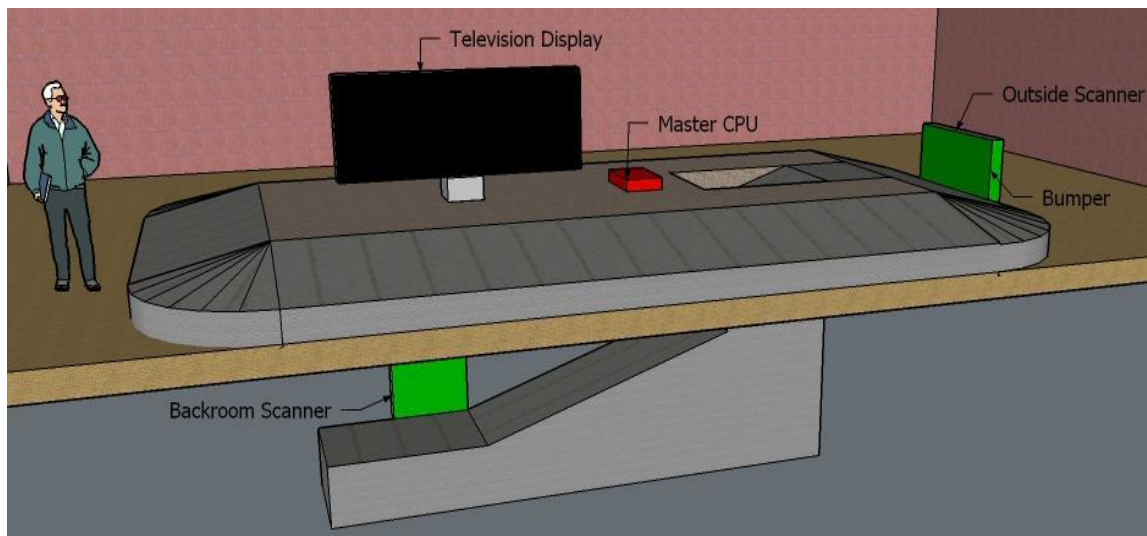


Figure 4.4.2 - Example of mounting configuration for a sloped conveyor system

4.4.2.3 Single/Multi Level Conveyor System Complexity

The group decided to focus on the flat single-level and sloped multi-level conveyor systems as they are the most commonly used type of luggage carousels. There are three known and confirmed configurations that could be supported by the project. These include: the sloped multi-level system which has a single carousel and a loading belt, the flat single-level system which has a single carousel, and the flat single-level system which has a single carousel and a loading belt. It is possible other configurations exist as well, but they will not be put into consideration for the group's project. Figure 4.4.3 below depicts how the system would be mounted.

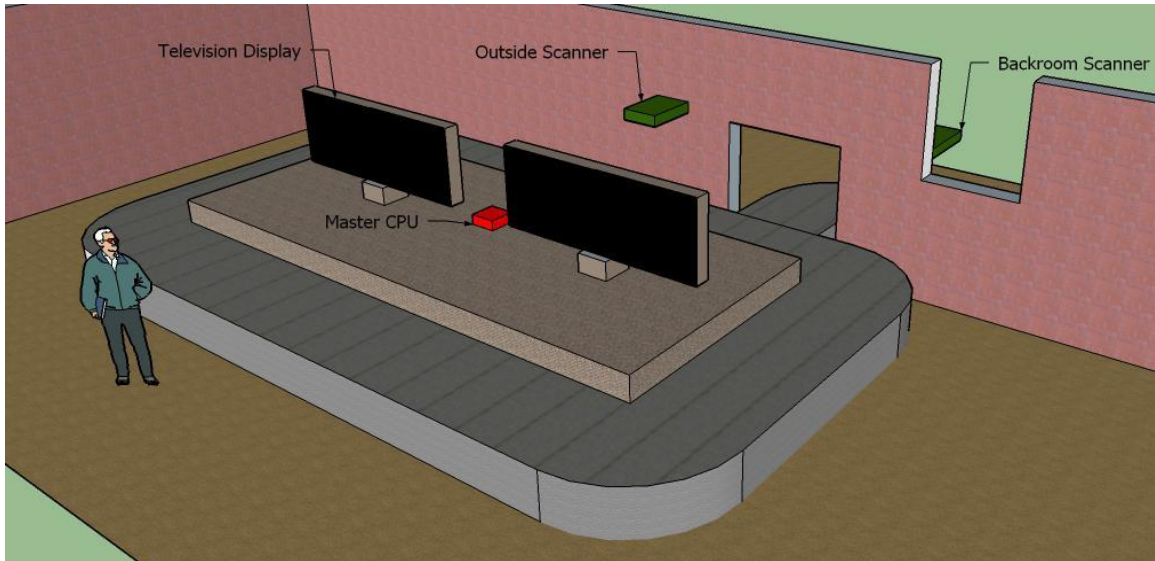


Figure 4.4.3 - Example of mounting setup for a possible flat conveyor system

4.4.3 Device Housing Design

The group decided to go with the only available option of buying a pre-built case and modifying it as necessary for the project. Several holes will be drilled out using tools on hand to allow all the necessary wires and ports to be attached correctly. This decision was made to avoid any potential hazardous violation of standards regarding the casing.

5 Design Summary

5.1 Hardware Design Summary

For the RFID scanning unit, a device that can scan not only at a distance but also through potentially thick objects was needed for use in a luggage situation. The tags orientation and position could be in a multitude of locations and distances away from the scanning antenna as well. Ideally, a Ultra High Frequency receiving unit would work to cover distances above a conveyer belt in the ranges over 1 meter. This would allow the scanner to be placed at a distance over the track that the luggage would be placed on with no worries of oversized baggage not being able to pass under the scanning area, while still providing enough power to penetrate various materials and local interference.

UHF devices have a higher starting cost, upwards of \$500 to \$1000 in the receiver and transmitter circuit boards, and associated antenna. We decided for this project to scale down the demonstration of the system by using a high frequency near field communication device. The HF devices are meant for close or vicinity scanning, these range from a few centimeters to a meter from the scanning surface. The HF type of devices are for card reading, where you scan a card over a scanning surface, which would be located directly on the circuit board, or a security device antenna that can scan objects as they pass through a passageway or doorway. This minimum required distance allows us enough room to demonstrate the product and procedures of the tag and reading device while maintaining a lower overhead cost of smaller scaled parts.

In our application, we are not using an internal board integrated antenna but rather an external antenna mounted through a jack. This will allow proper separation of the RFID transmission and the wireless Bluetooth transmission from interfering with each other as that RFID device generates a much larger power output compared to the Bluetooth. Each system are on different carrier frequency, as the Bluetooth is 2.4 GHz base frequency, and the RFID system runs on 13.56 MHz , and upwards of 960 MHz for the UHF systems.

Since the HF RFID device being used is a lower power device between 100mW and 200mW, the scan range is limited to a few inches from the radiating antenna, and as such does not have enough power to scan tags if they are obstructed with objects within the luggage. A more powerful system, along with at least 2 antennas are needed to compensate for issues in regards to penetration and orientation of tags.

The connection of the scanning device to the microcontroller is through a simple serial communication line, this type of connection makes it easily upgradable to a higher power device that also uses any form of serial communication.

Currently, the chip being used requires as little as a few kilobits of programming to scan a basic passive tag and is within the size and ability of the MSP430 line of microcontroller to handle the data flow of the scanner to another serial device. All the tag data picked up by the scanner is passed through the msp430 and directed to a serial Bluetooth transmitter, this data is transmitted to the main processing board.

The main processing board is comprised of a Raspberry Pi, with peripherals of a Bluetooth attachment and a Wi-Fi attachment. The option to go with the Bluetooth as the serial communication was its simple ability to pair and send commands to a single device. This allows the ability for the master unit to act as the master Bluetooth of a piconet, and is capable with expanded programming, to control multiple scanners within a system, should the need arise for multiple scanning devices needing to be installed to a single conveyer track. The Pi device allows for expanded operating systems to be installed, its price is cheap in comparison to a full on computer since it's a single board operational system. Along with the availability and functions, it supports Oracle based operations that most of the subsystem operating on. Overall the Pi offer a multitude of online community support for the creation of projects.

Attached to the Pi is also a Wi-Fi device used for the purpose of communication with an external source for text message notification, this can either be direct connection to a local hotspot or local server of the airport in order to pass message request for text message notification.

Along with the text notification, a visual notification on a display is presented on a video monitor, this is accomplished by using an Android based video processor that connected directly to the HDMI port of the monitor.

Since the system will be operating in an airport, the possibility of the system being in operation 24/7 means that power requirements for the system will need a constant draw to power the RFID system for tag activation since the tags being used are passive devices. With constant wireless communication, between all devices on the network, the system will have little downtime. As such, a constant power supply from a wall outlet would be ideal for the operation of the system over other options of battery based systems.

5.1.1 Parts List

A multitude of parts were used during the prototyping stage of the project. The individual components are listed in Tables 5.1.1 through 5.1.7.

