SMART CART – AN ENHANCED SHOPPING EXPERIENCE

By

Nikhil Raman

Kartik Sanghi

Rohan Singh

Final Report for ECE 445, Senior Design, Spring 2012

TA: Justine Fortier

01May 2012

Project No. 41

Abstract

This report explains in detail the functional design for the Smart Cart device prototype built to make the shopping experience time efficient from the shopper's point of view and cost efficient from Wal-Mart's point of view.

The device essentially maintains a real time list of details of cart contents and provides a convenient payment solution when shopping is finished which eliminates the long waiting periods at the checkout counters. Our calculations have shown that the overall shopping time is considerably reduced through the use of the Smart Cart device.

Contents

1. Introduction

We designed and built a device which maintains a real time list of contents in a shopping cart. The barcodes of the items placed in the cart were scanned and the corresponding data was then made available to the shopper. In order to prevent theft, we included a weight monitoring system based on FlexiForce weight sensors to compare weights between the product database and weight on cart. Although we were unable to implement a payment system due to time constraints, it can be added on later since the wireless communication setup already present can easily be leveraged.

1.1 Project Motivation

The smart cart will be an all in one cart. It will allow the user to keep track of the total cost as and when items are added to the cart. It will also communicate wirelessly with an in store component to make easy payments on the go. In the case of any ambiguity, the shopper will also have the option of going up to the checkout counters. This new system would reduce the long wait times at the checkout counters, increase the efficiency of the checkout procedure, and would provide the shopper with up to date cost and total information which makes the whole experience more convenient.

1.2 Project Objectives

This project aims to design a system which reads the barcode on each item that is placed in the cart and updates the product information which is available to the shopper. Pressure/Weight sensors will be used to detect the presence of new items in the cart.

The barcode scanner extracts the barcode which is transmitted to the microcontroller through an USB connection. The microcontroller reads information from a SD card inserted into the microcontroller. This SD card has all the information about the product. This data is then formatted and presented to the user for review and confirmation on a LCD screen.

New items in the cart will be detected by tracking the change in the output of pressure sensors. The same sensors will be used to detect when items are removed from the cart. A program will be implemented to confirm the removal from the shopper's shopping cart. Another program will be implemented to work as an anti-theft mechanism to prevent the shopper from leaving without a successful payment.

1.3 Project Modules

Figure 1 shows the functional block diagram for our device:

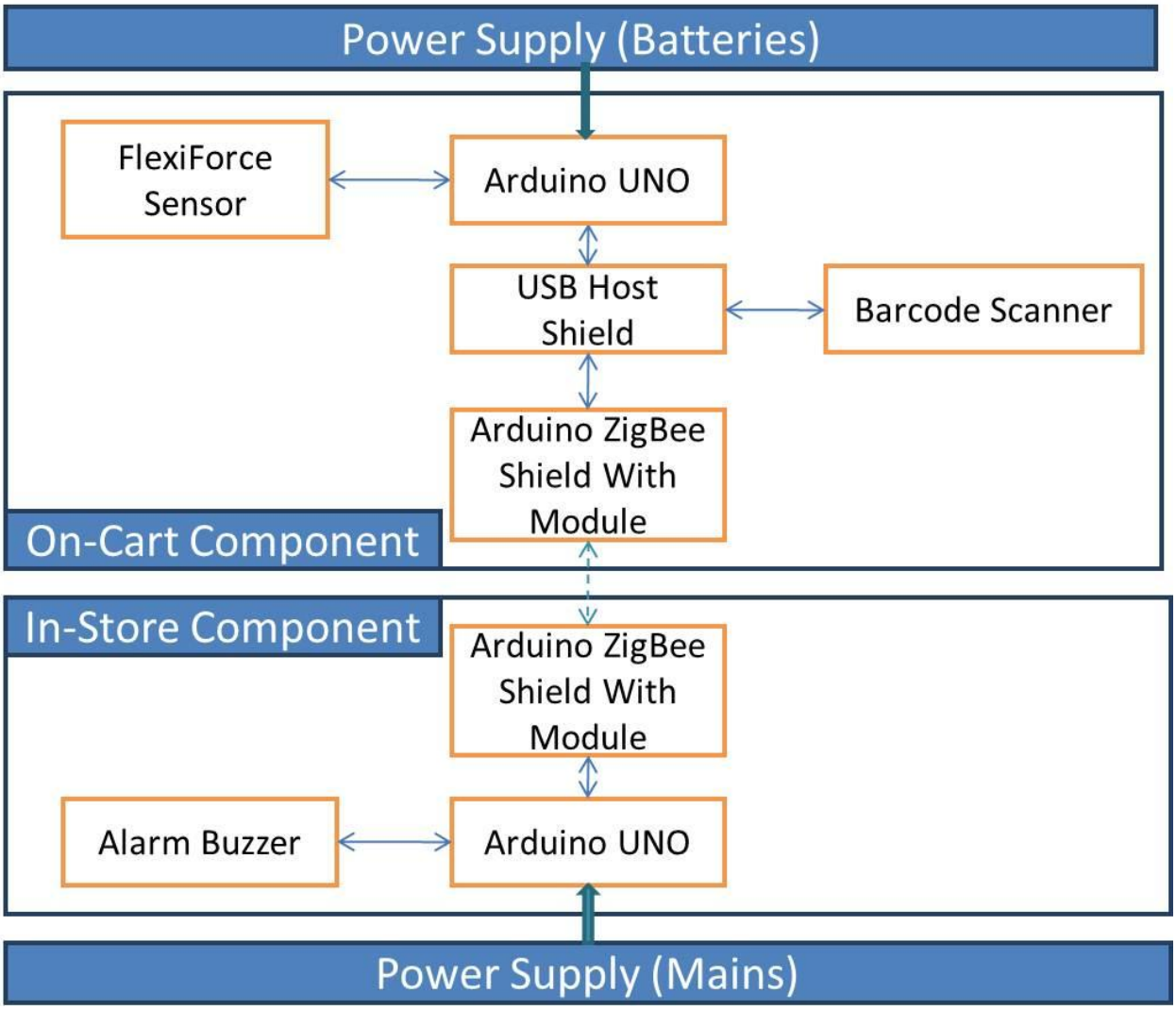

Figure 1: Block Diagram

We approached our design in a modular fashion. The various modules with their purpose and functionality are as follows:

1.3.1 Arduino Uno

Arduino Uno is the primary microcontroller of the whole system. It is based on the Atmel ATMega328 microprocessor. The Uno will be used for data processing, controlling the ZigBee shield, LCD shield, reading data from the alarm system and for reading data from the FlexiForce sensor.

