THE B.O.S.S.

Buyer Operated Shopping System

December 3, 2004

Prepared for:
Dr. Kathleen Kramer & Dr. Chuck Pateros
EEE 191

Prepared by:
Dustin Mendes
Erik Loftis
Bill Leslie
Brian Momeyer

Table of Contents

Lists of Graphics

Graphic **Page**

Introduction

In recent times there has been a growing demand for efficiency and time saving techniques in every aspect of society. From faster internet to at home grocery delivery, consumers have grown accustomed to rapid delivery of goods and better use of their time. On a recent trip to a grocery store it became evident that this is an industry that is behind the times of recent technological advancements. While some grocery stores have recently added self-check lines to their stores, there still is a need for a more rapid way to check out. By creating a device that will allow the consumer to ring up grocery items while they shop we will be able to alleviate the current problems facing grocery store checkout. Such a system would not only cut down on the consumer's time spent in the store but would also allow the grocery stores to cut back on the amount of cashiers needed.

Problem Statement and Objective

Grocery stores need a way to decrease the bottleneck effect that often occurs at checkout lines. The Buyer Operated Shopping System will provide grocery stores with a solution to this problem by allowing customers to scan and total their items as they shop. The Buyer Operated Shopping System is a shopping cart contained apparatus (Figure 1**)** that consists of four main components: a barcode scanner, a scale, a touchpad mouse, and a central processing unit with display.

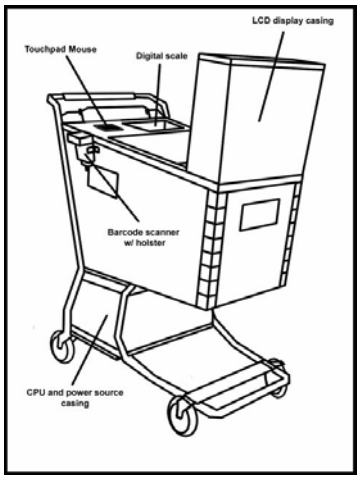


Figure 1 - Prototype Illustration

Actions that are normally done at the checkout line are done by the consumer as they shop. The unit displays real time item subtotals on an LCD screen. The unit houses an onboard barcode reader to scan items and produce tags. A scale is integrated into the system and is used to weigh and price produce and similar items. Finally, an output is provided from the unit to a designated cashier for customer payments of cash, check or other non-store account related options. The system alleviates the need for long checkout lines while also presenting a more efficient and personalized shopping experience for the customer. Along with the added convenience for the shopper the B.O.S.S. also helps reduce costs for the grocery store. By implementing the B.O.S.S. grocery stores will be able to cut back on the amount of cashiers that are needed. This will lower operating costs and will in turn pay for the systems.

Project Description

The Buyer Operated Shopping System is a shopping cart contained apparatus that consists of four main components: a barcode scanner, a scale, a touchpad mouse, and a central processing unit with display. Actions that are normally done at the checkout line are done by a consumer as they shop. The user is able to add and remove items from their cart while simultaneously scanning, pricing, and totaling their purchases. A simplified system schematic can be found below (Figure 2).

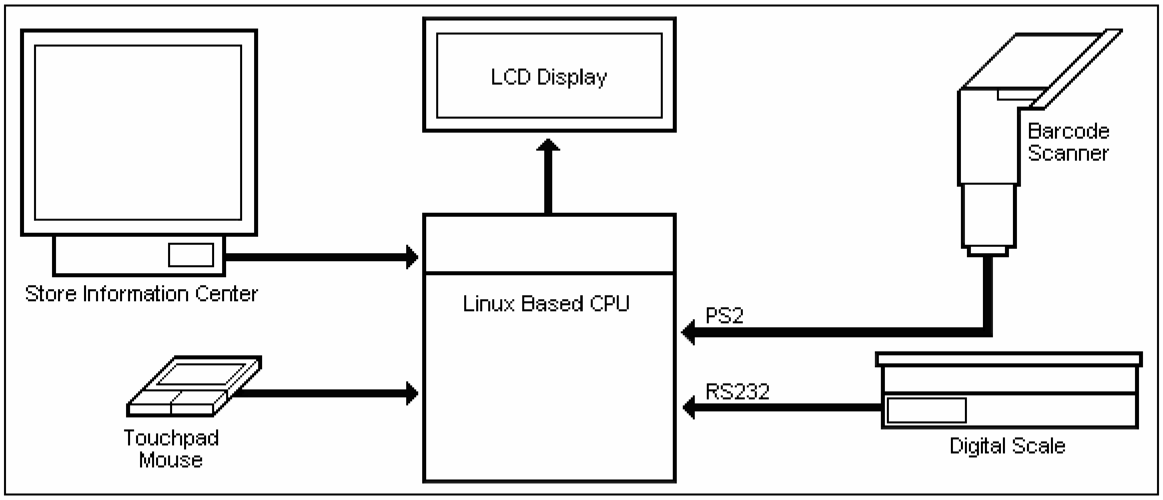


Figure 2 – System Simplified Schematic

As seen above in Figure 2, the main system will control all the functions of the B.O.S.S. The central controlling system is a hard drive based CPU that is housed in a Mini ITX box. This will allow the B.O.S.S. to have adequate memory storage to allow future upgrades and system overhauls. While the system is running the CPU will be receiving input from the digital scale, barcode reader, and the touchpad mouse. The outputs from the CPU are to the LCD to allow the customer to interact with the system and also to the stores database. A much more in depth description of the systems functions can be found later in the report.

The B.O.S.S. will be placed in docking stations where they will be available for customers to checkout. When a customer enters a store they will check out a system which will be assigned to them with their credit card as a deposit. This is to ensure that the customer will return the cart and if not they will be charged for it. Following this the customer will then be allowed to take a B.O.S.S. unit and begin shopping. It is also anticipated that the deposit of a credit card will help reduce shoppers from stealing items by not scanning them during checkout.

As the customer begins to shop they will use either the barcode reader to add items to their cart or the scale to add produce items. As items are scanned/weighed they are added to the cart's subtotal which is dynamically updated on the screen. At anytime while shopping the user may remove items from their cart by clicking the "remove" button on the screen and scanning the item they wish to discard. If the customer wishes to remove a produce item they will have to select remove item and then scan the produce barcode tag.

One of the key features of the B.O.S.S. is the integration of a digital scale. Users will be able to take pay per weight items such as produce and weigh them on the cart's scale in order to select the desired amount of produce. This is done by first scanning an item's barcode then weighing the item or items on the scale. After adding the desired amount of produce to the scale the user will click on the acquire button on the screen that will then take the weight output from the scale. The screen will then show the item and its grand total cost in the subtotal area on the screen of the B.O.S.S.

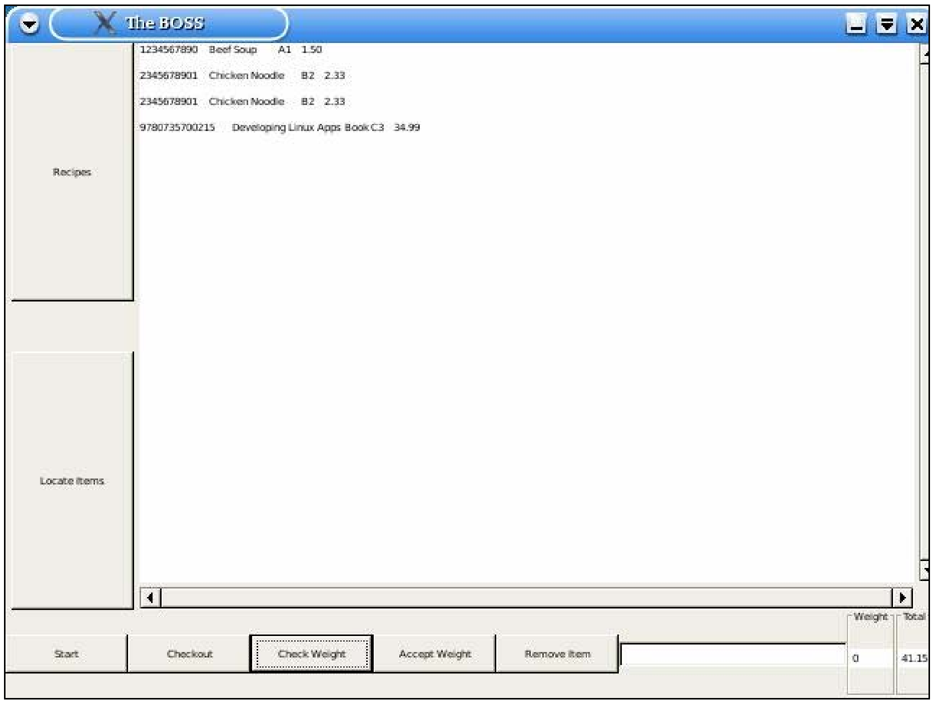
Once the user has added all their items to the cart they may then checkout by selecting a key labeled "checkout." Once the customer has selected the checkout option they will need to proceed to the B.O.S.S. only checkout line. Here they will be charged for the items they have selected and will also return the cart. Once the cart has been returned the customer will no longer be responsible for it and they may have their deposit back.