Table of Capacitors

Parts	Qty	Value	Device	Package
C1, C2, C3, C5, C9	5	100n	C-EUC0402	C0402
C4, C10, C25	3	2.2u	C-EUC0402	C0402
C6, C7, C24	3	27p	C-EUC0402	C0402
C8	1	4.7u	C-EUC0402	C0402
C12, C13	2	16p	C-EUC0402	C0402
C14	1	10u	C-EUC0603	C0603
C15, C16	2	1500p	C-EUC0402	C0402
C17, C18	2	1200p	C-EUC0402	C0402
C19	1	220p	C-EUC0402	C0402
C20, C22	2	680p	C-EUC0402	C0402
C21	1	10p	C-EUC0402	C0402
C23	1	100p	C-EUC0402	C0402

Table 5.1.1 - Table of Capacitors used

Table of Inductors

Parts	Qty	Value	Device	Package
L1	1	330n	L-EUL2825P	L2825P
L2	1	150n	L-EUL2825P	L2825P

Table 5.1.2 - Table of Inductors used

Table of Resistors

Parts	Qty	Value	Device	Package
R1, R2,R5	3	10k	R-EU_R0402_GE	R0402_GE
R3	1	100	R-EU_R0402_GE	R0402_GE
R4, R6, R7, R8, R9, R10	6	1k	R-EU_R0402_GE	R0402_GE
R11, R12, R14	3	330	R-EU_R0402_GE	R0402_GE
R13	1	47k	R-EU_R0402_GE	R0402_GE

Table 5.1.3 - Table of Resistors used

Table of LED's

Parts	Qty	Value	Device	Package
D1	1	blue	LEDCHIP-LED0603	CHIP-LED0603
D2	1	red	LEDCHIP-LED0603	CHIP-LED0603
D3	1	green	LEDCHIP-LED0603	CHIP-LED0603

Table 5.1.4 - Table of LEDs used

Table of IC's

Parts	Qty	Value	Device	Package
U1	1		TRF796XS-PQFP-N32(RHB)	TRF7960_61
U2	1		LMX9838	LTCC70
U3	1		20 G2X[1/5]2---N20	N20
U4	1		LM3480-3.3	SOT-23-3

Table 5.1.5 - Table of Integrated Circuits used

Table of Crystals

Parts	Qty	Value	Device	Package
Y1	1	13.56 MHz	CRYSTAL	CRYSTAL-SMD-5X3.2
Y2	1	32.768 KHZ	CRYSTAL	CRYSTAL-SMD-5X3.2

Table 5.1.6 - Table of Crystals used

Table of Miscellaneous Parts

Parts	Qty	Value	Device
JP1	1	5.5x2.1mm Barrel	POWER_JACKSMD
S1	1		SWITCH-MOMENTARY-2SMD
SJ1, SJ2, SJ3, SJ4	4		Jumpers
UFL1	1		U.FL
ufl cable cable	1		

Table 5.1.7 - Table of Miscellaneous parts used

5.1.2 Schematic

Figure 5.1.1 shows the TRF7970, used for the broadcast and reception of tag data.

Capacitors C4, C5, C34, C35 are power filter capacitors design to allow fast switching from power off to power on when the device wishes to go from a standby mode to on status quickly.

Y1 is the 13.56 Mhz crystal used to generate the carrier wave for the HF tag transmission, and is split for use as a system clock and data clock for proper timing of the serial data output. C6 and C7 are used to set the oscillation of the crystal based on its internal impedance.

C15 to C20, and C23, C24, along with L1 and L2 form the modulation and demodulation of the carrier wave to activate and receive the data from a tag.

Input/Output pins are used for communication with the MSP430, in this setup, it is in SPI serial communication mode. This is set up based on I/O 0, 1, and 2. By setting I/O 0 to ground, a logic zero is set, and I/O 1, and 2 are tied to VDDX using R1 and R2 to reduce current draw, to set them as logic high values. This setup causes the TRF7970 to boot into serial communication, and not parallel

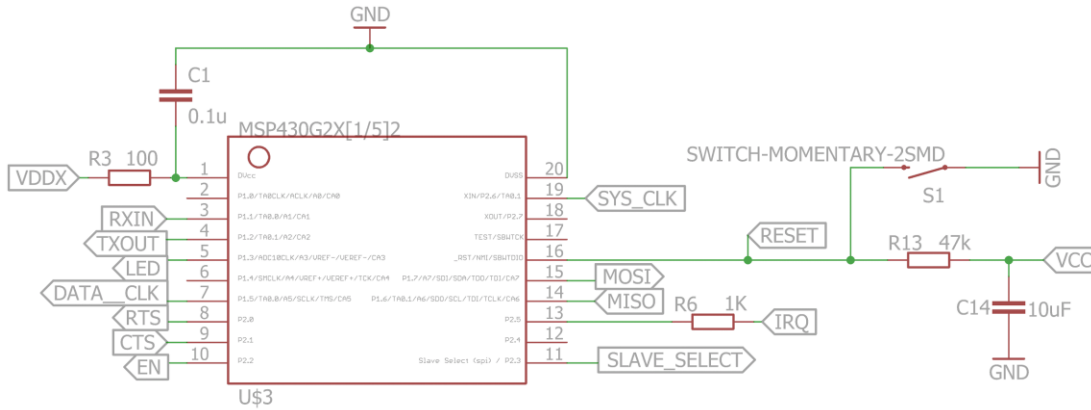


Figure 5.1.2 MSP430G2553

Figure 5.1.3 shows the connection of the jumpers to the LMX9838 for baud rate settings, the 32.768 KHz crystal for communication timing of the Bluetooth. The 5V connection is used to power the main VCC that chips will draw voltage and current from for operation. The LED is used as the heartbeat signal to show that the MSP430 programming is not stuck in a loop and requires a physical reset.

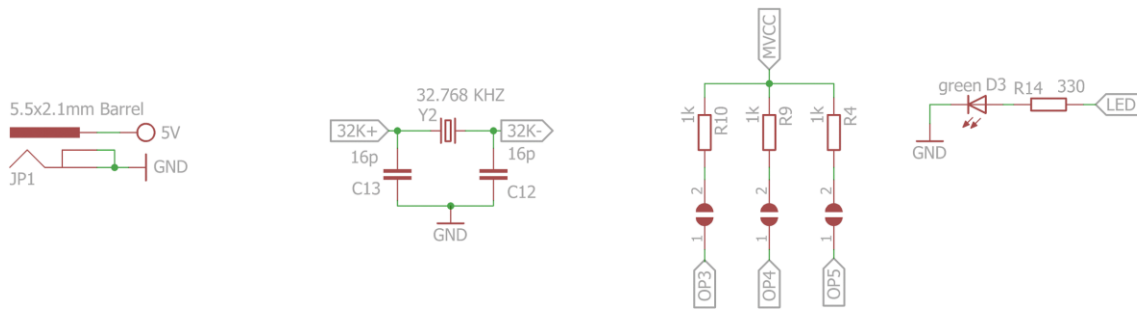


Figure 5.1.3 Other

Figure 5.1.4 shows the LMX9838, it is powered separately from the TRF7970 to avoid current draw requests from both devices. So an LM3480 surface mount voltage regulator is used to drop the 5V input voltage to 3.3V. Majority of the pins on the LMX7838 are unused or connected to a ground plane.

C11, and C10 are used as power filter caps for startup from standby to on status to assist in switching from zero volts to 3.3V in a quick smooth fashion.

Pins 12 to 15 are used for the UART connection to the MSP430 as mentioned in the MSP430 schematic. Pins 16, 25 and 26 are operational pins to set the baud rate of the UART communication.

Pins 7 and 19 are used for status indication that data is being sent through the device.

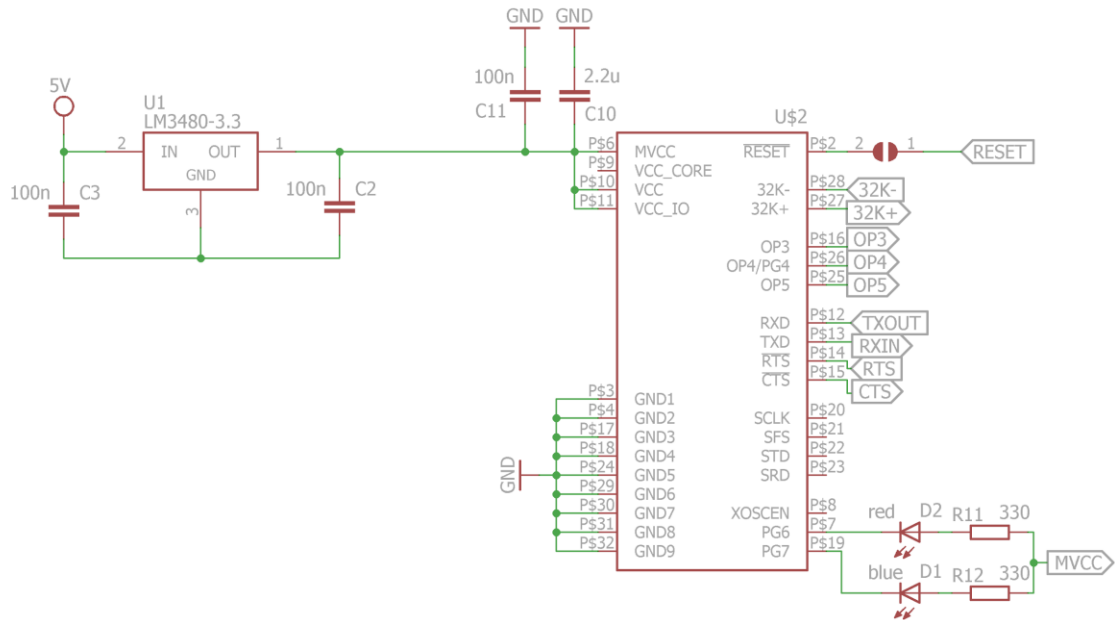


Figure 5.1.4 LMX9838

5.1.2.1 Pin Layout Description

Tables 5.1.8 and 5.1.9 show the pin description for the Bluetooth and Near Field Communication chip