1.3.2 Arduino ZigBee Shield with ZigBee Module

ZigBee shield provides an easy technique to interface the ZigBee module with the Uno. It allows easy access to well-known wireless libraries that come with the Arduino.

The ZigBee module provides the device with the wireless communication capabilities that will be used as part of the anti-theft mechanism and payment process. The module will constantly transmit a pre-defined pulse as long as no payment is processed. The pulse changes to hold different information once the payment is processed. Both these pulses are received by the same module present in-store.

The Uno connected to the in-store component decodes the data and transmits a pre-defined pulse in response. Based on the information encoded in the in-store transmission, the on-cart Uno sends information which would help trigger the alarm system.

1.3.3 Alarm System

When the cart reaches a certain distance to the in store component and if there is any discrepancy between the weight of the cart and the weight of the scanned items, the alarm system would trigger. There would be a sound something like a buzzer which would indicate a theft.

1.3.4 USB Host Shield

Since there is only one USB-B port available on the Uno, a USB Host Shield will be used in order to fulfill barcode scanning requirements for the component that is not connected directly to the Uno. The barcode scanner could be easily connected to the Arduino through this host shield.

1.3.5 FlexiForce Sensor

The sensor will be used to detect the presence of new items and also notify the shopper if certain items have not been scanned. Therefore, if the sensor shows a weight change of 10lbs. and the products scanned only total up to 8lbs., the shopper will be notified immediately about the discrepancy. Similarly, if items are removed, the sensor will show a negative change which the Uno will interpret and notify the shopper.

1.3.6 Barcode Scanner

The barcode scanner is the source of all the product data that will be available to the Uno. It is a normal scanner which transmits the data through USB port to the Uno. The Uno will in turn look up the code on the database and put out this data on the LCD screen.

1.3.7 TFT Touch Shield

The touch shield integrates with the Uno seamlessly. It comprises of a 2.8” TFT LCD touch screen and micro SD slot for additional data. The card will be used to hold the product database. The shield comes standard with LCD driver libraries that will work easily with the Uno. The data to the screen will come from the Uno over which the shield connects. The user will interact with the device through the touch screen. The user will be able to perform actions such as delete items, view the items already in the cart, etc.

2. Design

2.1 Design Procedure

This section intends to provide insight into how we chose the circuits and components that we used in the final device prototype.

2.1.1 Microprocessor

At the initial design stage, we considered the microcontrollers listed in Table 1.

Table 1: Microcontroller Trade-Offs

Microcontroller	Benefits	Drawbacks
Arduino ATMega328	Familiar environment, simple coding, minimal pin mapping, strong support ecosystem	Very few pins available for general use, not very robust
TI MSP430	Greater flexibility, multiple pins to allow same functions to occur simultaneously	TI Assembly language coding is difficult, inefficient support system
PIC	Easy to code	Very rigid in usability, suited to far simpler applications

We chose to go with the Arduino development environment because the support ecosystem for the Arduino is very strong, the IDE uses a C-programming based structure with which all us team members were very comfortable and Arduino development did not require soldering any microprocessor packages as we received a ready to use development board with defined pin mapping and easy to use libraries.

2.1.2 FlexiForce Sensor Circuitry

Figure 2 shows the FlexiForce sensor's resistance with respect to the force applied to it

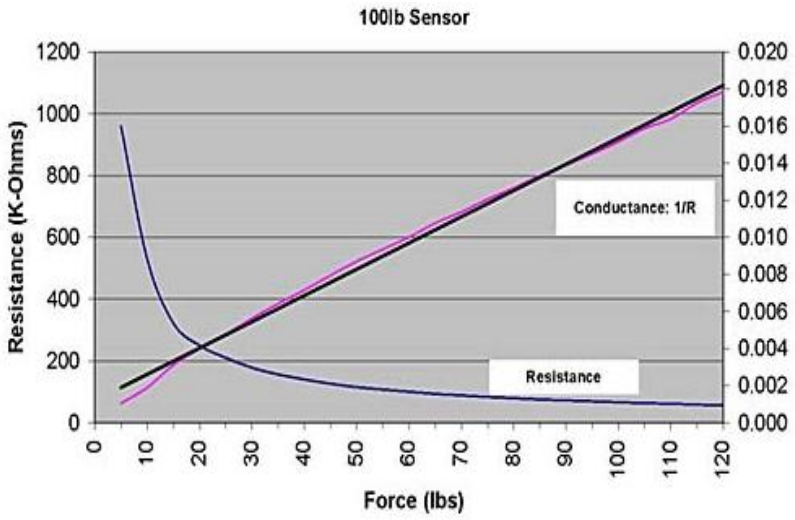

Figure 2: FlexiForce Resistance Variation

In order for the weight monitoring system to be accurate, it was crucial that the non-linearity in the sensor output be suppressed or even better, removed. The purpose of the sensor circuitry that we built was to eliminate this non-linearity.

We started the design by using the suggested MCP6004 single supply rail op-amp as our linearizing device. The benefits of using the MCP6004 were that it was cheaper, provided good noise immunity and a full 0 to +Vcc voltage swing at the output.

When we tested out our design using various feedback resistances, we noticed that the response time of the overall system was considerably degraded in comparison to how the sensor worked individually. Therefore, we concluded that the MCP6004 had poor mid to high frequency characteristics. To counter this issue, we replaced the MCP6004 with the LM324N op-amp which had all the MCP6004 benefits and also the added benefit of better high frequency characteristics.

2.1.3 Alarm Buzzer Circuitry

We started designing the alarm buzzer circuitry from scratch using in-lab components and a simple speaker system that would amplify the sound signals that it receives. Some way into the design, we faced multiple design challenges that included greater and more sophisticated filtering requirements and a far greater PCB footprint than we had budgeted for. Therefore, we moved to a more simplistic design which included a buzzer from the ECE parts shop.

The buzzer from the parts shop worked straight out of the box, but needed protection from high currents to prevent its diaphragm from burning out. It also needed additional capacitive delays to be included if we needed an intermittently beeping buzzer rather than a single non-stop tone. Therefore, based on the delays included in the code that controlled the buzzer, we placed the appropriate resistors and capacitors to provide a more robust RC circuit setup that would limit current and fulfill the capacitive requirements of the alarm buzzer.

2.1.4 ZigBee Network Setup

A typical star type ZigBee network looks as shown in Figure 3.