Project Components

Graphical User Interface

To allow the customer to interact with the functions of the B.O.S.S., a user friendly graphical user interface (GUI) was needed. In order to generate a GUI that would allow the user to successfully navigate the functions of the B.O.S.S. we decided to use the Glade/GTK+ program to develop it. This program allowed us to select how the GUI would look and also how it would function on the front end.

To incorporate the totality of the B.O.S.S.' functions we decided on the GUI layout that is found in Figure 3. As you can see from the Figure we have selected a layout that allows the user to select from seven different buttons. These buttons control the functions of the stored recipes, location of items, starting the system, checking out, checking the produce weight, accepting the produce weight, and removing a previously scanned item.



```
1234567890   Beef Soup      A1   1.50
2345678901   Chicken Noodle    B2   2.33
2345678901   Chicken Noodle    B2   2.33
9780735700215    Developing Linux Apps Book C3   34.99
```

Weight: 0
Total: 41.15

Figure 3 – Graphical User Interface Screen Capture

The system is designed so when a customer wishes to begin shopping they will need to click on the start button with the accompanied touchpad mouse. This will then allow the user to begin their shopping experience.

The GUI has been set up in such a manner that when the customer wishes to scan an item they need to place the cursor in the text entry line. This is because the barcode reader is recognized by the system as a keyboard input and thus the system needs to be in text entry mode to recognize its input. The customer will place the cursor on the text line by clicking on the mouse when the cursor is over the text line. Through a brief tutorial the customer will learn the system and this should not hinder their ability to use the B.O.S.S.

The next unique function of the B.O.S.S. is the ability to shop for produce with its onboard scale. This will allow the user to buy any amount of produce that they desire. When the customer wishes to purchase produce they will follow a similar procedure to that of selecting regular items. They will first need to scan a barcode for the item that they would like to buy which will be found on the bottom of the produce stand (Figure 4).

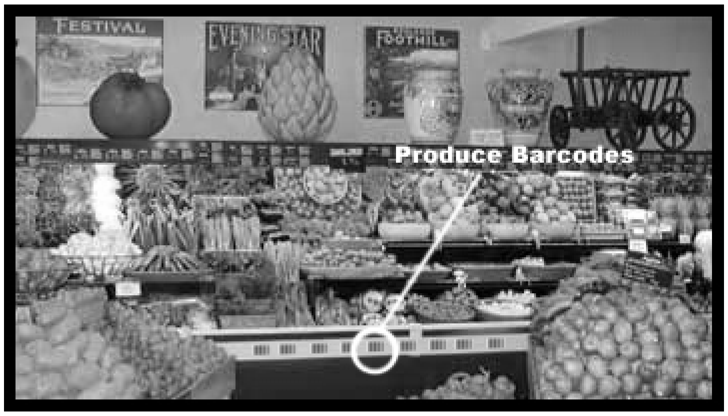
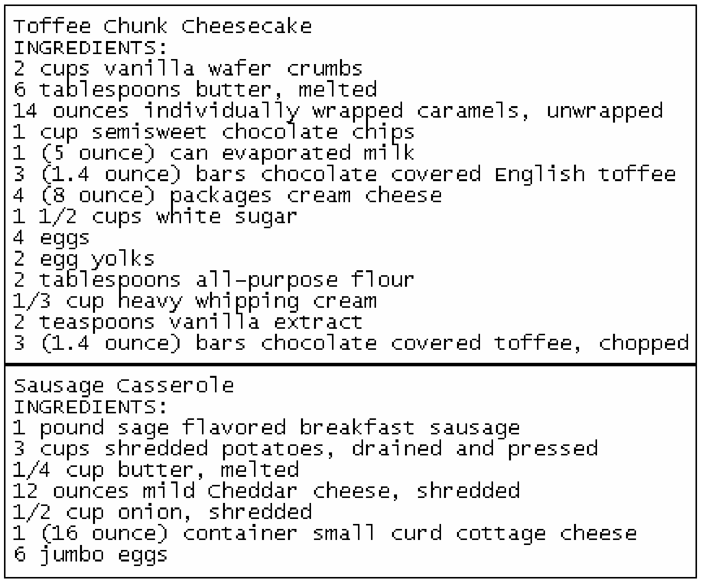


Figure 4 – Barcodes in the Produce Aisle

This will search the database for the selected produce price per weight. The customer will then place the desired amount of produce onto the onboard digital scale and click on check weight. This will then take the output from the digital scale and output the current weight, at time of clicking button, to the weight section of the GUI. If the customer is satisfied with the amount of produce that was weighed they will need to click on the accept weight button. This will then add the produce and its weight to the running list along with its price. However, if the customer is unsatisfied with the amount of produce that was on the scale at the time of checking the weight they can either add more or take away the desired amount and then recheck the weight. Once they are satisfied with the amount of produce they can select to accept the weight. While realizing that it would be very simple for the customer to weigh less than they place in their cart, we are relying on the honor system to keep people honest. Since this is not available to anyone, but rather people who leave a deposit to check out the B.O.S.S. we feel that this will help keep people honest. The amount of money that the grocery store will save on cashiers will more than offset the minimal amount of money lost to theft.

In the event that the customer scans/accepts an item that they no longer wish to purchase, they will be able to remove the item(s) from their shopping list. This will be done by first selecting the remove item button on the GUI and then scanning the item that they wish to remove. This will not only remove the item from the shopping list but also remove its cost from the running total. We realize that this will be slightly more difficult when the customer wishes to remove produce because they will have to go back to the stand to scan the barcode. However, this will keep people from just placing fresh produce anywhere in the store when they decide that they no longer want it.

The next unique function of the B.O.S.S. is the recipe lookup button. By clicking on the recipe button the user will pull up a list of preloaded recipes which show the ingredients and amount of each needed (Figure 5).



```
Toffee Chunk Cheesecake
INGREDIENTS:
2 cups vanilla wafer crumbs
6 tablespoons butter, melted
14 ounces individually wrapped caramels, unwrapped
1 cup semisweet chocolate chips
1 (5 ounce) can evaporated milk
3 (1.4 ounce) bars chocolate covered English toffee
4 (8 ounce) packages cream cheese
1 1/2 cups white sugar
4 eggs
2 egg yolks
2 tablespoons all-purpose flour
1/3 cup heavy whipping cream
2 teaspoons vanilla extract
3 (1.4 ounce) bars chocolate covered toffee, chopped
```

```
Sausage Casserole
INGREDIENTS:
1 pound sage flavored breakfast sausage
3 cups shredded potatoes, drained and pressed
1/4 cup butter, melted
12 ounces mild Cheddar cheese, shredded
1/2 cup onion, shredded
1 (16 ounce) container small curd cottage cheese
6 jumbo eggs
```

Figure 5 - Recipe Examples

This is designed to help customers remember what they need to purchase to make certain dishes. The ability to lookup dish ingredients will help speed up the shopping process which will in turn generate a faster turnover of carts. While having only included a few recipes into our database the actual grocery stores would benefit from loading as many recipes as possible. Since these are simply text files they take up relatively no space and the onboard system can hold many recipes.

The last function that we have integrated into the GUI is the item lookup button. When the user wishes to locate an item in the store they will first click on the locate button. This will then pull up a list that is organized alphabetically by first letter of the items name. The user will then scroll the first letter of the item which they wish to find and the list will tell them on what aisle it is located. During our initial thought on how the locate of items would work we thought that the most efficient manner in which to do this would be to allow the customer to type in the name of the item they are looking for. However, we found that the integration of a keyboard into the cart was not practical. The keyboard would take up too much space on the cart and so this option was not a design issue for us.