Table of LMX9838 Pin Layout

Pin Name	Description
32K-	Crystal Oscillator negative terminal
32K+	Crystal Oscillator positive terminal
OP3	Baud rate selection pin
OP4	Baud rate selection pin
OP5	Baud rate selection pin
TXOUT	UART Transmit output

Table 5.1.8 LMX9838

Pin Name	Description
RXIN	UART receive input
RTS	Request to Send
CTS	Clear to Send
PG6	Link Status
PG7	RF Traffic
MVCC	Internal Voltage Regulator Input
RESET	Reset on low

Table 5.1.8 *continued* LMX9838

Table of TRF7970 Pin Layout

Pin Name	Description
VDDX	Internal Regulator output
OSC_IN	Crystal Oscillator input
OCS_OUT	Crystal Oscillator output
VSS_D	Ground
EN	Chip Enable
EN2	VDD_X Enable
SYS_CLK	Reference clock off Crystal
DATA_CLK	MCU Communication Clock
MOSI	Master Out Slave In SPI data
MISO	Master In Slave Out SPI data
SAVE_SELECT	SPI direction select
ASK/OOK	Modulations Select
MOD	External Data Modulation

Table 5.1.9 TRF7970

Pin Name	Description
RX_IN1	Main Receiver Input
RX_IN2	Aux Receiver Input
TX_OUT	Transmit Output
VIN	Main Voltage Input

Table 5.1.9 *continued* TRF7970

5.1.3 PCB Layout

The layout (Figure 5.1.5) is set up with the microcontroller in the center controlling the Bluetooth chip on the left, and the RFID chip on the right. Main power from the 5v is routed directly to the TRF7970 where its internal regulator generates 3.3 V on its regulated output pin. This voltage is then used by the MPS430 for operational use. Also off the 5v input, though a 5v to 3.3v regulator is the Bluetooth chip. This is running off a separate regulator to avoid excessive current draw directly from the TRF7970 that could cause interference or slower performance of the Bluetooth communication.

Both the RFID and Bluetooth chip have external crystals of frequency control since each device is a radio frequency communication chip. These crystals are used to set their carrier wave frequencies.

Also attached to the msp430 and Bluetooth chip is a status indicating LED, for visual confirmation of data flow and the msp430's programing is not stuck in an internal loop with a heartbeat indication. A reset button is provided to the Bluetooth and MSP430 for hard resetting of the devices should any data or code become locked up during initial testing of the device. The RFID chip is run directly off the msp430 for control, so any resets are be done through software then hardware. Since the TRF7970 is the primary source of the MSP430, and loss in direct power of the 5V input, results in all chips being reset when powered down.

Due to the serial communication rates of the msp430 and clock rates provided by the provided crystals, jumpers are placed outside the Bluetooth device in order manual set the baud rate of communication between the Bluetooth serial and the msp430. This provides the ability to set the speed faster or slower based on what the msp430 can handle. Final design would include hard wired baud settings, and a more secure surface mounted msp430 device over the dip style currently showed.

All other components show to the right of the TRF7970 are capacitors and

inductors used for the generation and filtering of the carrier wave of the HF RFID system.

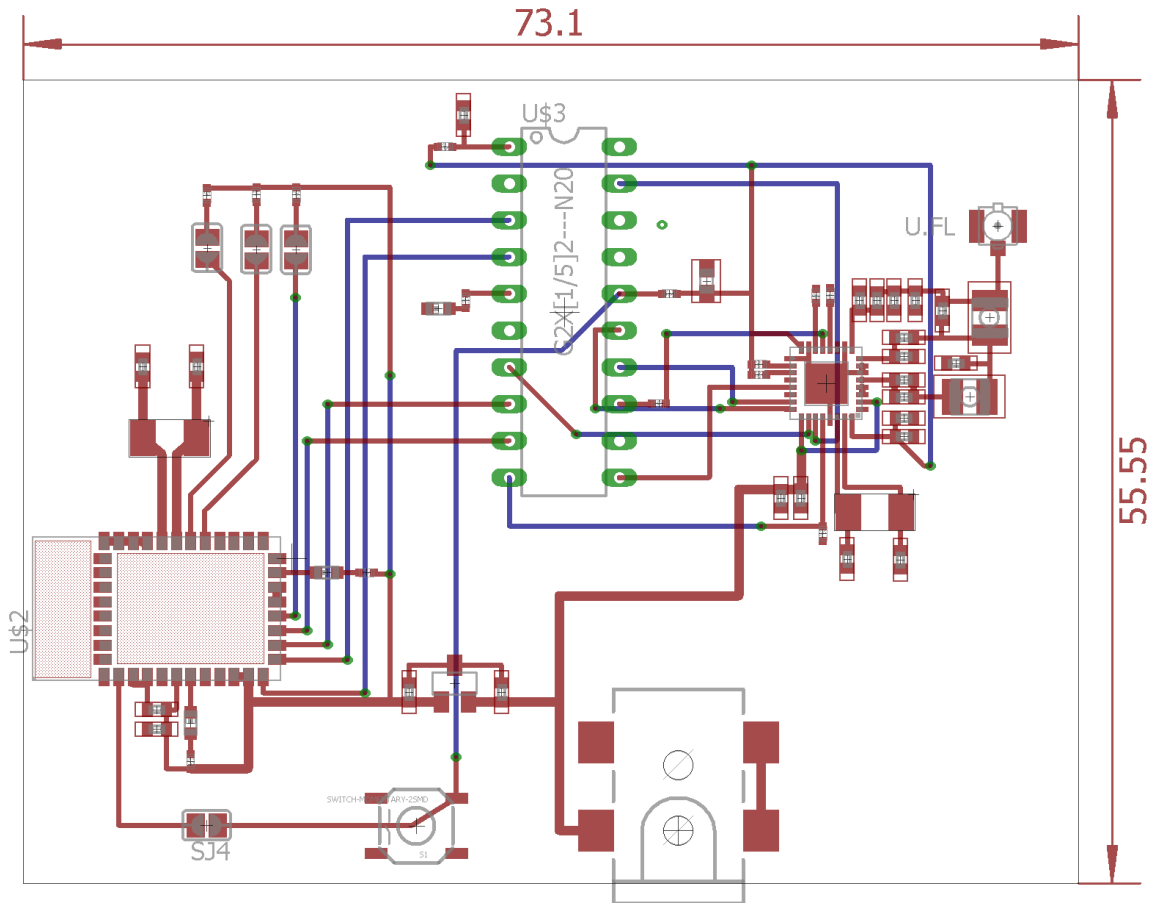


Figure 5.1.5 - PCB Layout

5.2 Software Design Summary

The design for the group's system consists of four parts, each of which need their own unique software and serve their own unique purpose. The first part of the system relies on the Microcontroller software. The second part of the system relies on the Master CPU's software. The third part of the system relies on the airline's implemented webservices. The final part of the system relies on the Android Stick's software.

The Microcontroller software is written in C. The group chose this language because it was the highest level language the MSP430 was capable of running. The Microcontroller software is responsible for operating the RFID receiver and passing the data found on the RFID chips to the MSP430 board's Bluetooth device. The Bluetooth then sends the data to the Master CPU.

The Master CPU runs an Embedded Linux Operating System called Raspbian Wheezy, and the group's code is written in Java ME Embedded 8.1. The group chose Java because it is a very high level language which the developers in the group are comfortable using. Java also offers many libraries and tools which the group can utilize to improve performance of the system and speed up development of the system. The Java ME Embedded 8.1 Runtime Environment is tailor made to run on embedded devices and has explicit support for the Raspberry Pi board. The Master CPU is responsible for managing the Microcontrollers and the Android Sticks. It also serves as the central processing hub for the entire system. The Master CPU controls when the Microcontrollers scan for RFID chips, it interprets the RFID chips' data, caches the data, manages the communications with the host airline's webservices, and manages when the Android Sticks should be updated and what their new data should be. After the Master CPU interprets the RFID data it receives from the Microcontroller, it sends a notification request to the host airline. After it sends the notification request, it updates its internal cache and sends updated data to the Android Sticks.

The airline's implemented webservice runs whatever server-side language they wish to implement with. The group has provided instructions for implementation requirements in Section 2.4.3.3 of this document. The purpose of this web service is to send notifications to the passengers that their bag is available on the carousel to be picked up. It is the host airline's choice as to whether they would like to implement their notifications as a text message, as a mobile application push notification, or in some other fashion. The host airline can also use this service for updating their internal baggage systems and databases. This webservice is not expected to return a result to the Master CPU. For testing and demoing, the group has decided to implement this webservice in PHP. The group chose PHP for the implementation because it is a simple enough language to implement the expected behavior of a host airline's real-world webservice.

The Android Stick runs the Android 4.4 Kitkat Operating System. The group's software for the Android Stick is written in Java and compiled into an Android Mobile Application. The software is designed to only show a single view, which is a table showing the current bags available on the carousel. It is possible in future development that the application could also support the ability to default to an advertising screen, or any screen the host airline would like, when the system is not in use, however that functionality is not part of the group's current design for this project. The software is designed to be aware of incoming data from the Master CPU over a Bluetooth connection. The incoming data is interpreted as the updated data to be displayed on the table and shown to everyone waiting in the baggage claim area.

Due to some overlapping functionality and structure between the Master CPU and Android Stick, the group has designed a shared library which will abstract the shared functionality and structure. This library will be dependencies for both the

Master CPU and the Android Stick, and both subsystems will be built off of the abstracted library. The group decided to do this to reduce the amount of code which would need to be rewritten for both systems. Having the code shared between the two systems ensures an additional level of system synchronicity and cohesion, and makes maintaining the repeated functionality and structure significantly easier.