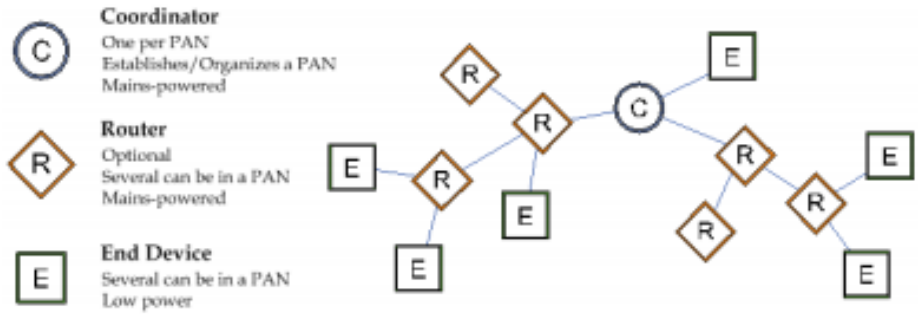

Figure 3: ZigBee Star Network

We started with a set up where we had only routers and no end devices or coordinators. The issue with this set up was that we could have point to point half duplex communication as long as all of the

parameters such as serial number, network ID and network channels had to be predefined. For a device that might end up in production, such a setup is hugely inefficient as each device would have to be paired with one other device leading to a multitude of devices and poor coordination.

To counter the problem, we set up the network with a coordinator and router. This setup works better since only a network ID parameter needs to be defined and the coordinator handles the rest of the network setup overheads.

2.1.5 Software Design for Barcode and LCD Control

The software requirements for the course were integrating the barcode scanner with the Arduino and retrieving product information from database stored in SD card for the barcode scanned. The corresponding product information is then displayed on the LCD screen. The barcode scanner was connected to the Arduino through a USB host shield which provided the serial communication link between the scanner and Arduino. The TFT LCD shield was chosen because it had a micro SD card reader through which information from the card could be read by the Arduino and could also be written back.

2.2 Design Details

This section serves to detail component choices for the circuit and the equations involved in calculation of component values.

2.2.1 FlexiForce Sensor Circuitry

Figure 4 shows the circuit schematic that we used for the FlexiForce sensor.

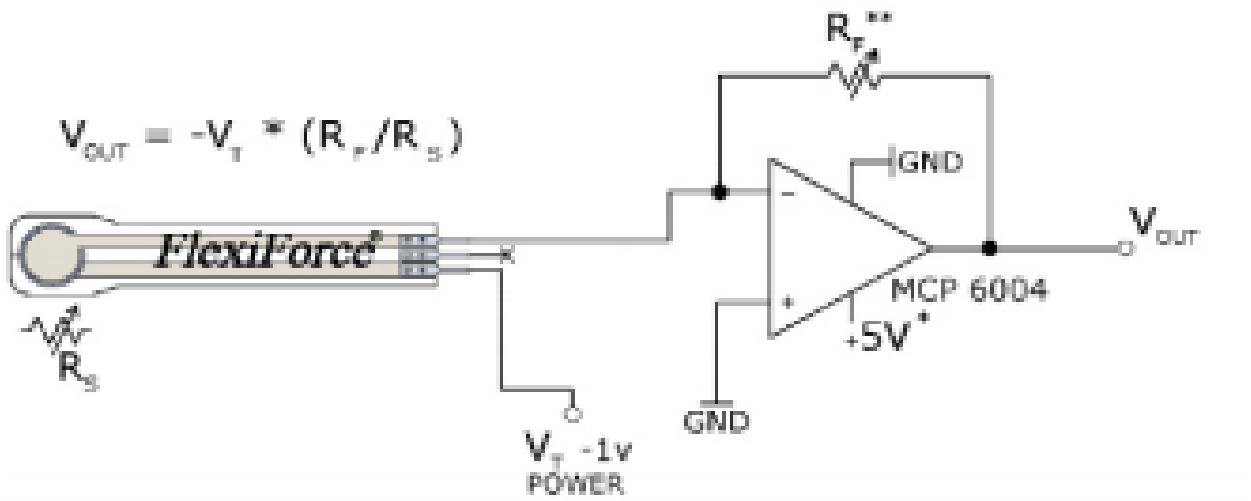

Figure 4: FlexiForce Circuit Schematic

As we mentioned earlier, the resistance output is highly non-linear and the LM324N op-amp serves as the linearizing device. Even as resistance is non-linear, the conductance of the sensor is highly linear. Using an inverting op-amp configuration, we use the following equation:

$$V_{OUT} = -V_T * (R_F/R_S) \quad (1)$$

From equation 1, we see that the op-amp configuration translates the non-linear resistance to the more linear conductance. The feedback resistance was chosen so that the amplifier would provide unity gain at the point where the sensor is fully loaded.

The specifications are listed below:

R_S (unloaded) = 5MΩ (2)

R_S (full stress) = 200kΩ (3)

We want Gain = 1 at full stress. Therefore, R_F= 200kΩ (4)

The resultant output of the circuit is as shown in Figure 5.

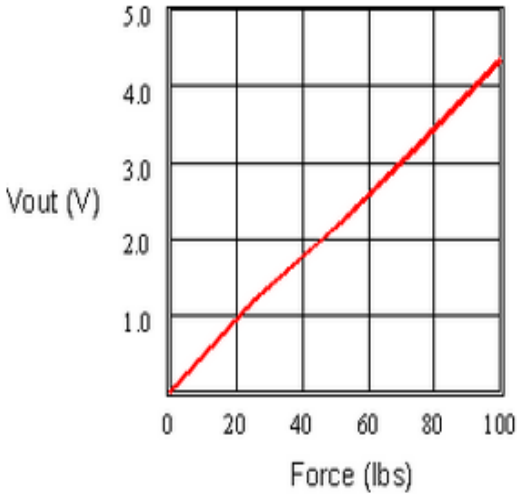

Figure 5: FlexiForce Sensor Circuit Output

2.2.2 Alarm Buzzer Circuitry

Our code states that the buzzer has a certain intermittent delay between buzzes. Based on the buzzer ratings, we chose to use an x ohm resistance.

The time constant of the RC circuit used to set up the delay. The RC circuit is set up as follows:

R_{BLEED} = 110Ω (5)

$1/R_{BLEED}C_{DELAY}$ = Delay (6)

From Eq. (6), we get C_{DELAY} = 1/ (R_{BLEED}*Delay) F (7)

The alarm buzzer schematic is shown in Figure 6.