When the customer has finished their shopping with the B.O.S.S. they will need to go to the designated checkout line for users of the B.O.S.S. (Figure 6). We realize that using the B.O.S.S. still demands that the customer enter a checkout line but since the only purpose of this line is to pay, and not scan items, it will move rapidly. Upon entering the line the user will click on the checkout button which will total their purchases and display the final costs of all items they have selected. When they enter the B.O.S.S. designated checkout line they will be greeted by a cashier who will enter their total into the cash register and charge the customer. The customer will then be able to pay either by check, cash, or credit. Since this form of checking out will not give feedback to the store as to what items have been purchased we found a different way to do this. Each time that the system is totaled, the list of items will be downloaded to the from the grocery list and stored on the carts internal hard drive. This will be an updated list that will compile all items that have been purchased with that cart and can be downloaded by the grocery store whenever they desire. This will give the store feedback as to how many of each item has been purchased by the B.O.S.S. carts.

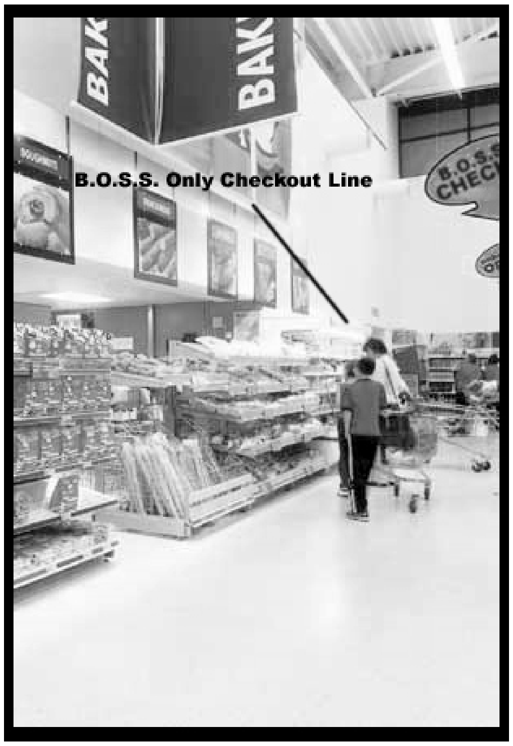


Figure 6 – B.O.S.S. Designated Line

With the designed GUI the customer is able to easily interact with the B.O.S.S.' functions and navigate it efficiently during their shopping experience. Realizing that however simple a system is there will still be issues with learning how it runs. This is also the case with the B.O.S.S. so it is recommended that prior to the first use the store will need to give the customer a brief overview. This will help alleviate customers not knowing how the functions work and will help customers use the system more efficiently.

Database Functions

The database, code found in Appendix III, file is a list of tab delimited entries in the following order: UPC code, item description, location, and price. The functions for the B.O.S.S. to utilize this database file were written in C last semester to be standalone executables, but are now integrated into the backend of the B.O.S.S.' coding to decrease response time.

When an entry is made into the UPC code text entry box, a callback function is executed that first checks to see whether the remove_item_mode flag is set, then executes the pricelookup() function returns the price and also adds the description string to the array of items (session_list) already in the cart. There are preventative checks to make sure that the UPC code is valid, and that a user cannot remove an item not in the cart.

The session_list array is dynamically updated, containing only the items currently in the cart. We chose to use an array to submit The B.O.S.S.' hard drive to as little stress as possible, and also since an array can be searched much more quickly than a text file residing on the hard drive- this results in a faster response time for the user. The actual database file is not updated dynamically, but can be updated if new items are added or deleted from inventory. Being tab delimited, the database file can be easily updated using Excel, or any other spreadsheet program.

The total, and text_view_box, which lists all items currently in the cart, along with a scroll bar if the item list increases past the height of the window, are also dynamically updated after each UPC input. If an item is removed, the item's entry in session_list is removed and the total is updated. All database functions have been tested and have been demonstrated to work successfully. Below you will find a detailed description of the primary functions of the code. While this is just a description of the main functions of the B.O.S.S. code, you can review the full code along with its comments in Appendix III.

Function Summary

text_view_update(GtkEntry*entry)

text_view_update() is passed a pointer to the GtkEntry widget, allowing the programmer to use lookup_widget(GTK_WIDGET(entry), "textview1") to find the pointer to the text_view widget.

text_view_buffer is filled one character at a time until it reaches a string terminator (\0), then it inserts a space and continues until the end of the line.

A text buffer widget is created with:

```
GtkTextBuffer *buffer
buffer = gtk_text_buffer_new (NULL);
```

The buffer widget is assigned text with:

```
gtk_text_buffer_set_text(buffer, text_view_buffer, -1);
```

And the text_view widget is finally displayed with:

```
gtk_text_view_set_buffer (GTK_TEXT_VIEW (text_view), buffer);
```

priceLookup(char upc_string[])

priceLookup() is passed a character array from the on_UPCentry_activate() callback.
A pointer to the file “barcode_database.txt” is initialized with:

```
FILE *upc_database;
upc_database = fopen("barcode_database.txt","r");
```

And the text is searched line by line by using the fgets() function, grabbing a max of 120 characters per line:

```
while( (fgets (line_string , 120, upc_database) != NULL) && found == 0 )
```

The contents of each single line is placed into dbase_upc[] character by character until a tab is found, as the database is tab delimited. The dbase_upc string is then compared to the upc_string, and the session_list is updated with the entire line_string from “barcode_database.txt”:

```
if ( strcmp(dbase_upc, upc_string) == 0 )
{
                printf("FROM PRICELOOKUP, MATCH!!!!\n");
                found = 1;
                if (remove_item_mode == 0)
                strcpy(session_list[item_count-1], strcat(line_string,""));
}
```

If the strings don’t match, the next line is searched until the end of the file.
If the entire file has been searched, and the item has not been found, an error message occurs like so:

```
if (found == 0)
        printf("Item not in database, please contact a manager.\n");
```

If the *found* flag is set to ‘1’, then the function returns the price, as a floating point value, in the database that corresponds to the upc code entered:

```
else  //if the item was found in database
{
```

```
            pn=0;
            while(TRUE) //no conditions, will just use a break statement
    {
            nextchar = line_string[i]; //grab a char, put it into a buffer
            if ( nextchar == '\n' )
            {
                    dbase_price[pn] = '\0';
                    description[dn] = '\0';
                    break;
            }
                    if ( nextchar == '\t' && tabcount < 2)  //we want to skip two fields
            {
                    tabcount++;
                    i++; //skip to next character
                    continue;  //skip this field since it ends in a tab
            }
            if (tabcount == 0)
            {
                    description[dn] = nextchar;
                    dn++;
            }
            if (tabcount == 2)
            {
                    dbase_price[pn] = nextchar;
                    pn++;
            }
            i++;
            }
    }
    rewind( upc_database);
    return atof(dbase_price);
```

popup (char *szMessage)

This function creates a button that can be passed any text string, it can be used as an error message or a helpful dialog.

on_start_clicked(GtkButton *button, gpointer user_data)

This function is passed a pointer to the button widget that was pressed, and a pointer to the *user_data* widget. It sets the *active_session* flag, updates the *total_display* to zero, and pops up a friendly welcome:

```
active_session = 1;
GtkWidget *total_display = lookup_widget(GTK_WIDGET(button), "total");
popup("Enjoy shopping with The Boss!");
gtk_entry_set_text(GTK_ENTRY(total_display), "0.00");
```

on_UPCentry_activate(GtkEntry *entry, gpointer user_data)

This function is a callback from the text entry box where a UPC code is entered. If an item is being removed, it searches the *session_list[][]* array to find an instance of the item**,** and once that item is found, it removes one entry from the *session_list*, then shifts all items in the *session_list* to eliminate the blank line:

```
while (r < sizeof(session_list)/sizeof(session_list[r])) //gives length of row
{
        while (c < sizeof(session_list[r]))
        {
                if (session_list[r][c] == '\t')  //WE ONLY WANT THE UPC HERE!!!!
STOP AT A TAB!!!
                {    //   printf("BREAKING \n");
                        c++;
                        list_upc[i]='\0';
                        break;
                }
                list_upc[i] = session_list[r][c];
                c++;
                i++;
        }

        if (strcmp(list_upc, entry_text) == 0)
        {
                printf("MATCH!!!!\n"); //DEBUG
                while(r < sizeof(session_list)/sizeof(session_list[r]))
                {
                        strcpy(session_list[r], session_list[r+1]);
                        r++;
                }
        }
```

Then the text view box is updated with:

```
text_view_update(entry);
```

If the item is being added to the *session_list*, then *item_count* is incremented and the total is updated:

```
item_count++;
total = total + priceLookup(entry_text);
```

Digital Scale

The digital scale built for the B.O.S.S. Project uses a pressure sensor to measure the weight of an object and outputs over RS-232 an 8-bit number between 0 and 255. A circuit schematic, PCB layout, and parts list have been included in the Appendix.