5.3 Miscellaneous Design Summary

5.3.1 Tag Housing

The group decided on the paper reel design which consists of the RFID tags being placed on the printing paper. These RFID tags will be ordered from a manufacturer and printed with the data information required for the project's specifications. These tags will be attached to luggage bag handles in the same manner as existing barcode printed information systems. Additional information could be printed on the paper reel by the airlines discretion.

5.3.2 Carousel Conveyor Mounting

Even though multiple types of single and multi-level conveyor systems exist, the group will build the mounting for only the two most common scenarios. For the flat single-level conveyor system, the scanning component will be mounted like a shelf above the carousel track. The Master CPU will be located in a discrete location within close proximity of the TV displays and out of reach of the public. For the sloped multi-level conveyor system there will be two scanning components that will be mounted. One will be attached to the side of the conveyor track in the back room, while the other is attached to the bumper railing of the carousel track out in the front. The Master CPU in this setup will also be located in a discrete location within close proximity of the TV displays and out of reach of the public.

5.3.3 Board Housing

The group decided to go with the only available option of buying a pre-built case and modifying it as necessary for the project. Several holes will be drilled out using tools on hand to allow all the necessary wires and ports to be attached correctly. This decision was made to avoid any potential hazardous violation of standards regarding the casing.

6 Prototype Construction

6.1 Parts Acquisition

There were several components that the group was required to buy or utilize to meet the project needs. The acquisition of each necessary component is outlined in Table 6.1.1.

Item	Acquisition
Raspberry PI 2 Model B	Bought
4GB microSD card w/ Adapter	Bought
Generic PL2303HX USB To TTL To UART RS232 COM Cable Module Converter	Bought
TRENDnet Micro-Bluetooth USB 3.0 Adapter	Bought
Edimax EW-7811Un Wi-Fi USB Adapter	Bought
MK808B Plus - Android 4.4 Stick	Bought
Asus VE248 monitor	Owned
PCB	Bought
Mounting	Built
RFID Receiver	Bought
RFID Tags	Bought

Table 6.1.1 - Acquisition status of different project components

6.2 Prototype Board

In figure 6.2.1 the PCB layout as discussed in section 5.1.3 has been manufactured and constructed. Each part was surface mounted to the device, with the MSP430 DIP holder being manually soldered on since the group decided to go with a removable, and editable chip, instead of adding the needed components to program the chip externally. Any future designs would be programmed once, and surface mounted to the board.

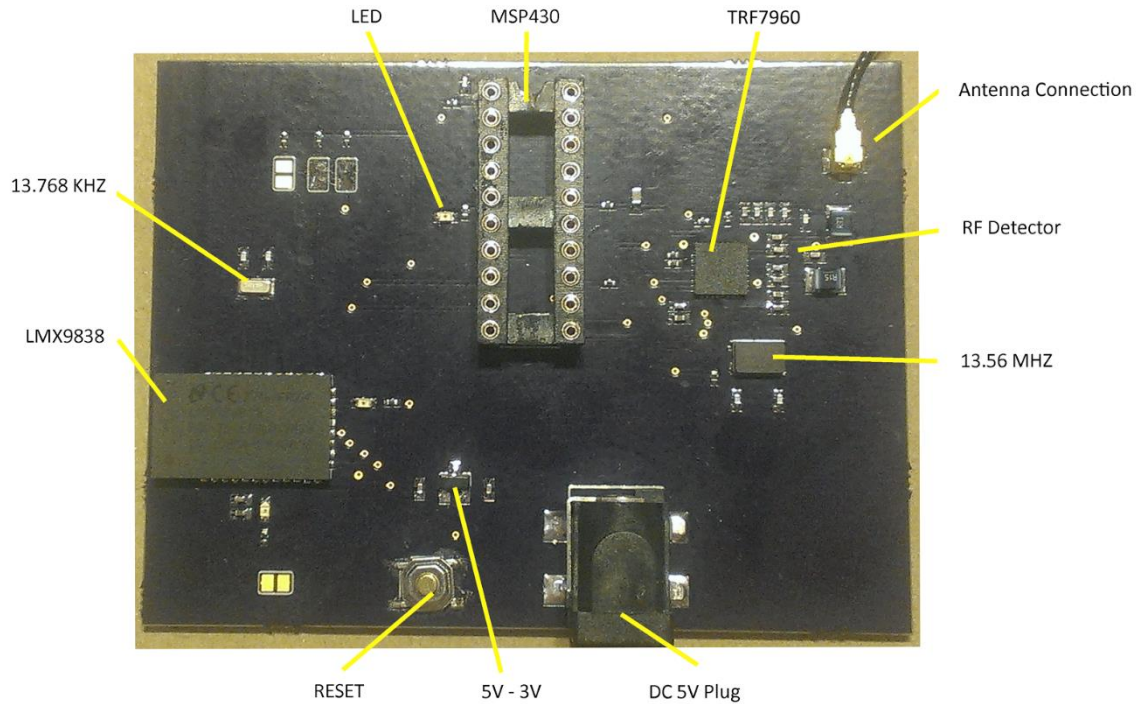


Figure 5.2.1 Manufactured Prototype Scanning Unit

The finalized construction of the Schematic and PCB layout as resulted in a self-contained board that can be placed in a basic housing unit as indicated in figure 3.4.2. With a 5V DC input from either a large battery for mobility or a wall mounted plug for standard use, this board is set up to only require the RF Antenna to be screwed into an external jack on the mounting box. The Bluetooth, RF demodulation, and any tag information decoding is handled by the onboard microprocessor.

7 Project Prototype Testing

The group required a set of testing scenarios and required specifications in order to ensure the quality and functionality of the project system. The majority of the testing cases were full system integration tests, as can be seen in Table 7.1.1.

<u>Required Specifications Testing</u>	
Requirement	How it will be Tested
Must meet airport transmission standards	The components will be tested to see if they exceed the defined standards.
RF Receiver must accurately scan the RFID chip 95% of the time	Scanning unit will be run several times and accuracy averaged out of the recorded results.
System must accurately identify who the luggage belongs to 95% of the time	Full system will be run several times and accuracy averaged out of the recorded results.
Must notify the correct recipient with a maximum delay of 60 seconds.	The full system will be run to ensure the delay is no greater than 60 seconds when receiving the notification.
Must not exceed funds of \$2,000 to construct	The components will be selected to not exceed the funds of construction.
Will use wall outlet to power scanners and microcontroller.	System will be checked to ensure it supports the wall outlet power supply before being plugged in and tested.
Must not have any negative impact on the environment or personnel	The system will be put in demo use for a week and any negative environmental or personnel changes will be monitored.
Must meet size requirements to allow clearance for luggage	Luggage bag dimensions will be measured using a ruler or tape measure.
Must not emit any harmful radiation and is safe to handle	Power transmission levels will be measured and kept at the minimum required values for functionality
Information the system transmits must be secure and not allow unauthorized access.	Additional Bluetooth devices will be used to attempt to break the strict pairing of the Bluetooth devices.

Table 7.1.1 - Requirements related Testing

7.1 Hardware Testing Environment

Ideal location would be at an airport in the working environment the system is supposed to be used in. This environment would consist of a conveyer system, either single level or multiple. The scanning equipment will need to be able to scan with operational interference from machinery, and ability to transfer data wirelessly at a distance through at least a single brick layer wall. The individual parts of the hardware system will be located at an appropriate distance, but this distance is determined by airport layout.

Since the system is split up into the scanning unit, the processing master unit, and the monitor display device, the maximum distance and reliability will be determined by the interaction of each individual node. The distance from the scanning unit to the master unit, and the master unit to the display unit. Combined distance will determine the maximum placement of each device within the network.

Another factor will be the wifi connection of the master unit with the local network or internet. Since offloading customer data for text message sending will need access to a network with internet access or direct access to the internet.

7.2 Hardware Specific Testing

Since the majority of the system is software based, the hardware acts as relays to pass information from subsystem for data processing. Specific test included mock data being loaded into the system and determining proper transmission of the data to each device in the subsystems..

This included, testing data transmission of the RFID scanner by scanning a tag, checking internally within the microprocessor that the data was received through an LED indicator, and on chip activity through LED to verify the operation of the device.

Secondary test to verify transmission of data received by the microprocessor in the scanning unit was to load mock information, a full tag UID and baggage information was loaded into the system, and let to be parsed out by the Pi for proper breakdown for display.

Within the main unit a full software tests was done for the interaction between the wifi and airport network, and the bluetooth communication back to the scanning unit for command and control.

7.3 Software Testing Environments

7.3.1 Emulators

7.3.1.1 Raspberry Pi

For the software testing environment, we emulated the Raspberry Pi. By emulating the Raspberry Pi, we were able to perform unit testing and check other functionality of the Java ME Embedded 8.1 Runtime Environment prior to running the same tests on the physical device. Our group used the embedded emulator provided by Oracle in the Java ME Embedded 8.1 development kit.

7.3.1.2 Android Stick

For the software testing environment, we emulated the Android Stick. By emulating the Android Stick, we were able to perform unit testing and check other functionality of the Android Operating System prior to running the same tests on the physical device. Emulating the Android Stick was very straightforward. Since the Android Stick will be running the Android Operating System 4.4 KitKat, we simply needed to use one of the many Android Operating System 4.4 KitKat emulators available to download for free.