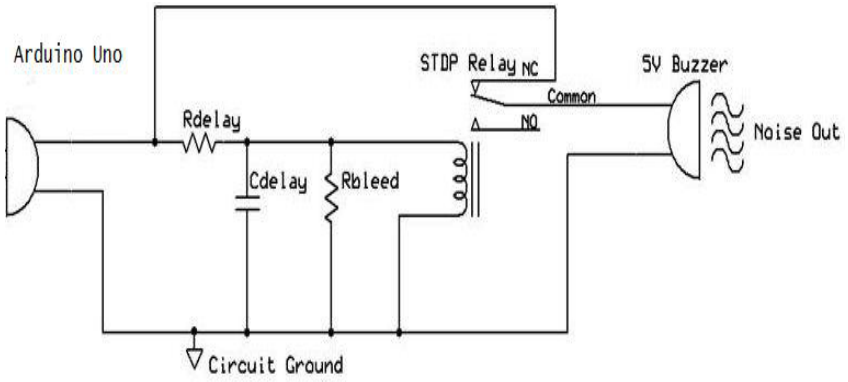

Figure 6: Alarm Buzzer Circuit Schematic

2.2.3 ZigBee Network Setup

Table 2 lists the criteria to set up a reliable point to point ZigBee network between two modules:

Table 2: ZigBee Coordination Criteria

Parameter	Coordinator	Router
PAN ID	Arbitrary	Same as coordinator
Destination High Byte	Router Serial Number High Byte	Coordinator Serial Number High Byte
Destination Low Byte	Router Serial Number Low Byte	Coordinator Serial Number Low Byte

The results of the configuration are shown in Figure 7.

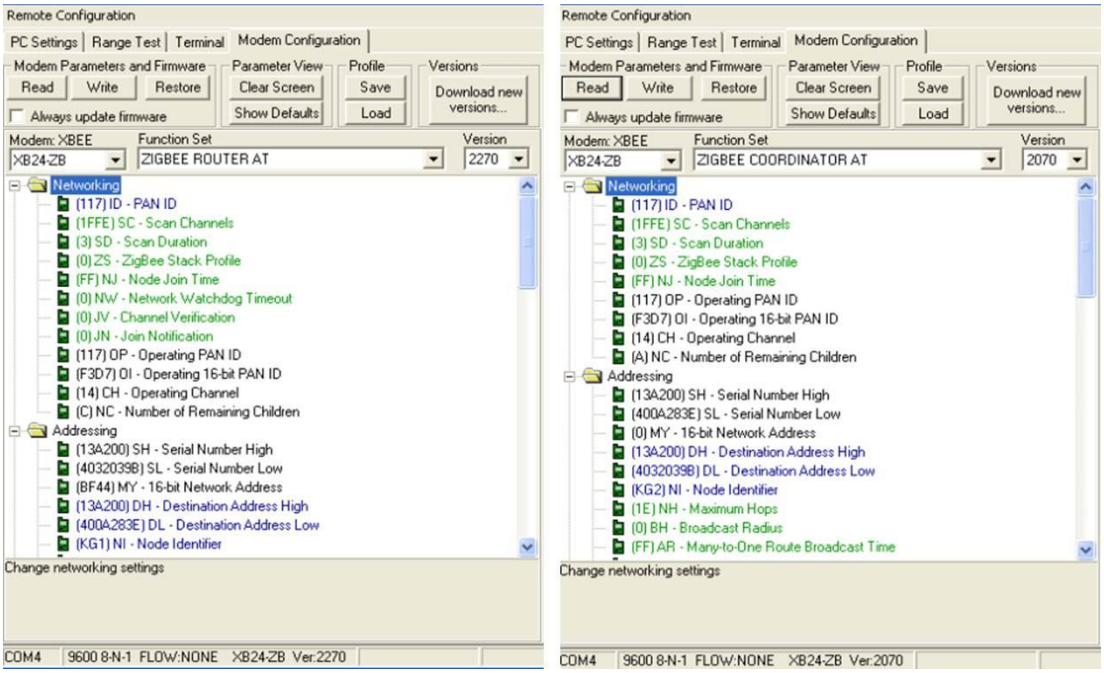

Figure 7: Configuration Results for ZigBee Modules

2.2.4 Software Design and Algorithms

The Arduino UNO receives data from the barcode scanner through a UART serial communication port. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the microcontroller a second UART reassembles the bits into complete bytes.

Algorithm for Barcode scanning:

The barcode scanning algorithm is similar to a PS/2 style keyboard. As soon as the barcode is scanned, the host shield reads every bit sequentially and the necessary ASCII conversion to each bit is performed. As the read is completed the content is displayed on the Serial monitor. Considerable time delay between each successive read and write is given so that any possibility of a bit being incorrectly read is eliminated. As this procedure is completed for a particular barcode, the barcode is stored in the Arduino's EEPROM to help identify the product with our database. Various libraries for USB support were given in the Arduino development center that helped to include the functionality of read and write by both the host shield and the Arduino.

Algorithm for Barcode compare and display

Initially the barcode information from the Arduino's EEPROM is retrieved and stored in an array. The database is stored in the SD card in a .CSV format. The database contains all product information such as name, price, barcode and weight. The file when read by an Arduino displays the information separated by commas.

The Read() function of Arduino reads all characters up to the first comma and stores them in an array. These characters are the barcode of the product. Then in a separate function the barcode read is compared to the barcode scanned and if a match is found, the successive characters after the comma which include the name and price corresponding to that barcode are displayed on the serial monitor.

Algorithm for Weight monitoring

The FlexiForce sensors on the cart give a particular voltage rating as and when any item is placed on them. This voltage is then calibrated to a corresponding weight and is stored to provide the total actual weight. This keeps updating as and when a new item is added. The weight of the item scanned is retrieved from the database and is maintained separately to provide the total scanned weight. This keeps updating as and when a new item is scanned.

The theft detection criteria is- Total actual weight > Total scanned weight. Thus upon the exit of the cart if this criteria is met then a signal from the Arduino on the cart is transmitted through a TRX component on the cart to an in-store component which triggers the buzzer to set off as a theft has been detected.

3. Design Verification

Since we designed our device in a modular manner, it was important that each module work individually before integration. We therefore tested each module individually and this section contains a summary of all the important test results.

3.1 Testing Procedure and Results

The following sections give a detailed explanation of the testing procedures and quantitative results for each of the modules that were tested.

3.1.1 Power Supply

The operating voltage for the Arduino is 5V. The required input voltage is 7-12V which is provided to the Arduino by two batteries of 3.7V each placed in series. Going below 5V will not shut off the Arduino instantly but could make the entire design unstable. To get a good estimate for the battery life of the operation the Arduino was connected to the scanner and TFT LCD shield. The current consumption of these three devices is listed in Table 3.