The digital scale was built using three primary components, which will receive further attention later in this section. The pressure sensor, a piezeo-electric resistor which when pressure is applied to it drops its resistively with a linear relation to the pressure applied. This sensor changes from approximately 1.2 kOhms to 15 kOhms as pressure is applied to its plunger. When set up as the first resistor in a voltage divider, an analog DC signal is produced. This DC voltage signal corresponds directly with the pressure that is being applied across the sensor. The second component is the PIC16F870 microcontroller; this IC performs Analog-to-Digital Conversion (ADC) on the DC analog signal obtained from the sensor, samples the signal and performs mathematical operations before outputting it as an 8 bit serial signal. The third component is the MAX233 IC, which converts the TTL serial signal output by the PIC into an RS-232 serial signal that can be transmitted serially across an RS-232 port and cable and read by another processing unit.

Other components that have been incorporated into the design are a voltage regulator, an ICD programming port, and a 9 pin Female Data Bus port. The voltage regulator allows the scale to run off a +12 volt power supply by stepping down this voltage to a constant 5 volts, a voltage that will not overpower the sensor and IC's and allow for programming over the ICD port. The ICD port allows the PIC microcontroller to be programmed once the PIC has been soldered to the board, using MPLab software and a PIC Puck. The 9 pin female RS-232 connector allows the serial signal off the MAX233 IC to be retrieved by another computer or processing unit, via a RS-232 cable. The scale only needs to transmit and has been designed without the ability to receive a RS-232 signal. Therefore the scale is always transmitting and the system is always listening, no acknowledgement of bits being sent or received is needed. This keeps the transmission over RS-232 as simple as possible. Below is a simple schematic of the layout of the system.

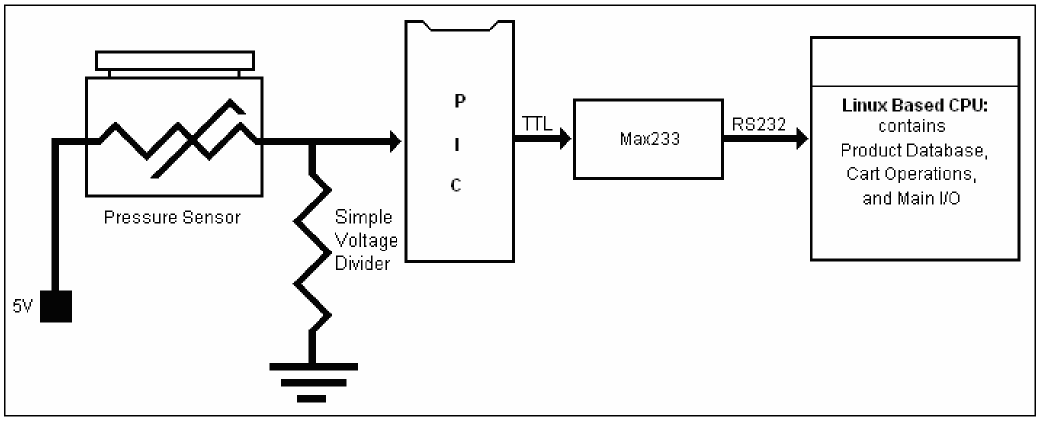

P I C
TTL
Max233
RS232
Linux Based CPU: contains Product Database, Cart Operations, and Main I/O
Pressure Sensor
Simple Voltage Divider
5V

Figure 7 – Simplified Scale Schematic

The PIC16F870 is a 28 pin microcontroller. Our design implements the PIC in its dual inline pin (DIP) package, this package proved easier to work with that a smaller surface mount version. The PIC16F870 has the ability to do both ADC and serial transmission, two features that were needed for our design. Our design uses a high-speed crystal in conjunction with two ceramic capacitors to generate a clock with a frequency of a constant 10.24MHz. The PIC uses this configuration connected across its OSC1 and OSC2 pins to effectively generate the clock. Our PIC has been programmed so that it receives the analog DC signal off the voltage divider on the AN0 pin on the PIC. From this the PIC does an 8 bit digital conversion on the signal and samples 100 times before it outputs serially a low, high, and averaged value. The PIC was programmed to allow it to sample and output in such a fashion. The programming of the PIC was done using MPLab software, where our program was written in C language and then compiled into Assembly language. A complete printout of the source and header code used to program the PIC with MPLab can be found in the Appendix. The PIC transmits TTL serially off the TX pin with a baud rate of 9600 bytes per second, no parity bit, and 1 stop bit. The Master Clear pin which is a low active is held high, by connection to the power source through a resistor. The power supply of the PIC is decoupled with a .1uF capacitor.

The MAX233 is a 20 pin IC. Our design uses the MAX233 in its DIP package. Our design allows the use of only one capacitor, used for decoupling across the power source. The MAX233 simply receives our TTL signal from the TX pin of the PIC and converts it to RS-232 serial, which connects directly to our 9 pin female RS-232 connector.

Our design allows the scale to operate off one single +12 volt power supply. The scale has also been designed so that it will start and run once power has been supplied to it and requires nothing beyond a power source to begin sampling from the sensor. No reset switches are needed in our design and therefore have not been incorporated. Our design incorporates two LED’s which have been connected so that if the first lights up then the scale has been supplied with a power supply, the second LED lights up whenever serial transmission to the 9 pin RS-232 connector takes place, resulting in a flashing of the LED. These two LED’s allow a user to quickly note whether or not power has been supplied to the scale and whether or not the scale is transmitting to the serial database.

Barcode Reader Functions

As stated in the prior sections, the B.O.S.S. will be outfitted with a handheld scanner device that will allow the customer the ability to scan items found throughout the store. They will need to use the barcode reader to scan items ranging from canned goods to meat to produce.

When the user wishes to scan an item they will first need to place the cursor in the text entry location on the GUI (Figure 8). This will allow the user to begin scanning items. Since the barcode reader is recognized by the system as a keyboard input, in order for it function properly the cursor needs to be in text entry mode. This is accomplished by placing the cursor in the text field. As aforementioned this problem was unable to be fixed through code.