7.3.2 Mocked Airline Lookup Service

For the software testing stages and for demoing, the airline endpoint paths service have been mocked. We used PHP Version 5.6, since it is the most recent version. We did not take advantage of any extra PHP plugins for our mocked web services, since we planed our mocked web services to be extremely basic. They simply need to emulate the results of a query to a real world web service, not the procedures a real world web service would follow for a query.

This mocked instance is hosted online, at LivingBucket.com, and is a simple php script containing a map of fake airlines and endpoint paths to their passenger notification services. The script accepts a payload containing a valid fake airline IATA code, and the script returns the endpoint path mapped to that IATA code.

In a real world scenario, the web service would be connected to a simple database containing the airline mappings. However, for the purpose of simplifying our mocking process, we stored the airline mappings as an in-memory hash table, where the key is the airline's IATA code and the value is the endpoint path.

7.3.3 Mocked Passenger Notification Services

For the software testing stages and for demoing, the airline passenger notification services was mocked. These mocked instances were hosted online, at LivingBucket.com, and are simple php scripts containing a map of fake passenger data. The script accepts a payload containing the lookup key for a passenger, and the script returns the passenger information.

In a real world scenario, the web service would depend on a baggage database to find the passenger's notification information. However, for the purpose of simplifying our mocking process, we stored the baggage data as an in-memory hash table, where the baggage identification code is the key, and the passenger information is the value.

7.4 Software Specific Testing

7.4.1 Microcontroller Testing

7.4.1.1 Integration Testing

7.4.1.1.1 Scanner Input Integration Testing

The code for reading the input from the scanner was thoroughly tested by using an integration test. Our group has determined that, since unit testing the microcontroller code would be impractical due to its tight coupling with the hardware, we proceeded directly to integration testing. The integration testing for the Scanner Input involved us manually scanning tags and ensuring the tags' raw data can be found in the correct registry locations. In order for our integration testing for this portion of the project to pass successfully, we must be able to find the raw data, in its entirety, at the exact location we expect to find it in for every single tag we test with.

7.4.1.1.2 Bluetooth Output Integration Testing

The code for transmitting data over the Bluetooth chip was thoroughly tested by using an integration test. Our group has determined that, since unit testing the microcontroller code would be impractical due to its tight coupling with the hardware, we proceeded directly to integration testing. The integration testing for the Bluetooth Chip Output involved us creating fake data in the Microcontroller Unit's registry and instructing the Microcontroller Unit to transmit the data with its Bluetooth chip. We then receive the data on another device and check it for accuracy.

7.4.1.1.3 Full System Integration Testing

The full system integration testing for the Microcontroller Unit is used to ensure that we are able to manually scan tags and receive the tag's raw data on a separate device via a Bluetooth connection. In order for the Microcontroller Unit's full system integration test to pass successfully, we made sure it was able to correctly receive the raw tag data, in its entirety, on our secondary testing device for every single tag we test it with.

7.4.1.2 Performance Testing

7.4.1.2.1 Scanner Input Performance Testing

Performance testing for the Microcontroller Unit is extremely important for the success of the project. The Master CPU will be waiting for information, so the Microcontroller Unit needs to be able to respond quickly. One of the major potential bottlenecks for the Microcontroller Unit is how quickly it is able to scan the baggage tags. Since this is such a significant potential bottleneck, the group performed rigorous performance testing on it. Data on time delays from when a tag is scanned to when it's processed and the system disseminated the data to each component is charted in section 7.5.

7.4.1.2.2 Bluetooth Output Performance Testing

Another of the major potential bottlenecks for the Microcontroller Unit is how quickly it is able to transmit the raw tag data with its Bluetooth chip. Since this is such a significant potential bottleneck, the group performed rigorous performance testing on it. In order for the Bluetooth Output performance test to pass successfully, the system must be able to prepare the Bluetooth Chip and transmit the stored data as quickly as possible. To accomplish this task we increased the rate of connections between the bluetooths to poll the scanning unit faster than the required tag scanning of the previous test. This means that any time a tag was available for transmission, the bluetooth connection would already be waiting.

7.4.1.2.3 Full System Performance Testing

The entire Microcontroller Unit was subjected to a strict performance test. Based on our analysis of a real-world worst-case scenario, the group expects the microcontroller unit to be able to scan an RFID tag and finish transmitting the data over Bluetooth in roughly 250 milliseconds.

7.4.2 Main CPU Testing

7.4.2.1 Model Unit Testing

7.4.2.1.1 Baggage Object Unit Testing

Our group performed standard Java Object Unit Testing for the Baggage Object, because it is a normal Java Data Object with little to no unique functionality other than to store data in an organized fashion. To meet our code coverage requirements, all nine getter methods and all nine setter methods are unit tested for basic getter and setter functionality. These unit tests were done for the sake of regression testing.

7.4.2.1.2 BaggageCache Unit Testing

The unit testing for the BaggageCache class involved testing the getter method for the baggage cache LinkedHashMap, as well as testing the functionality of each of the methods. The *addBaggage(baggage : Baggage)* method was tested to ensure the *baggage* object is always added to the cache in the correct order, and it was tested to ensure the method did not add anything to the cache if the *baggage* object is *null*. The *getBaggage(key : String)* method was tested to ensure the correct Baggage Object was returned for the given *key* value. The method was also tested to ensure that a *null* is returned if the key is not found in the LinkedHashMap. The *updateBaggage(key : String, timeout : Date)* method was tested to ensure the Baggage Object associated with the *key* value is correctly updated with the *timeout* value. The method was also tested to ensure the program would not crash with a *Null Pointer Exception* in the event where the sent key is not associated to any Baggage Object in the LinkedHashMap. The method was tested to ensure a Baggage Object is not updated with a *null timeout* value. The *updateCache()* method was tested to ensure it correctly removes expired Baggage Objects from the LinkedHashMap baggage cache.

7.4.2.1.3 MasterController Unit Testing

The unit testing for the MasterController class involves testing the methods in the class. The *findBaggage(key : String)* method was tested to ensure the BaggageCache's *getBaggage(key : String)* method is correctly called.

The *processBaggage(...)* method was tested to ensure that it correctly processes the incoming baggage data. The method was first tested to ensure the BaggageTagService is called to parse the baggage data and create a new Baggage Object.

If the BaggageTagService fails to parse the baggage data, an error message is logged and the *processBaggage(...)* method exits. Otherwise, the method

findBaggage(key : String) method is called correctly using the newly created Baggage Object. In the case where the Baggage Object is not found in the cache, the method *findAirlineEndpoint()* method is called. If the *findAirlineEndpoint()* method returns a null value, the program will log an error message and continue with normal operation. If the endpoint is correctly found, the method AirlineNotificationService is called. The final step of this case is to add the created Baggage Object to the BaggageCache. In the case where the Baggage Object was originally found in the cache, the method was tested to ensure the cache is correctly updated with a new timeout value. Regardless of whether or not the Baggage Object was originally found in the cache, the *processBaggage(...)* method is then tested to ensure the *updateDisplay()* method is correctly called.

The *addBaggage(baggage : Baggage)* method was tested to ensure the BaggageCache's *addBaggage(baggage : Baggage)* method is correctly called. The *findAirlineEndpoint()* method was tested to ensure that if an airline mapping exists in the *endpoints* HashMap, the existing airline endpoint path is returned. However, in the case where the mapping does not exist, it should perform a query to the AirlineEndpointService. The found endpoint path from the service call should be added to the *endpoints* HashMap prior to the method returning.

The *updateCache()* method was tested to ensure the BaggageCache's *updateCache()* method is correctly called. The *updateDisplay()* method was tested to ensure that the DisplayUpdateService is correctly called.

7.4.2.2 Services Unit Testing

7.4.2.2.1 BaggageTagService Unit Testing

The BaggageTagService was tested to ensure the *createBaggage(Raw Bluetooth Data)* method operates correctly, and always returns an instance of a BaggageResponse Object. If the data entering the method is unable to be parsed correctly, the method was tested to ensure the returned BaggageResponse contains a null Baggage Object and that the *success* variable is set to *false*. If the data entering the method is correctly parsed, the method will be further tested to ensure the returned BaggageResponse contains a valid Baggage Object with correct values and that the *success* variable is set to *true*.

7.4.2.2.2 AirlineEndpointService Unit Testing

The AirlineEndpointService was tested to ensure the *getAirlineEndpoint(airlineCode : String)* method operates correctly, and always returns an instance of an EndpointResponse Object. In the case where the airline code does not exist in the testing web service, the method was tested to ensure the returned EndpointResponse contains a null String for the *endpoint* variable and that the *success* variable is set to *false*. In the case where the airline code does

exist in the testing web service, the method was tested to ensure the returned EndpointResponse contains the correct web service endpoint for the airline and that the *success* variable is set to *true*.

7.4.2.2.3 AirlineNotificationService Unit Testing

The AirlineNotificationService was tested to ensure the *notifyAirline(endpoint : String, payload : Payload)* method operates correctly. This method does not return a value, so this method was tested to ensure the method correctly exists if either the *endpoint* or *payload* variables are invalid. The method was also tested to ensure the json message created from the *payload* variable is correct.

7.4.2.2.4 DisplayUpdateService Unit Testing

The DisplayUpdateService was tested to ensure the *updateDisplays(baggage : LinkedList<Baggage>)* method operates correctly. This method does not return a value, so this method was tested to ensure the method correctly exists if the *baggage* variable is invalid. The method was also tested to ensure the messages being sent to the display devices are correct.

7.4.2.3 Integration Testing

7.4.2.3.1 Bluetooth to Notification Service Integration Testing

An integration test for the Bluetooth to Notification Service was created. This test ensured that the incoming bluetooth data to the MasterController's processBaggage(...) method can follow all of its steps all the way through and successfully call the AirlineNotificationService's notifyAirline(endpoint : String, payload : Payload) method.