Table 3: Power consumption of various components

Device	Active Current consumed (mA)	Total power (W) (current * 5V)
Arduino	200	1
Barcode Scanner	100	.5
LCD screen	60	0.30

After adjusting the operating voltage to 5.2V, a set was data was recorded that showed the real operating voltage available to the Arduino as time progressed. The data is tabulated in Table 4.

Table 4: Operation voltage of Arduino

Voltage (V)	Time (Min)
5.2	0
5.2	16
5.2	34
5.1985	52
5.198	86
5.18	101
5.17	125
5.168	140
5.165	170
5.150	200
5.1492	230

Based on this data and assuming that adding more devices to the Arduino like XBee module will use up more power, the operating voltage would go down below 5V in approximately 9-10 hours. Based on the assumption that the cart will be used for 4 hours a day, the anticipated battery life came out to be 2-2.5 days. During the final stages of the testing when the entire design was integrated and the battery had to

support all the devices at once the design became very unstable after continuous use. Due to which we concluded that the battery could last for 5-6 hours rather than 9-10 hours when using all the components at once. Thus the battery would last for only around 1-1.25 days if the cart is still used for 4 hours a day. If the cart usage is reduced per day then the battery can survive for 2-3 days.

3.1.2 FlexiForce Sensor

Since the sensors formed a part of the anti-theft mechanism, significant testing was performed on the sensor circuitry to test response under various stresses. It was also important to ensure that all measured weights were within a predefined 7-10% tolerance built in to the software.

Table 5 summarizes the data collected for various weights using the sensor circuitry.

Table 5: Measurement Results for FlexiForce Sensor

Actual Weight (lbs.)	Measured Weight (lbs.)	Percentage Error (%)
10	9.3	7
20	19	5
30	27	10
40	36	10
50	48	4
60	59	1.67
100	100	0

3.1.3 ZigBee Module Performance

Wireless communication capabilities of our device can be leveraged in the future for far greater applications than the simplistic ones we have right now. However, it is imperative that the wireless network be robust and efficient. In order to ensure this, we require low packet error rates and efficient radiation patterns around the module antenna.

The packet error rates were tested using simple payload data consisting of "ABCD01234&%". We ran the same test 10 times and the packet error rates are summarized in Table 6.

Table 6: Error Rate Measurements for ZigBee

Test Number	Error Rate (%)
1	6.7
2	6.7
3	7.9
4	10
5	10
6	2.2
7	3
8	6
9	6
10	10

From the above data, the average error rate comes out to 6.85%.

The antenna was tested on the basis of its radiation patterns in comparison to an ideal dipole antenna. The results are shown in Figure 8.

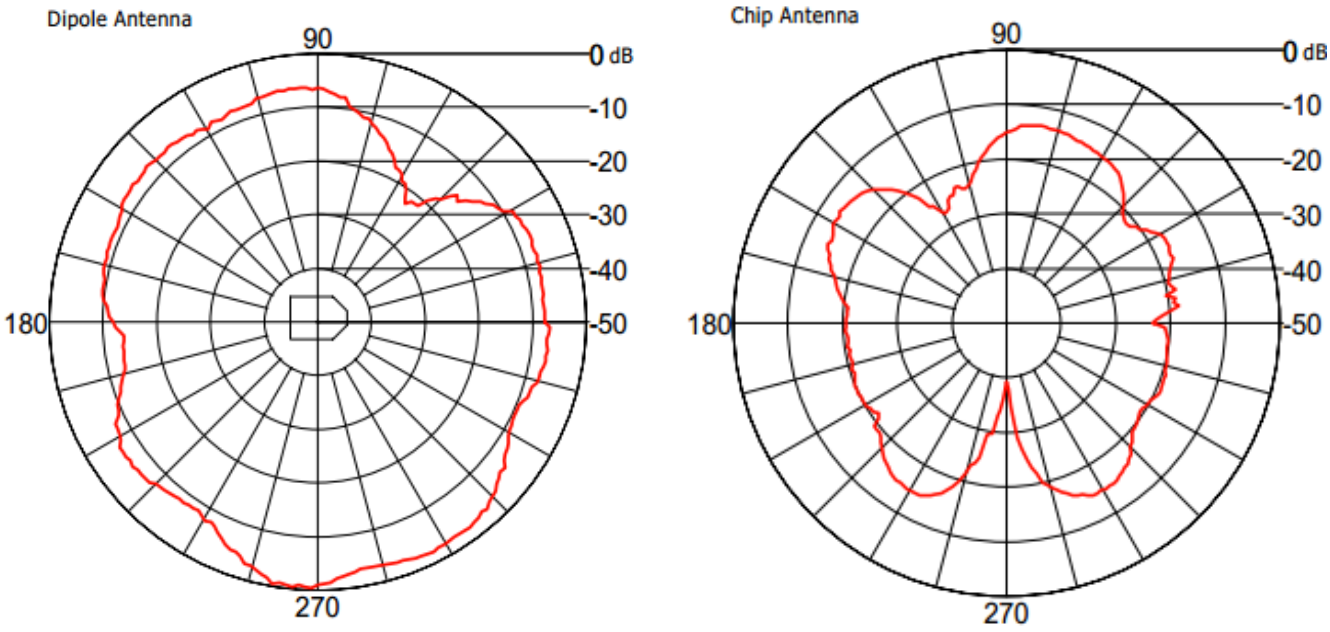

Figure 8: Antenna Radiation Pattern for XBee Antenna

3.1.4 Software Testing

These are the following results of the barcode scan and product identification.