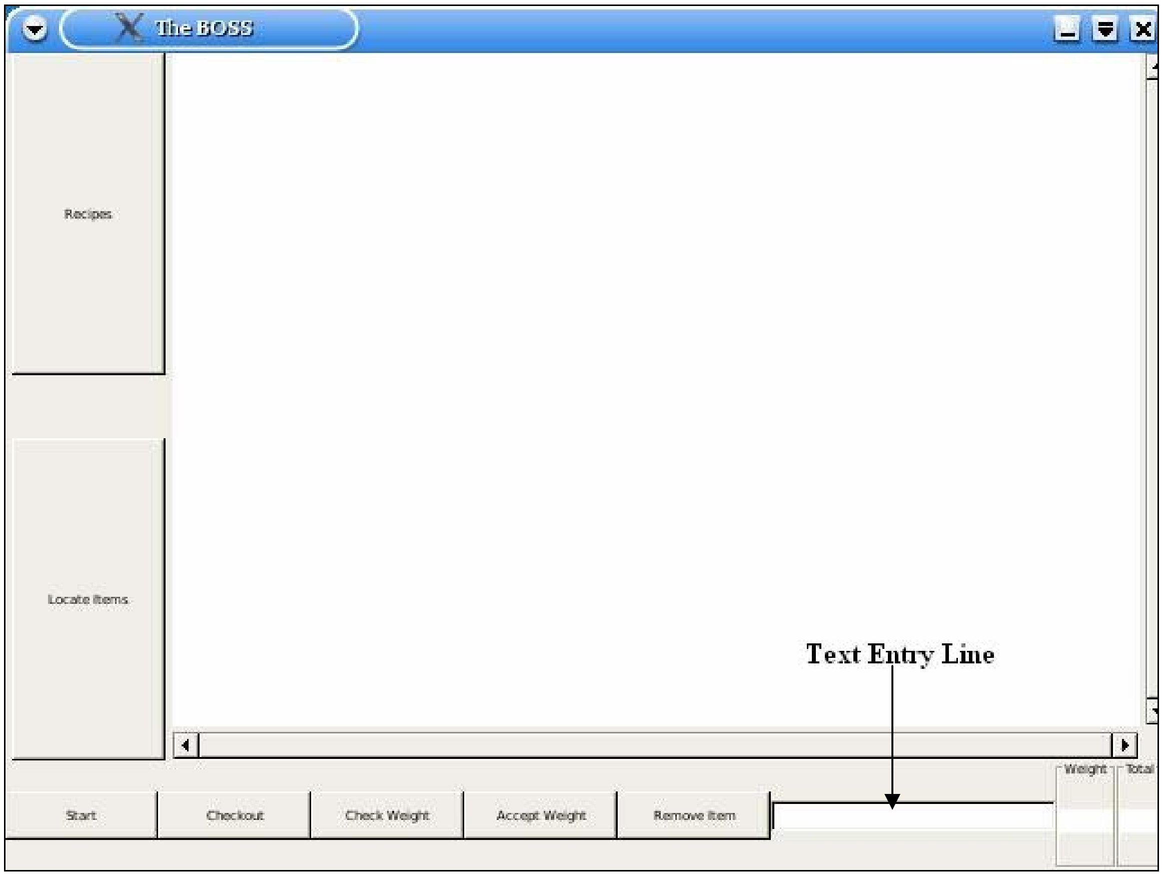


Figure 8 – Graphical User Interface

The Barcode Scanner we are using for the B.O.S.S. is the Welch Allyn ST3400 Long range decoded out Barcode Scanner. The ST3400 has a PS/2 interface with a built-in keyboard wedge. Its scanning capabilities range from 0 to 8 inches with an 8 inch maximum barcode width. The ST3400 has the very convenient characteristic of not needing extra software drivers to operate. All barcode scanned input appears as if it was typed by a keyboard. It inputs into any program that is expecting type. Also the prefix and or suffix of the scanner’s output can be easily altered to meet the needs of specific applications. The ST3400 can be set up to discriminate UPC, EAN, Code 39, Code 93, Codabar, Interleaved 2 of 5, Code 2 of 5, Matrix 2 of 5, Code 11, Code 128, MSI and Plessey barcode types.

This scanner is more than capable of handling all the complexities of this system. However, we initially considered the Digital Convergence.com™ Cat Optical Reader but we were unable to locate drivers that would work with Linux. With the Cat Optical Reader unable to function we decided on the Welch Allyn Barcode Scanner. After testing the Welch Allyn scanner it was determined to be more than capable for our project.

Power Supply

To power the B.O.S.S.' components it decided to use a deep cycle boat battery (Figure 9). The deep cycle option was selected because deep cycle batteries are designed to be discharged down as much as 80% time after time, and have much thicker plates. A deep cycle battery is designed to provide a steady amount of current over a long period of time. Deep cycle batteries can provide a surge when needed, but nothing like the surge a car battery can. These batteries are designed to be deeply discharged over and over again which is something that would ruin a car battery. Since the B.O.S.S. will need a power supply that will be able to handle consistent charging and discharging this was a perfect power source for our project.

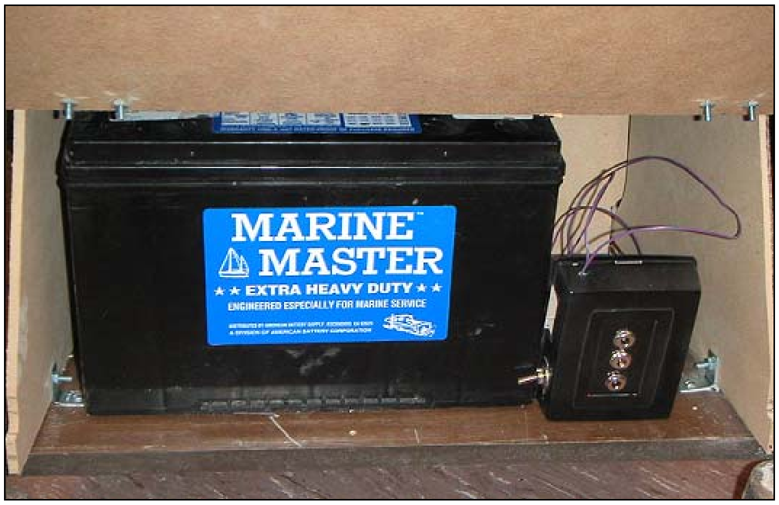


Figure 9 - Power Supply

With the deep cycle battery selected it was now necessary to figure out how to connect the components to it. Keeping safety in mind, learned in our Biomedical Engineering course, it was decided that the fewer connections made to the battery the better. By limiting the number of wires that were directly connected to the battery leads this would help minimize the possibility of electrical injury. A three plug type two mini outlet housing device was fabricated to plug our devices into. This allowed us to bypass all power inverters which were needed to convert the 120V AC to the 12V DC needed to power the components (monitor, CPU, and scale). The box was then connected to the boat battery to power the components. After testing our fabricated power box and power cords using the power supplies and volt meters in the lab it was determined that all fabrications were safe. Once determined that the devices were functioning properly we retested them using the actual boat battery.

To recharge the batteries at night the grocery stores will need to be outfitted with a charging system that is similar to ones found in cart barns at golf courses (Figure 10). This will allow them to fully charge multiple B.O.S.S. systems during the night. The cart was designed to have a hinged battery and CPU compartment to allow easy access to the battery for charging purposes and also to allow the battery to be removed in case of failure. However, for the purpose of our project a trickle charger will be used to charge our battery.

Figure 10 - Golf Cart Charging Units

With the cost of the deep cycle boat battery running rather high, $75.00, it was realized that this is decreasing the feasibility of deploying this system in grocery stores. This was the most reasonable option for us and if the stores decided to use the same battery they could negotiate a bulk discount for the batteries. Not having the ability to determine what price could be negotiated if purchased at bulk, it was unable to be determined how much this type of battery would ultimately run the manufactures of the B.O.S.S.

Display

When deciding on what type of a display to use, it was first necessary to determine where to mount the screen. Our initial idea was to mount the screen next to the scale which would place the screen between the customer and the basket of the cart. This would be a design error and would get in the way of the consumer. The feasibility of a hand held screen was also examined but was decided against due to the chances that this could get broken or damaged. The next option that was ultimately selected with was to place the screen at the end of the cart as illustrated in Figure 11.

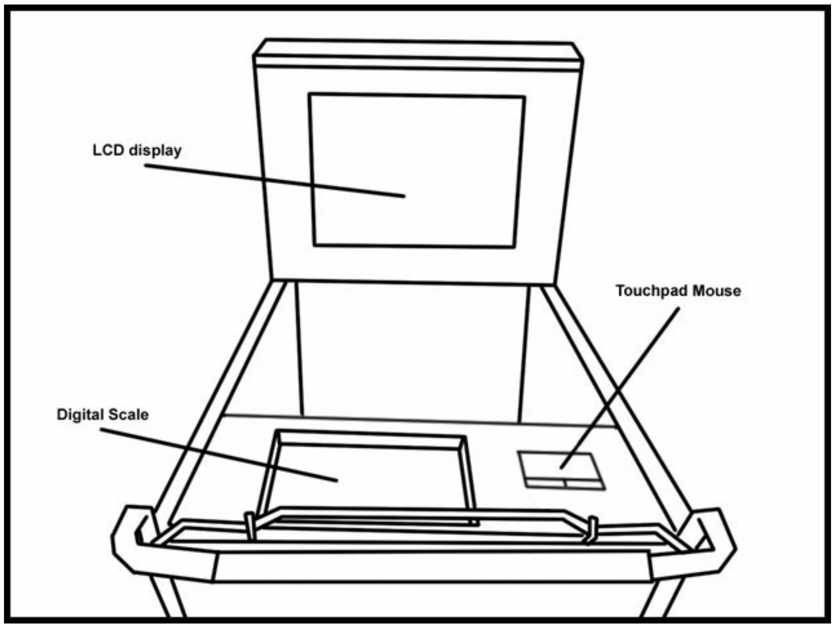


Figure 11 - Display Location

This allows the user to easily place items in the cart without having to maneuver around the cumbersome screen. However, by placing the screen at the end of the cart it became necessary to have a much larger screen. With the anticipated screen size of around 7-10 inches it was determined that some users might not be able to read the screen from this distance. After a trip to Fry's Electronics it was discovered that a 15+ inch monitor was beyond the budget for this project so it was decided that the B.O.S.S. would use the HP LCD monitors that our found in our lab. A screen box was fabricated to fit the HP 15 inch monitor perfectly and the monitor will be able to be inserted on the day of demonstration. It is understood that this sized monitor is highly expensive but it is believed that if the grocery store was to order them in bulk they would be able to negotiate a lesser cost. Had budget not been an issue for our group the option of a touch screen LCD would have been explored. This would alleviate the need for a touchpad mouse and would open up different mounting options for the screen.