7.4.2.3.2 Bluetooth to Display Update Service Integration Testing

An integration test for the Bluetooth to Display Update Service was created. This test ensured that the incoming bluetooth data to the MasterController's processBaggage(...) method can follow all of its steps all the way through and successfully call the DisplayUpdateService's updateDisplays(baggage : LinkedList<Baggage>) method.

7.4.2.3.3 Full System Integration Testing

An integration test for the Bluetooth to Display Update Service was created. This test ensured that the incoming bluetooth data to the MasterController's processBaggage(...) method can follow all of its steps all the way through and successfully call both the AirlineNotificationService's notifyAirline(endpoint : String,

payload : *Payload*) method and the *DisplayUpdateService*'s *updateDisplays(baggage : LinkedList<Baggage>)* method.

7.4.2.4 Performance Testing

7.4.2.4.1 Bluetooth Incoming and Outgoing Performance Testing

Our group did performance testing on the portion of the Main CPU which receives incoming Bluetooth data and processes it. This test was important because it showed us how efficient our data parsing processes are. This test provides us with results which indicated if we are taking too long to parse the incoming data.

Additionally, our group performed testing on a portion of the Main CPU which creates the Bluetooth message to send to the Android Sticks and sends it. This test is important because it shows us how efficient our Bluetooth message creation and our Bluetooth sending processes are. This test provides us with a result which indicated if we are taking too long to create and send the messages.

7.4.2.4.2 Wifi Incoming and Outgoing Performance Testing

Our group did performance testing on the portion of the Main CPU which sends data across wifi to our servers, waits for a response, and then parses that response. This test is important because it showed us how efficiently we are interacting with our testing server. This test provides us with results which indicates if our code is inefficient.

7.4.2.4.3 Cache Management Performance Testing

Our group did performance testing on the cache management subsystem of our Main CPU. This is the portion of the system which adds and removes values from our internal baggage cache. The performance of this subsystem is critical to the performance of our entire system. If the cache management subsystem runs slowly, all aspects of the system will be detrimentally impacted. These tests provided us with results which indicates our codes efficient.

7.4.3 Android Stick Testing

7.4.3.1 Model Unit Testing

7.4.3.1.1 Baggage Object Unit Testing

Our group performed standard Java Object Unit Testing for the *Baggage* Object, because it is a normal Java Data Object with little to no unique functionality other than to store data in an organized fashion. To meet our code coverage

requirements, all three getter methods and all three setter methods are tested for basic getter and setter functionality. These unit tests exist for the sake of regression testing.

7.4.3.1.2 BaggageCache Unit Testing

The unit testing for the BaggageCache class involved testing the getter method for the list of Baggage Objects. Standard Java Object Unit Testing was used because the BaggageCache does not have any unique functionality. These unit tests exist for the sake of regression testing.

7.4.3.2 UpdateDataService Unit Testing

The UpdateDataService class was tested to ensure the *updateData(Json Data)* method works as expected. The method was tested to ensure the incoming data is correctly parsed into a list of Baggage Objects. The method was also tested to ensure the BaggageCache is correctly accessed and correctly updated with the new Baggage data. This method was also tested to ensure it triggered the table view's update functionality to render the user interface.

7.4.3.3 Full System Integration Testing

The system for the Android Stick is fairly basic, so only the full system integration test will be necessary. In this test, we ensured the system works from start to end. The test sent Bluetooth data to the UpdateDataService class and check the created user interface table for accuracy. This test passes only under the condition that the table has been updated perfectly.

7.4.3.4 Performance Testing

7.4.3.4.1 Bluetooth Incoming Performance Testing

Our group performed testing on the portion of the Android Stick which receives incoming Bluetooth data and processes it. This test is important because it will show us how efficient our data parsing processes are. This test provided us with a result which indicated if we are taking too long to parse the incoming data.

7.4.3.4.2 Display Updating Performance Testing

Our group performed testing the portion of the Android Stick which updates the user interface table. This portion of the system is important because if we are unable to update the table efficiently enough, the entire purpose of the display becomes pointless. We need to ensure the table is updated in a quick enough time such that the Android Stick is ready and able to process any additional data the Main CPU sends it. The test results show if we were able to update the user interface in a reasonable amount of time.

7.6 Project Testing

Basic testing was done using the LED lights installed in the layout planning stage. These light indicated if we were connected to the device, and if data was being transmitted.

In Figure 7.6.1 the red LED indicated to us if the Bluetooth connection was established. When the red light was out, an active connection was established. During the course of testing, any time the red light would activate, it indicated a loss of connection. Outside of a proper connection, data flow was also needed to be visually represented to make sure that data is in fact being sent over the established connection. This was done by using the yellow LED, as data activity is present on the Bluetooth buffers, the light will blink as data is serially sent out of the device. If a tag is scanned, and the yellow light flashes to represent all the hexadecimal values of the tag being sent out of the scanning unit, then the unit is operating correctly.

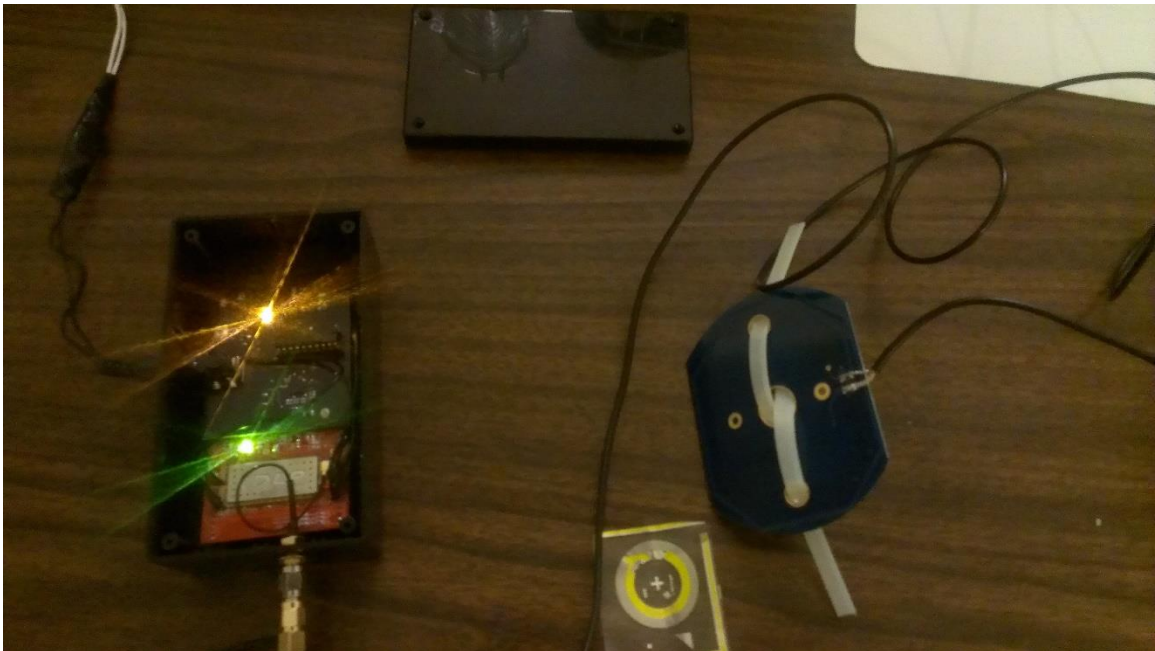


Figure 7.6.1 RFID Scanning unit w/tag

Due to an issue in either the TRF chip malfunctioning during shipping or installation, the RF modulation signal was out of tolerance of what an ideal carrier wave should have been. To resolve this issue in testing, we had to order a premade 13.56 MHz board that could modulate and demodulate the TRF chip. Attached is a premade board from DLP, who worked with Texas Instruments on the TRF RFID

project. This board contains only the TRF chip, and the modulation circuitry, and can be seen in Figure 7.6.2. In the final testing, since the tag was not being properly activated, we suspected the TRF chip to be bad. All connection to the onboard TRF chip was severed, except for the power regulation line to the MSP430, and the new TRF chip was wired to the scanning board in order attach a working modulation path for the tag. The new board was wired in parallel to the old circuitry directly to the MSP430 as shown in Figure 7.6.3.