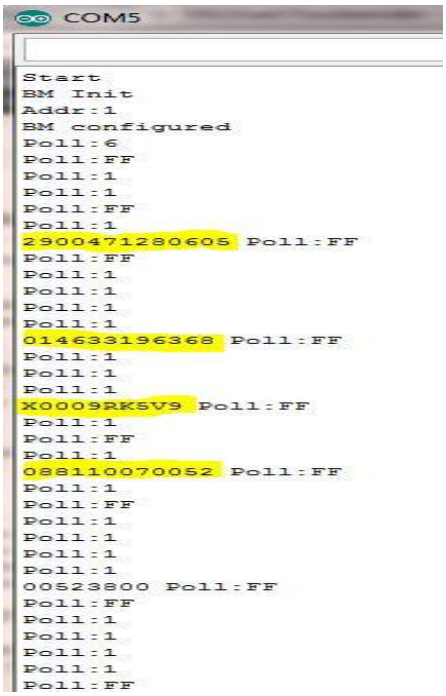

Figure 9: Barcode scanned

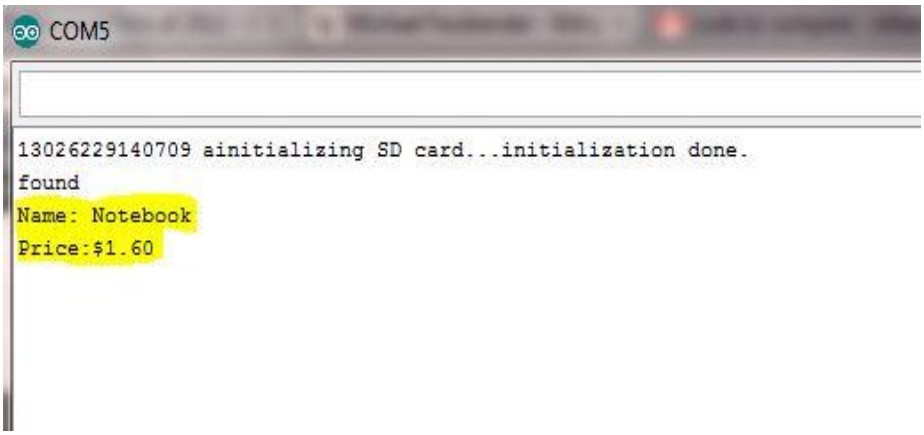

Figure 10: Product information

3.1.5 System Latency

System latency was measured using a stopwatch and the time came out to 3.78 seconds from scan to display. We are happy with these results since this is how much we anticipate the shopper to take between multiple product additions to the cart.

3.2 Testing Conclusions

Overall, our system performed admirably under various stress conditions. Our ZigBee modules showed higher error rates at a 270^0 orientation because of the weaker radiation field at this orientation. However, this is not a major issue since it is possible to align the various modules to be in the 270^0 orientation for a minimal amount of time.

The other major challenge faced by us was during integration. While all our modules worked satisfactorily individually, some of the modules could not be integrated together due to issues with the Arduino development board. We were unable to integrate the TFT LCD user interface with the rest of the device since the USB host controller for the barcode scanner and the LCD shield both used common pins on the Arduino resulting in a pin clash. As a result, we had to leave out the LCD shield from our final design prototype.

We also noticed that the FlexiForce sensor had issues with high frequency characteristics since it takes about 15 to 20 seconds to stabilize when new weights are placed. Therefore, a shopper must wait for this time period before placing new items on the cart. We believe that this is not a major issue as items are added to a shopping cart at far greater time intervals than 15 seconds.

In conclusion, we believe these tests helped us ensure that all the critical components of the design worked to specifications.

4. Costs and Benefits

4.1 Parts

Table 7 lists the cost of the parts used in our device.

Table 7: Cost of parts

Description	Manufacturer	Retail Price ($)	Wholesale Price ($)*	Quantity	Retail Total ($)
Arduino Uno	Adafruit	29.95	25.00	2	59.9
TFT touch Shield	Adafruit	59.00	50.00	1	59
XBee Shield with radio module	Liquid Ware	59.37	48.00	2	118.74
Barcode scanner	Amazon	29.97	20.00	1	29.97
Go between shield	sparkfun	13.95	7.00	1	13.95
USB Adapter	Arduino	14.98	10.00	1	14.98
Alarm Buzzer	sparkfun	2.00	1.50	1	2.00
Battery	Tenergy	41.99	30.00	2	83.98
Flexi Force Sensor (Pack of 4)	Tekscan	65.00	53.00	1	65

*Wholesale prices are product prices if purchased in quantities greater than 1000.

4.2 Labor

Table 8 lists the labor cost involved in our project

Table 8: Labor costs

Employee Name	Hourly Rate	Estimated No. of Hours	Total = Rate*Hours*2.5
Rohan Singh	$40	160	$16,000
Kartik Sanghi	$40	160	$16,000
Nikhil Raman	$40	160	$16,000

Grand total = Parts needed + Labor cost

= $447.52 + $48,000

= $48,447.52

Production total = Parts needed + Labor cost

= $347.50 + $48,000

= $48,347.50

Production costs are lower than the prototyping costs as with wholesale prices, Wal-Mart can save a minimum of $100 on every single Smart Cart device.

4.3 Wal-Mart Budget per Store

When preparing an implementation budget for Wal-Mart, it is critical that the following issues be considered:

Implementation consultants: The cost incurred by Wal-Mart in paying their engineering team who will coordinate the transition

Materials: The actual device cost

Manpower Usage: The actual labor (number of people) involved in modifying the carts

Cost towards disruption (if any): Many carts may be out of commission when they are being modified

Any utilities used: Electricity and water used

Consultants:

The engineering department at Wal-Mart and any consultants that are used are the largest component of the budget. Using an estimated number of 15 such employees for this project and an hourly wage of $40 per employee, the total cost (using a 3 day approximation for the implementation timeframe) comes out to $14,400.00

Material Cost:

The final value for materials comes out to $467.18

Manpower usage:

As explained above, this section includes the wages of the employees physically placing this device on the carts. It is not necessary for these employees to be technically capable. Estimating the hourly wage for such employees to be around $15 (considerable rounding up) and the time taken to physically place a device on the cart to be a maximum of 1 hour, the labor cost per cart would be $15. Approximating that 250 carts will be equipped, the total cost is $3750.

Cost towards disruption:

When the carts are being modified, they cannot be used for shopping. Therefore, it is essential that the modification process be taken up during the relatively lean hours of operation. Estimating (based on the fact that hundreds of carts are available at any given point at a single Wal-Mart store) that around 5 carts that would normally be used for shopping are actually being modified and using an estimated income per cart to Wal-Mart of $150, the cost of disruption per day would be $750.

The whole process must not take more than 3 days to be successfully implemented on all carts and this yields total cost of disruption of $2250.

Utilities used:

The cost of utilities used would be negligible compared to the typical total daily utilities bill of a Wal-Mart store.

Total Cost **= Consultants + Materials + Labor + Cost toward disruption**

= $14,400 + $447.52 + $3,750 + $2,250

= $20,847.52

4.4 Time Saving Estimate per Shopper

- Checkout Procedure (**Without Smart Cart**)

Assume that there are 5 people ahead of the person whose shopping time is being computed and every person holds 20 items in his cart.

Time required to scan and bag a product = 4-5 seconds

Total time for 20 products = 80-100 seconds

Total time for all 5 people = 400 – 500 seconds (6.66 min – 8.33 min)

Time required for payment by one person = 1 minute

Total Time for Payment by all = 5 minutes

Total wait time = 11.66 min – 13.33 min

Approximately the total wait time is 12-14 minutes, so on an average the person has to wait **13** minutes.

If miscellaneous time of around 1 minute is included, then the total time is **14** minutes.