Cart Fabrication

With components of the B.O.S.S. needing a place to live it was necessary to fabricate a cart that could house these items in the most efficient manner. The fabrication dimensions were created to fit the cart used in this project only. As seen in Figure 12 a stock grocery cart was greatly modified to house the totality of the B.O.S.S.

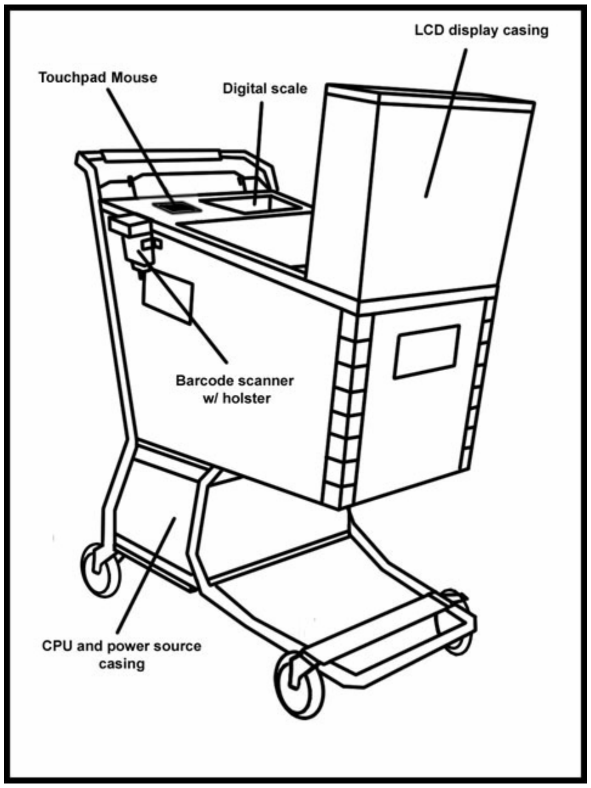


Figure 12 - Cart Design

To fabricate the components housing on the cart 1/4 inch masonite board was used to design each part. L-brackets were used in all box fabrications and only nuts and bolts were used to hold parts together. This will allow us to disassemble any part if necessary. Since this is a prototype of the actual B.O.S.S. that grocery stores would purchase, it was decided to dress this one up. Purple carpet for the inside and parts of the outside was selected to give the B.O.S.S. that extra bit of sex appeal. Realizing that this is not practical for the actual units it was determined that in order to sell this item you need to catch someone’s eye first. For the organic shape of the screen housing planters foam was shaped and coated it in BONDO to help give it strength. The BONDO was then sanded to form the desired shape. This dressed up the cart nicely and made it more appealing to the eye. In order to give the cart a cohesive feeling it was decided that the cart should be painted all black. This gives it a sleek look as a prototype and will help create a buzz for consumers.

As described in the display section of the report, a housing unit was fabricated (Figure 13) to hold the 15 inch HP monitor. It was designed to hold a monitor with a screen size of 15 inches. However, enough room was left on the sides that a different brand monitor would also be able to fit into the housing. This would allow the grocery store to select the most economically appealing 15 inch LCD monitor. As stated above the wood and foam monitor casing has been shaped to form an organic housing structure. The foam was then covered in BONDO that was shaped into the final shape as seen in the overall picture. This housing provides a sturdy and rigid place to keep the monitor.

With the system incorporating a digital scale it was needed to somehow integrate this into the design of the cart. As seen in Figure 13 the scale was placed near the customer and the mouse. This will allow easy access to the digital scale and we could conceal its components. A flat bottom vertical scale design was selected which allows the user to load as much of an item as they so desire. When integrating the scale it was taken into account that some items, like cantaloupe, tend to roll around if you place them on a flat surface. By insetting the scale bottom in relation to the sides, a perimeter fencing was created that will confine all items that are being weighed. The printed circuit board and plunger are housed below the scale's bottom. This will keep them our of harms way but still allow access to them if needed.

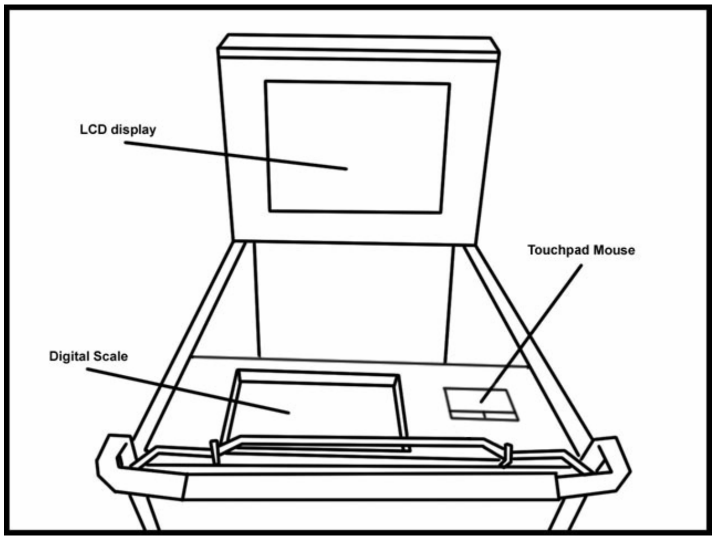


Figure 13 - Monitor, Scale, and Mouse

To select what option that the customer wishes to perform on the B.O.S.S. a touchpad mouse was integrated into the system. This type of navigation device was selected because it allows the user to control the cursor without having to move the whole mouse. Rather they can simply touch the pad to change cursor location. It has also been inset into the system interface panel of the cart. This will prevent the mouse's casing from moving around or being stolen.

Benefits and Constraints

When looking at the design of the B.O.S.S. it was not only necessary to look at the feasibility of the design but also at the constraints that were necessary to plan into the design. Below you will find a breakdown of the major constraints that were factored into the design of the B.O.S.S.

Economic Considerations:
Being college students with definite budget concerns, we have decided that $500.00 is a reasonable cost for the duration of the project. Hopefully we can be granted at least half by Associated Students or other similar organizations.

On the macroscopic scale we believe our system is very beneficial in the area of cost efficiency for the retail industry. Our system decreases demand for checkout clerks and other retail employees which obviously lowers costs. The issue of loss control that is relevant to our type of system is overshadowed by a lessened need for health insurance and workers compensation costs.

Environmental Considerations:
The design will not produce any hazardous waste to the environment and the production of the carts components will not be environmentally exhaustive. The greatest use of natural resources will come in the way of providing steel to build the metal frame of the shopping cart, however certainly using recycled steel will lighten the burden on the environment.

Manufacturability Considerations:
Manufacturing the design can be accomplished by contracting out portions of the design. For example the Microcontroller unit can be order from a PCB producing company and individuals components can be assembled at an assembly plant. Costs of the electronic components will be, for the most part, minimal. The greatest costs will come in the way of the assembly of the cart.

Reliability and Sustainability Considerations:
Because of the nature of the design and its intended monetary use, the system must be reliable and error free. If it is not there will be obvious. Even small miscalculations, when compounded, can add up to large costs to the customer, the business, or both. It can not be overstated how important reliability is to our project.

Legal Considerations:
We have yet to find any significant patent issues that directly relate to our system. However, the barcode scanner company, Symbol, makes a similar handheld product that has yet to be implemented in any consumer situation. They have also partnered with Palm to manufacture a product that is readily used for inventory and warehouse operations. It should also be considered that our investigation of legal considerations is an ongoing process and will continue to grow in depth as the project moves forward.

Health and Safety Considerations:
We do not believe that the quick checkout system has any serious health or safety issues. A minor cause of concern could be in a situation where the handheld barcode scanner is used as a weapon for children to hit each other with.