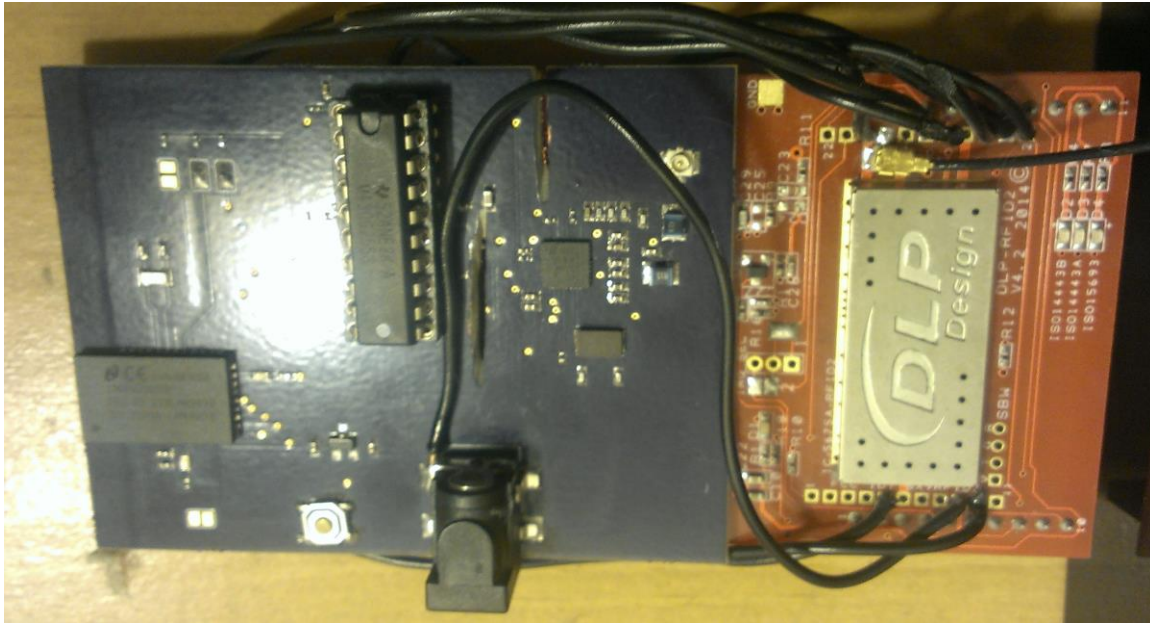


Figure 7.6.2 Augmented Scanning Unit front

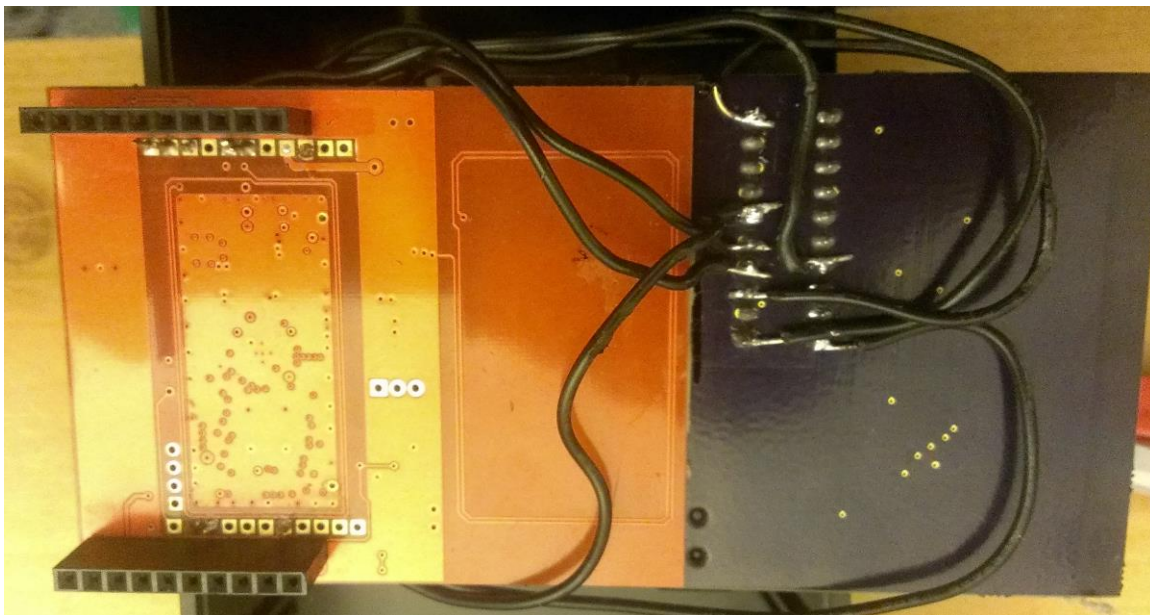


Figure 7.6.3 Augmented Scanning Unit back

With the new TRF board in place, and secured to the new board, each tag scanned was properly demodulated and was able to be seen by the MSP chip. The rest of the circuitry of the power supply and Bluetooth remained intact, and additionally runs the newer board that was attached.

The final outcome of the completed project demonstration is showed in figure 7.6.1, where the monitor is displaying tag information, and the plastic shelf is acting as the scanning area where luggage would flow underneath. Located to the right of the monitor is the Scanning unit, and behind it is the Raspberry Pi. Each tag and its UID are associated with a personal device. These devices are above the shelf with the scanning antenna attached, they are in the form of phones and tablets. As a tag is scanned under the antenna, the customer is displayed on the monitor as well as a notification would ring and display on the tablet associated with that tag.



7.6 Measurable Test Results

The group ran ten tests with the system and collected different bits of data during those tests. The total amount of time it took to scan the baggage tag, to when the bag was displayed on the screen was measured. This data can be seen in Table 7.5.1.

Test Attempt	Time
1	1.1
2	1.53
3	1.25
4	1.21
5	1.26
6	1.2
7	1.11
8	1.1
9	1.15
10	1.15

Table 7.5.1 - Amount of time to scan and display the baggage tag data.

The time for each test was recorded and found to be an average 1.2 seconds. The total amount of time it took to scan the baggage tag, to when the passenger was notified that their baggage is on the carousel can be seen in Table 7.5.2.

Test Attempt	Time
1	1
2	1.43
3	1.15
4	1.1
5	1.16
6	1.19
7	1.1
8	1.1
9	1.09
10	1.13

Table 7.5.2 - Amount of time to scan the baggage tag and notify the owner.

The time for each test was recorded and found to be an average 1.5 seconds. The group also kept note of how many bags were successfully identified. The group saw a 100% success rate for bag identification and information parsing when a tag was detected.

Tag data reading reliability was based on the distance from unit tests done. At the tag to scanner area, any tag at or over 4.5 inches would result in a possibility of no tag reading being too far from the radiation field to power up. The Bluetooth device transmitting the tag information showed that it was able to clear a sizable distance while maintaining connection and reliable data over 10 meter from the device., upwards of 20 meters from the device, data was delayed as error bits held up the transmission of the correct data. This resulted in the Bluetooth buffer waiting a longer time, but still transmitting the available data as the buffer unloaded everything at once.

For situation where a wall interrupted line of sight, a single cinder wall reduced the distance of readability in half, resulting in a 10 meter or less range, still within acceptable distance based on conveyer layouts. Any additional walls that obstructed line of site with the device caused a complete disconnect from the device, as no data was being received.

Note that the device used for testing is subject to its own power and range of connection. Any final product would require testing of each individual setup, to determine how far apart and what obstacles are in the way, to better tailor the Bluetooth device to the environment it will be broadcasting in.

During testing, the group also made several other discoveries. The group found that the maximum length the scanner was capable of reading the baggage tag from is 11.5 centimeters. The group also found that the tag data would occasionally be incomplete. After a lengthy investigation, the group found that the reason for the incomplete tag data was the tag leaving the edges of the RF radiation zone. Since the tag was not completely in the RF radiation zone, only some of data was correctly read by the Scanner Unit as the RF field diminished and turned the passive tag off mid transmission.

The group also noticed the Master CPU was requesting tag information from the Scanning Unit too quickly and the Scanning Unit was frequently not working. The group implemented a delay in the Master CPU between scan times. The group first tried a delay of one second, but then found that to be way too large, and also violated the update times required for proper tag scanning times. The group eventually found that a ten millisecond delay yielded the best result, which is below the 250 millisecond delay overall that we restricted the scan update times to be within.

8 Administrative Content

8.1 Project Milestones

The group has planned out several milestones to be completed throughout the design and prototyping stages of the project's completion. They are outlined in the following Tables 8.1.1 through 8.1.5.

Summer 2015	Ernest	Adrian	Tomasz
May 31- June 6	-Compose Project Proposal -Design Documentation	-Compose Project Proposal -Design Documentation	-Compose Project Proposal -Design Documentation
June 7 - June 13	-Research and define Hardware Design	-Research and define Software Design	-Research and define Software Design
June 14 - June 20	-Research and define Hardware Design	-Research and define Software Design	-Research and define Software Design
June 21 - June 27	-Research implementation of code on Microcontroller	-Research repository and mocked services	-Research SMS functionality
June 28 - July 4	-NCR Sponsor Pitch	-NCR Sponsor Pitch	-NCR Sponsor Pitch
July 5 - July 11	-Research and define additional components	-Research and define additional components	-Research and define additional components
July 12 - July 18	-Work on Microcontroller Schematic	-Test Raspberry Pi OS	-Test Raspberry Pi OS

Table 8.1.1 - Milestones for the project for Summer 2015

Summer 2015	Ernest	Adrian	Tomasz
July 19 - July 25	-Combine and draft report for review	-Combine and draft report for review	-Combine and draft report for review
July 26 - Aug 1	-Finalize Senior Design 1 Paper	-Finalize Senior Design 1 Paper	-Finalize Senior Design 1 Paper
Aug 2 - Aug 8	-Submit Senior Design 1 Paper	-Submit Senior Design 1 Paper	-Submit Senior Design 1 Paper

Table 8.1.2 - Milestones for the project for Summer 2015 continued

Break	Ernest	Adrian	Tomasz
Aug 9 - Aug 15	-Finalize PCB layout	-Develop Abstraction Library	-Develop Android Shared Library
Aug 16 - Aug 22	-Contact PCB manufacturers	-Develop Master CPU Model	-Develop Android Stick and Android App Model