- Checkout Procedure (**With Smart Cart**)

With the smart cart the total payment is wirelessly sent to a central payment station where the person only has to make the final payment. This reduces any time required to scan the items during checkout. With the assumption that 5 people with 20 products are already waiting at the payment station, the total shopping time for any person is computed in the following way:

Time required for payment by one person = 1 minute

Total Time for Payment by all = 5 minutes

Time required to bag a product – 2 seconds

Time required to bag 20 products – 40 seconds

Time required for bagging by all – 200 seconds or 3.33 minutes

Total wait time = 8.33 minutes

Approximately, the total wait time is **8** minutes. If miscellaneous time of 1 minute is included then the total time is **9** minutes.

Thus the user saves 5 minutes by using the Smart Cart device.

5. Conclusion

5.1 Accomplishments

The following is a list of objectives successfully satisfied by the device:

- Accurate and real time weight monitoring
- Accurate barcode reading and database comparison
- Reliable and robust wireless communication
- Efficient anti-theft mechanism

5.2 Challenges

The following is a list of challenges faced during the building of the device:

- Lack of integration of SD card from the LCD TFT shield and USB Host controller shield due to the use of same pins for the SPI interface
- Sensor challenges –Very small sensing area made it difficult to concentrate weights of bigger objects correctly
- Non-linear output requiring linearization

5.3 Ethical considerations

Most of the IEEE codes of Ethics have been religiously followed. A few Codes of Ethics are directly applicable to the projects which are:

- To be honest and realistic in stating claims or estimates based on available data
- To improve the understanding of technology; it's appropriate application, and potential consequences
- To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others

These pointers directly relate to this project and it is important to credit Wal-Mart for this idea and rest of the ethical codes are not directly related to this project but have been taken care of.

5.4 Future work

Given the modular and expandable design of the device, future improvements from both hardware and software points of view are possible. Here, we list just a few developments that we believe can make our device more attractive:

- More responsive User Interface – An LCD screen that's more than just a display
- Smartphone application – Given the growth of the Smartphone industry, designing a Smartphone application that will make using the barcode scanner and payment terminal redundant will make the device more cost effective and also more widespread

6. References

[1] IEEE Code of Ethics

http://www.ieee.org/about/corporate/governance/p7-8.html

[2] TFTLCD and SD card

http://www.ladyada.net/products/tfttouchshield/

[3] XBEE MODULE

http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module#specs

[4] BARCODE SCANNER

http://www.circuitsathome.com/mcu/connecting-barcode-scanner-arduino-usb-host-shield

[5] FLEXIFORCE SENSOR

http://www.tekscan.com/flexible-force-sensors#specifications

Appendix A Requirement and Verification Table

Requirement	Verification
1. Input voltage provided to each component is in the range: a. 7-12V provided to Arduino b. 1-1.5V provided to sensor c. Constant 5V provided to Barcode scanner	1. A potentiometer will be used to divide the voltage to desired values for the components placed in parallel a. Use DMM to check input to potentiometer is 7-9V b. Use DMM to check input to Arduino is 7-9V c. Use DMM to measure output of the 2 potentiometer branches which should be 1-1.5V for sensor and 4-5V for Arduino
2. Sensor senses small variations in weight from 0 to 1000 lbs. a. 1-1.5V supply provided to sensor b. Constant 5V provided as supply rails to Op-Amp in drive circuit c. Voltage changes with respect to change in weight	2. Connect feedback resistance to the drive circuit to amplify output a. Increase feedback resistance in order to achieve voltage change of 5mV b. Increase weight on sensor to verify 5mV steps c. Obtain trade-off between resolution and resistance value to check for performance degradation
3. ZigBee module transmits enough power such that received levels fall within sensitivity for various receiver distances. a. Constant 0dBm power output from transmitter b. Modulation error rates within IEEE standards specifications	3. Connect ZigBee module antenna to spectrum analyzer a. Check that antenna has very low S11 value at 2.4GHz to ensure successful radiation b. Transmit using one module and receive with other module c. Ensure transmit power is 0dBm using spectrum analyzer d. Measure received power spectrum on a spectrum analyzer to verify prior calculation estimates e. Check this spectrum for various receiver distances f. Compare received and transmitted spectrum to check for possible degradation
4. Verify software programs are working correctly a. Weight is correctly inferred from voltage change on the sensor b. Barcode data is correctly read into USB	4. The software that will be written for the device will define the degree of success achieved. a. Check the register holding the weight information using the Arduino debugger

c. Barcode data is correctly correlated to product database d. ZigBee module data correctly demodulated	b. Manually check the register holding the barcode data to verify that it translates to a correct value c. Manually check the database for the product matching the data and confirm if the software is pulling the same data d. Use ZigBee emulator to transmit a known pulse (eg. All 1's) and verify using the same emulator that the same pulse is received.
5. Verify latency rates of the system a. Time taken for barcode data to be read and deciphered b. Time taken to run the code to correlate the data to a product c. Time taken to transmit data to LCD	5. The success of the device depends on how quickly the data can be processed and shown to the user a. Manual timing will be used to estimate the system latency (eg. Using a stopwatch)

Appendix B Arduino Code

B1 Barcode Scanning

```
#include <LiquidCrystal.h>
#include <avr/pgmspace.h>

#include <avrpins.h>
#include <max3421e.h>
#include <usbhost.h>
#include <usb_ch9.h>
#include <Usb.h>
#include <usbhub.h>
#include <avr/pgmspace.h>
#include <address.h>
#include <hidboot.h>

#include <printhex.h>
#include <message.h>
#include <hexdump.h>
#include <parsetools.h>
#include <EEPROM.h>

#define DISPLAY_WIDTH 16

// initialize the LCD library with the numbers of the interface pins
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

USB     Usb;
int i;
//USBHub     Hub(&Usb);
HIDBoot<HID_PROTOCOL_KEYBOARD>    Keyboard(&Usb);

class KbdRptParser : public KeyboardReportParser
{

protected:
virtual void OnKeyDown (uint8_t mod, uint8_t key);
virtual void OnKeyPressed(uint8_t key);
};

void KbdRptParser::OnKeyDown(uint8_t mod, uint8_t key)
{
   uint8_t c = OemToAscii(mod, key);

   if (c)
      OnKeyPressed(c);
}

/* what to do when symbol arrives */
void KbdRptParser::OnKeyPressed(uint8_t key)
{
static uint32_t next_time = 0;      //watchdog
static uint8_t current_cursor = 0;  //tracks current cursor position

   if( millis() > next_time ) {
```

```
      lcd.clear();
      current_cursor = 0;
      delay( 5 );  //LCD-specific
      lcd.setCursor( 0,0 );
    }//if( millis() > next_time ...

    next_time = millis() + 200;  //reset watchdog

    if( current_cursor++ == ( DISPLAY_WIDTH + 1 )) {  //switch to second line if cursor outside the screen
      lcd.setCursor( 0,1 );
    }


    EEPROM.write(i, key);
    i++;
    Serial.write( key );
    lcd.print( key );
};

KbdRptParser Prs;

void setup()
{

   i = 0;
    Serial.begin( 115200 );
    Serial.println("Start");

    if (Usb.Init() == -1) {
        Serial.println("OSC did not start.");
    }

    delay( 200 );

    Keyboard.SetReportParser(0, (HIDReportParser*)&Prs);
    // set up the LCD's number of columns and rows:
    lcd.begin(DISPLAY_WIDTH, 2);
    lcd.clear();
    lcd.noAutoscroll();
    lcd.print("Ready");
    delay( 200 );
}

void loop()
{
  Usb.Task();

}
```