Social, Political, and Ethical Considerations:
Our quick checkout system, like other automated advancements in retail and inventory operations, can be labeled as anti-union. The decreased need for checkout employees at grocery and other retail stores makes our project business management friendly but also represents the majority of the social, political, and ethical considerations of our project.

Task Descriptions

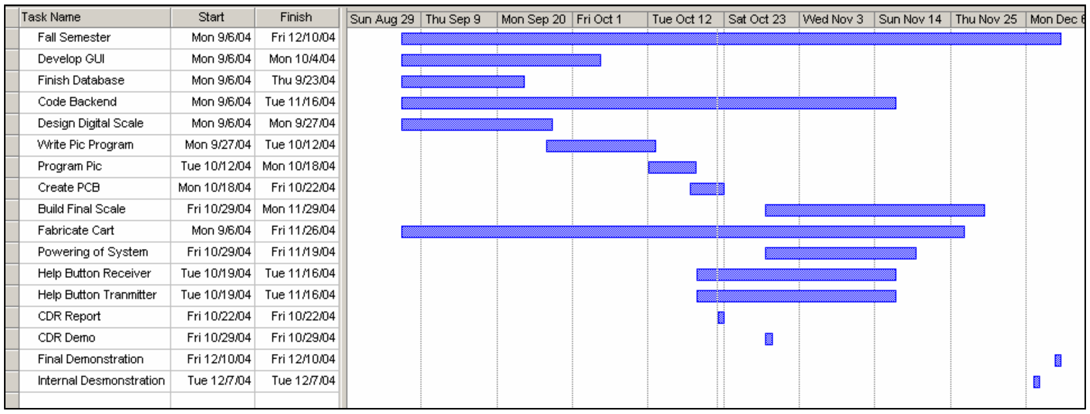
The work tasks needed for the successful completion of the project were shared between the members of the B.O.S.S. team, the major tasks of each participant can be found below:

Bill Leslie:	Designed a graphical user interface (GUI), worked with the power aspects of the project, and cart fabrication.
Erik Loftis:	Designed the digital scale from the pic programming to the design on the PCB and worked with the power aspects of the project.
Dustin Mendes:	Designed team website, worked with the power aspects of the project, early work with the digital scale, and cart fabrication.
Brian Momeyer:	Designed the graphical user interface (GUI), the backend computer code, worked with the power aspects of the project, and the product database.

Design Schedule

All major systems developments and project milestones were completed on time which ensured a successful finale to the B.O.S.S. A Gantt of our last semester's task can be found below. As seen the system in its entirety was completed by our end of semester demonstration.

This project has taught the members of the B.O.S.S. how to set development goals and work towards an end product. The members will take away the ability to forecast work and stick to a schedule.



Task Name	Start	Finish
Fall Semester	Mon 9/6/04	Fri 12/10/04
Develop GUI	Mon 9/6/04	Mon 10/4/04
Finish Database	Mon 9/6/04	Thu 9/23/04
Code Backend	Mon 9/6/04	Tue 11/16/04
Design Digital Scale	Mon 9/6/04	Mon 9/27/04
Write Pic Program	Mon 9/27/04	Tue 10/12/04
Program Pic	Tue 10/12/04	Mon 10/18/04
Create PCB	Mon 10/18/04	Fri 10/22/04
Build Final Scale	Fri 10/29/04	Mon 11/29/04
Fabricate Cart	Mon 9/6/04	Fri 11/26/04
Powering of System	Fri 10/29/04	Fri 11/19/04
Help Button Receiver	Tue 10/19/04	Tue 11/16/04
Help Button Tranmitter	Tue 10/19/04	Tue 11/16/04
CDR Report	Fri 10/22/04	Fri 10/22/04
CDR Demo	Fri 10/29/04	Fri 10/29/04
Final Demonstration	Fri 12/10/04	Fri 12/10/04
Internal Desmonstration	Tue 12/7/04	Tue 12/7/04

Figure 14 – Project Schedule in Gantt Chart Form

Budget/Costs

Below you will find the approximate budget for the B.O.S.S. All the required hardware needed to complete this project was easily obtained commercially and with the help of a grant provided to us from the Associated Students here at USD, the project was completed independently of team member funds.

ITEM	COST PER ITEM	# OF ITEMS	SUBTOTAL
Microcontroller	$6.00	3	$18.00
Scanner	$55.00	1	$55.00
Scale Components	$75.00	1	$75.00
Scale PCB	$20.00	3	$60.00
Mini ITX	Donated $300.00	1	$300.00
Display*	Borrowed $200.00	1	$200.00
Cables	$12.00	2	$24.00
Housing and Hardware	$50.00	1	$50.00
Battery	$75.00	1	$75.00
Transmitter/Receiver Components	$25.00	1	$25.00
Grocery Cart	Donated $150.00	1	$150.00
		TOTAL COST	**$1032.00**

Table 1 - App. B.O.S.S. Budget

***Note:** as aforementioned we utilized a monitor provided by USD

The above budget was what the final design cost the group to fabricate. As shown above some items were donated to the group and this made it possible to create the prototype. As aforementioned, by buying items in bulk the grocery stores and cart manufactures could drive down fabrication costs by purchasing items in bulk. Overall, the design met cost expectations that the group set at the beginning of the project. The group ran into design difficulties and had to seek alternative methods for solving these problems. One example of this was an inability to get our first scanner to function properly and had to reorder a different model which will work.

Conclusion

With the Buyer Operated Shopping System complete, an all inclusive system was created that allows customers to shop easier. The B.O.S.S. incorporates all the needed functions necessary to checkout and also gives the consumer some new features to make shopping easier. From the ability to look up stored recipes to being able to locate what aisle an item is found on the B.O.S.S. will improve the customer's experience.

By implementing our system into grocery stores, supermarkets will be on the cutting edge of technology and will be able to reach out to new customers. This will help increase their sales but will also drive down employee costs. This system will take the place of previously needed cashiers which will allow the store to save money.

Completion of the Buyer Operated Shopping System assisted in our professional growth as engineers by providing valuable experience in terms of meeting design criteria and solving problems. For more information regarding the Buyer Operated Shopping System you may refer to our website which is located at www.sandiego.edu/~dmendes.

Bibliography

Market Research:

1. “Scan-it-Yourself' Device Allows Shoppers to Eliminate Time in the Checkout Line.” Symbol Technologies, Inc. http://www.symbol.com/news/pressreleases/press_releases__miscpr__kroger.html

2. “Home Depot Reengineers Front-end Capabilities.” 360 Commerce. http://www.360commerce.com/itn_a_7.html. 02/01/03

General UPC Information:

3. Brian, Marshall. “How UPC Barcodes Work.” How Stuff Works. http://electronics.howstuffworks.com/upc.htm/printable

PIC Microcontroller Reference:

4. Peatman, John B. Embedded Design with the PIC18F452 Microcontroller. Prentice Hall.

5. “Program Microchip PIC micros with C.” Microchip.com. http://www.piclist.com/techref/io/lcd/pic.htm

C Programming References:

6. Dale, Nell B. Programming and Problem Solving With C++. Microchip.com. http://www.microchipc.com.

7. Zak, Diane. An Introduction to Programming with C++. International Thomson Publishing.

Filter Reference:

8. Ludwig, Reinhold and Bretchko, Pavel. RF Circuit Design: Theory and Applications. Prentice Hall.

Industry Contacts:

Larry Smith
Lighthouse Computing Systems
Ramona, CA
Larry specializes in POS equipment and has been very gracious to share his knowledge with the Quick Cart group.

Cary Doris
USD Bookstore
Shipping and Receiving Supervisor
Cary is not an electrical engineer but he is an expert in scanner hardware and POS systems in general.

Appendix I

Background

The barcode can be traced back to two individuals, Bernard Silver and Norman Joseph who were tasked with a design problem presented by a local food chain president. The task was to develop a system that automatically read product information during checkout without making the clerks look up the prices manually. For their initial design the two engineers used ultraviolet light to read a pattern of concentric circles that would tell them what item was scanned.

Then in 1966 the Barcode was commercialized when the National Association of Food Chains (NAFC) challenged equipment manufacturers to develop a system that would speed the checkout process. In 1967, RCA installed one of the first scanning systems at a Kroger store in Cincinnati. The product codes were represented using Silver and Woodland's method of circle styled barcodes. The success of the RCA/Kroger code led to the food retail industry agreeing on a need for a standard coding scheme that could be adopted by all equipment manufacturers, food producers, and product dealers. In 1969, Logicon, Inc. by request of the NAFC began working on a design that would standardize the barcode system.