Table 8.1.3 - Milestones for the project during Break

Fall 2015	Ernest	Adrian	Tomasz
Aug 23 - Aug 29	-Order Parts for the Microcontroller	-Develop Master CPU Services	-Develop Android Stick Services
Aug 30 - Sept 5	-Order PCB	-Develop Master CPU Services	-Develop Android Stick Services
Sept 6 - Sept 12	-Work on Microcontroller programing	-Develop Mocking Webservices	-Develop Android App Services
Sept 13 - Sept 19	-Work on Microcontroller connections	-Integrate Master CPU Services	-Integrate the Android Stick Services
Sept 20 - Sept 26	-Test Microcontroller code	-Develop Master CPU Cache Management	-Integrate the Android App Services
Sept 27 - Oct 3	-Build demo hardware	-Install and test code on Raspberry Pi	-Install and test code on Android Stick
Oct 4 - Oct 10	-Build demo hardware	-Integration Tests with Raspberry Pi and Android Stick	-Integration Tests with Raspberry Pi and Android Stick
Oct 11 - Oct 17	-Integration Tests with Raspberry Pi and Microcontroller	-Integration Tests with Raspberry Pi and Microcontroller	-Integration Tests with Raspberry Pi and Android Stick
Oct 18 - Oct 24	-Integration Tests with Raspberry Pi and Microcontroller	-Integration Tests with webservices and Android App	-Integration Tests with webservices and Android App
Oct 25 - Oct 31	-Integration Tests with full system	-Integration Tests with full system	-Integration Tests with full system
Nov 1 - Nov 7	-Test Prototype	-Test Prototype	-Test Prototype

Table 8.1.4 - Milestones for the project for Fall 2015

Fall 2015	Ernest	Adrian	Tomasz
Nov 8 - Nov 14	-Microcontroller Mounting and Housing	-Raspberry Pi Mounting and Housing	-Raspberry Pi Mounting and Housing
Nov 15 - Nov 21	-Complete and polish Prototype	-Complete and polish Prototype	-Complete and polish Prototype
Nov 22 - Nov 28	-Combine and draft report for review	-Combine and draft report for review	-Combine and draft report for review
Nov 29 - Dec 5	-Submit Senior Design 2 Paper -Present Prototype	-Submit Senior Design 2 Paper -Present Prototype	-Submit Senior Design 2 Paper -Present Prototype

Table 8.1.5 - Milestones for the project for Fall 2015 continued

8.2 Budget & Financing

8.2.1 Expected Costs

The group had expected costs of around \$1500, assuming two RFID receivers can be obtained for \$500, \$100 for the microcontroller board to be constructed, \$100 for Raspberry Pi and necessary components, as well as a buffer of \$300 for any additional expenses and hardware enclosure/mounting. These costs do not include any hours paid for labor or working environments.

8.2.2 Financing

Due to lack of project sponsors and the expected costs exceeding the fully self-financed group, the group had been forced to scale down on the demonstration aspect of the presentation and will work with a smaller RFID Receiver range. Not all of the additional features desired for development could have been included as a result of this circumstance. The team's budget totaled \$400.

Item	Quantity	Acquired Price	Net Worth
Raspberry PI 2 Model B	1	\$43.00	\$43.00 + shipping
4GB microSD card w/ Adapter	1	\$5.99	\$5.99 + shipping
Generic PL2303HX USB To TTL To UART RS232 COM Cable Module Converter	1	\$4.59	\$4.59 + shipping
TRENDnet Micro-Bluetooth USB 3.0 Adapter	1	\$12.19	\$12.19 + shipping
Edimax EW-7811Un Wi-Fi USB Adapter	1	\$9.23	\$9.23 + shipping
MK808B Plus - Android 4.4 Stick	1	\$28.77	\$28.77 + shipping
Asus VE248 monitor	1	\$0.00	\$177.99
PCB	1	\$120	Varies
Mounting Costs		?	?
RI-I02-112A-03 Tag-it(TM) HF-I Plus Transponder Inlays	1	\$1.68	\$1.68
LMX9838SB	1	\$0	\$31.39
TRF7970ARHBR	1	\$0	\$6.98
MSP430G2553IN20	1	\$0	\$2.80
LP3963ES-3.3	1	\$0	\$4.45
Capacitor	24	\$2.01	\$2.01
Inductor	2	\$1.00	\$1.00
Resistor	14	\$0.46	\$0.46
Crystals	2	\$1.88	\$1.88

Table 8.2.1 - Project expenses and financing

Item	Quantity	Acquired Price	Net Worth
LED	3	\$0.84	\$0.84
Misc.	8	\$10.31	\$10.31
Demonstration costs		\$100	\$100
Total		341.95	565.56

Table 8.2.2 - Project expenses and financing continued

8.3 Advisors

Since one of the group members had a patent on a potential new product, we decided to meet with the company which held the patent rights [7]. The group communicated with NCR, National Cash Register, for some insight on possible implementations of our creation. During the meeting, it was concluded what the best way to communicate with the airlines was, and how to handle the notification system for the customers. On top of the discussion about the project at hand, the group also managed to see how current single board computer systems are used in modern Kiosk construction and the different types of RFID cards currently in use.

8.4 Facilities & Equipment

The core of the project was centered mostly on software implementations and therefore did not require any additional facilities or equipment. The group members responsible for the software development have already existing workstation environments that fulfilled the needs of the project. The UCF Lab section was utilized during Senior Design 2 to construct and assemble the required hardware components of the project.

9 Project Summary

The group successfully finished all the milestones set out for the first semester of Senior Design. After taking the time to divide roles and group scheduling at the very beginning, the group spent the majority of time invested in research and design. The group gradually became knowledgeable on the unfamiliar topics regarding the project and scheduled regular meetings to discuss the design aspects of the project.

After allocating a budget, the different subsystems were defined based on the initial specifications. Following the design definitions, requirements were created and the appropriate hardware and software components were properly selected. A variety of constraints, technical and financial, limited the possible options for the project's implementation. However, the goals and objectives were pursued and met to a satisfactory level for the C.L.A.I.M system.

Several subsystems were designed and developed that the group had no previous knowledge of working with. The experience gained from working on these devices proved to provide more information that went beyond the scope of the cumulative education prior to the start of the project. This forced the group to utilize outside resources to fully learn and comprehend some of the research components. There were some initial difficulties with the microcontroller board design for the scanning unit, but they were quickly smoothed out after group collaboration. There were additional concerns regarding the Bluetooth communication between the Master CPU and the other devices, but those were resolved as well.

The complete design and prototype process of the project helped the group members gain necessary experience in developing and implementing possible future projects in the workforce. The group hopes the work presented in the project inspires future groups that wish to look into this relatively untapped RFID technology and potentially improve on the design.

10 Appendix

Appendix A - Citation

[1] Department of Transportation, 'Department of Transportation', 2015. [Online]. Available: https://www.transportation.gov/sites/dot.gov/files/docs/2015MayATCR_1.pdf. [Accessed: 05- Aug- 2015].

[2]R. Journal, 'Hong Kong Airport Says It Now Uses Only RFID Baggage Tags - RFID Journal', *Rfidjournal.com*, 2015. [Online]. Available: <http://www.rfidjournal.com/articles/view?4885>. [Accessed: 05- Aug- 2015].

[3] Future Travel Experience, 'Air France-KLM unveils new permanent bag tag and bag tracking device', 2014. [Online]. Available: <http://www.futuretravelexperience.com/2014/03/air-france-klm-new-permanent-bag-tag-and-tracking-device-can-benefit-the-entire-industry/>. [Accessed: 05- Aug- 2015].

[4] Reboundtag.com, 2015. [Online]. Available: <https://www.reboundtag.com/>. [Accessed: 05- Aug- 2015].

[5] Vanguardid.com, 'ViewTag', 2015. [Online]. Available: <http://www.vanguardid.com/PermanentRFIDLuggageTags.aspx>. [Accessed: 05- Aug- 2015].

[6] Qantas.com.au, 'Next Generation Check-in Regional Implementation Begins', 2015. [Online]. Available: <http://www.qantas.com.au/travel/airlines/media-releases/mar-2011/5084/global/en>. [Accessed: 05- Aug- 2015].

[7] McGrath; Adrian, "Baggage Delivery Notification System and Method" U.S. Patent 20140210623, issued July 31, 2014.

Appendix B -Copyright Permissions

All images from Texas instruments are contained within Datasheets provided by Texas Instruments for their respective device. Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices.

"Zoedrink, Handscanner, May 6 2006, Creative Commons License
<https://www.flickr.com/photos/semfira/141350396/>"

“Joana e Miguel Paulo Pardal, RFID Journal Life 2009 (40), April 28 2009, Creative Common License,
<https://www.flickr.com/photos/pardaisjr/3492104325/in/album-72157617480608591/>”

Appendix C - Datasheets

Application Report SLOA138–April 2009
Implementation of the ISO15693 Protocol in the TI TRF796x

Application Report SLOA199–July 2014
TRF7970A + MSP430G2xx NFC/RFID Module Reference Design

TRF7970A SLOS743K –AUGUST 2011–REVISED APRIL 2014
TRF7970A Multiprotocol Fully Integrated 13.56-MHz RFID and Near Field Communication (NFC) Transceiver IC

RI-I02-112A-03, RI-I02-112B-03 SCBS833B –NOVEMBER 2006–REVISED JUNE 2014
RI-I02-112X-03 Tag-it™ HF-I Plus Transponder Inlays Large Rectangle

MSP430G2x53 MSP430G2x13 SLAS735J –APRIL 2011–REVISED MAY 2013
MIXED SIGNAL MICROCONTROLLER

LM3480 SNVS011G –JUNE 1999–REVISED FEBRUARY 2015
LM3480 100-mA, SOT-23, Quasi Low-Dropout Linear Voltage Regulator

LMX9838 SNOSAZ9F –JULY 2007–REVISED DECEMBER 2014
LMX9838 Bluetooth Serial Port Module