B2 Barcode Compare

```
#include <SD.h>
    #include <SPI.h>
    #include "TFTLCD.h"
    #include <EEPROM.h>
    #if not defined USE_ADAFRUIT_SHIELD_PINOUT
     #error "For use with the shield, make sure to #define USE_ADAFRUiT_SHiELD_PiNOUT in the TFTLCD.h
library file"
    #endif

    // These are the pins as connected in the shield
    #define LCD_CS A3    // Chip Select goes to Analog 3
    #define LCD_CD A2    // Command/Data goes to Analog 2
    #define LCD_WR A1    // LCD Write goes to Analog 1
    #define LCD_RD A0    // LCD Read goes to Analog 0

    // The chip select pin for the SD card on the shield
    #define SD_CS 5


    //#define SD_CS 5
    TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, 0);
    File myFile;
    int8_t saved_spimode;

    void disableSPi(void) {
      saved_spimode = SPCR;
      SPCR = 0;
    }

    void enableSPi(void) {
      SPCR = saved_spimode;
    }


    boolean check_bar(byte scanned_bar[], byte file_bar[], int length)
     {
              int f=0;
               //Serial.println("here");
               while(f<length-1)
               {
                 if(scanned_bar[f]!=file_bar[f+1])

                  {
                   return false;
                  }
                 else {
                   f++;
                  //Serial.print(f);
                 }
               }
          return true;

     }
    void setup()
```

```
{
  Serial.begin(9600);
  tft.reset();
  uint16_t identifier = tft.readRegister(0x0);

  byte value[13];
  int a=0;
  for(int address=0; address<13; address++)
  {
    value[a] = EEPROM.read(address);
    a++;
  }
    Serial.print(a);
  for(int b=0; b<=a; b++)
  {
    Serial.write(value[b]);
  }

  // advance to the next address of the EEPROM
  // there are only 512 bytes of EEPROM, from 0 to 511, so if we're
  // on address 512, wrap around to address 0

  Serial.print("initializing SD card...");
  // On the Ethernet Shield, CS is pin 4. it's set as an output by default.
  // Note that even if it's not used as the CS pin, the hardware SS pin
  // (10 on most Arduino boards, 53 on the Mega) must be left as an output
  // or the SD library functions will not work.
   int i=0,j=0;
   byte barcode[14];
   byte name[40];
   byte price[4];
  //byte barcode[2][12];

  int flag;
  int count =0;

  if (!SD.begin(SD_CS)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  // open the file for reading:
  myFile = SD.open("Book5.csv");
  if (myFile) {
    //Serial.println("Book5.csv:");
 int flag = 0;
 int barcode_index=0, name_index=0, price_index=0,name_length=0, price_length=0;
 char c;
    // read from the file until there's nothing else in it:
while (myFile.available()) {
  c = myFile.read();
  if (c== ',' && count == 0){
    boolean code_check = false;
    code_check = check_bar(value, barcode, a);
    if(code_check == true) {
```

```
      Serial.write("found");
      flag = 1;
      Serial.print("\nName: ");
    }
    barcode_index = 0;
    count ++;
    continue;
  }
  if (c == ','){
    if (flag == 1){
      Serial.print("\nPrice:");
    }
    continue;
  }
  if (c=='\r' || c=='\n'){
    count = 0;
    if (flag == 1){
      break;
    }
    continue;
  }
  if (count == 0){
    barcode[barcode_index] = c;
    barcode_index++;
  }
  else {
    if (flag == 1){
      Serial.write(c);
    }
  }
}
myFile.close();
} else {
   // if the file didn't open, print an error:
    Serial.println("error openng Book.csv");
  }

}

void loop()
{
 // nothing happens after setup
}
```

B3 ZigBee and Buzzer Control

```
const int ledPin = 13; // the pin that the LED is attached to
int incomingByte;      // a variable to read incoming serial data into

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  pinMode(7, OUTPUT); // set a pin for buzzer output
  int incomingByte;
}

void loop() {
  // see if there's incoming serial data:
  if (Serial.available() > 0) {
 //   delay (2000);
    // read the oldest byte in the serial buffer:
    // incomingByte = Serial.read();
    incomingByte = Serial.read();
    Serial.write(incomingByte);
    // if it's a capital H (ASCII 72), turn on the LED:
    if (incomingByte == 'H') {
      digitalWrite(ledPin, LOW);
//      Serial.print(incomingByte);
      buzz(7, 2500, 500); // buzz the buzzer on pin 4 at 2500Hz for 1000 milliseconds
 //     delay(500); // wait a bit between buzzes
//      Serial.println(incomingByte);
    }
    // if it's an L (ASCII 76) turn off the LED:
    else {
      digitalWrite(ledPin, HIGH);
//      Serial.write(incomingByte);
    }
   // Serial.print(incomingByte);
  }
//  Serial.flush();
}

void buzz(int targetPin, long frequency, long length) {
  long delayValue = 1000000/frequency/2; // calculate the delay value between transitions
  //// 1 second's worth of microseconds, divided by the frequency, then split in half since
  //// there are two phases to each cycle
  long numCycles = frequency * length/ 1000; // calculate the number of cycles for proper timing
  //// multiply frequency, which is really cycles per second, by the number of seconds to
  //// get the total number of cycles to produce
 for (long i=0; i < numCycles; i++){ // for the calculated length of time...
    digitalWrite(targetPin,HIGH); // write the buzzer pin high to push out the diaphram
    delayMicroseconds(delayValue); // wait for the calculated delay value
    digitalWrite(targetPin,LOW); // write the buzzer pin low to pull back the diaphram
    delayMicroseconds(delayValue); // wait againf or the calculated delay value
  }
}
```