The results of Logicon's work were Parts 1 and 2 of the Universal Grocery Products Identification Code (UGPIC) which was presented in the summer of 1970. Based on the recommendations of the Logicon report, the U.S. Supermarket Ad Hoc Committee decided on a Uniform Grocery Product Code. Three years later the Committee recommended the adoption of the UPC symbol which is still used in the United States today. The UPC symbol was submitted by IBM and developed by George Laurer whose work was a direct outgrowth of Silver and Woodland's earlier idea.

In 1974 one of the first UPC scanners, made by NCR Corporation, was installed at Marsh's supermarket in Ohio. On June 26 of that year the first product with a bar code was scanned at a check-out counter. It was a 10-pack of Wrigley's Juicy Fruit chewing gum. Today that pack of gum is on display at the Smithsonian Institution's National Museum of American History.

The development of the UPC system made grocer inventory methodology easier, smoother consumer/store interaction, and eventually an explosion of automated systems for all types of consumer applications. Most notably, larger retail stores such as Ralph's and Home Depot have implemented self-checkout systems which allow customers to check, bag, and pay for their items without the need of a cashier.

Home Depot, specifically, started testing self-checkout units in early 2002. The home improvement retailer called their system "FAST", a front-end accuracy and service transformation. They say the system works to improve speed, accuracy, and service at checkout locations. Working with NCR Corporation Home Depot's system consists of a user-friendly touch-screen and a GUI-based POS Java application. It works as follows:

After finishing their run through Home Depot, shoppers proceed to a stationary checkout center at the front of the store. The shopper scans their items and places them in store bags which are located on a large scale type platform. This platform makes sure that bagged items are scanned and eventually paid for. As of mid-2003, Home Depot had NCR FastLane self-checkout terminals in 55 stores around the country.

Despite the obvious benefits that this type of stationary self checkout system creates for consumers as well as stores, it still tends to result in checkout lines because of its stationary nature. While these systems do help speed up checkout times, consumers are limited by the lack of stationary units at a given store.

Dating back to the 1990's, Symbol Technologies Inc. the world standard in bar code-driven data transaction systems was developing and applying a portable shopping system. In 1997 they had announced an agreement with the Kroger supermarket chain to implement their Symbol Portable Shopping System™. Symbol's system allowed customers to scan bar-coded merchandise in the aisles as they shop. Their claim was that this greatly reduced time spent in checkout lines.

The Symbol Portable Shopping System™ is simple to use and is user friendly. Upon entering the store a shopper inserts an identification card to remove a portable shopping device. The lightweight scanner is equipped with "plus," "minus," and "equal" keys, as well as a small visual display. As the shopper selects merchandise, they can scan each item's bar code using the "plus" key and places the item in a cart. If a shopper decides against purchasing any particular item it may be canceled by way of the "minus" key followed by rescanning the product. After this is done its price is subtracted from the shopping total. The shopper can obtain a subtotal of purchases by pressing the "equals" key at any time. When shopping is complete the scanner is returned to the rack and the shopper advances to an express pay station.

Currently Symbol's Portable Shopping System™ is being increasingly implemented in stores nationwide. The "PSS" solution is portable and seemingly simple to use but it does not consider per weight purchases. This seems like a design flaw in grocery store application because of the large amounts of fresh produce purchasing that takes place. This is where our Buyer Operated Shopping System picks up the loose ends.

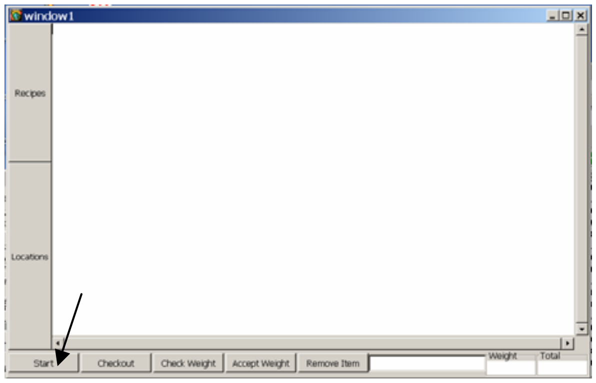
Appendix II

User Manual

Upon entering the store and checking out a B.O.S.S. the user should follow these easy steps to help navigate through the functions of the system.

Starting the System

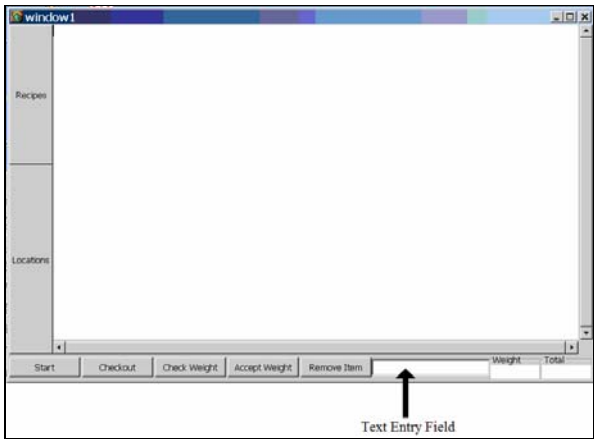
1. Click on the button labeled “Start” with the touchpad mouse.



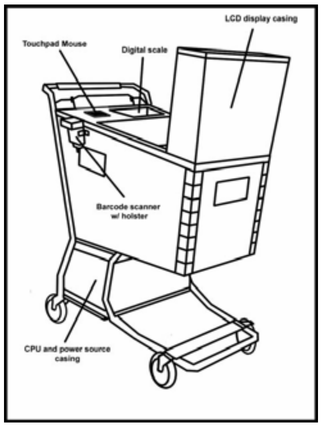
This will start the system and allow the user to begin their shopping experience

Scanning Items

1. Click on the text entry field with the touchpad mouse.



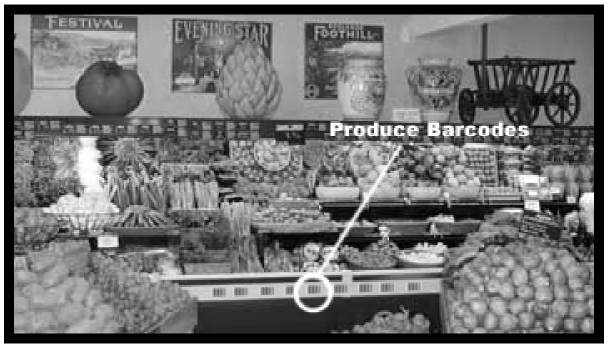
2. Now scan items using the supplied barcode reader. The item and its cost have now been added to the grocery list.



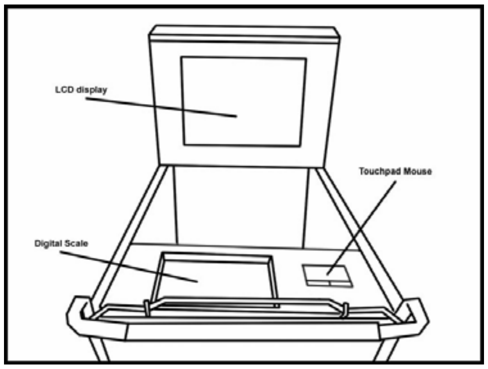
3. Continue scanning items until you wish to change functions.

Adding Produce

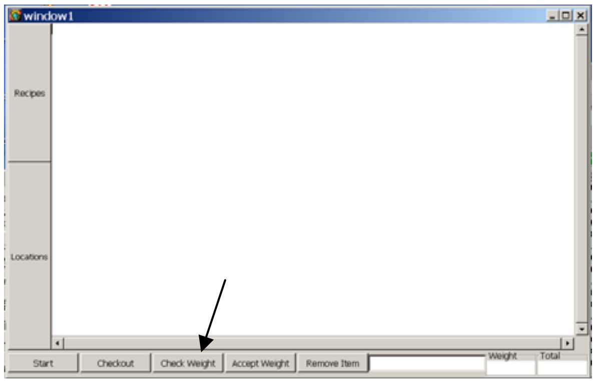
1. Scan the produce's barcode that can be found on the stand with the accompanied barcode reader.



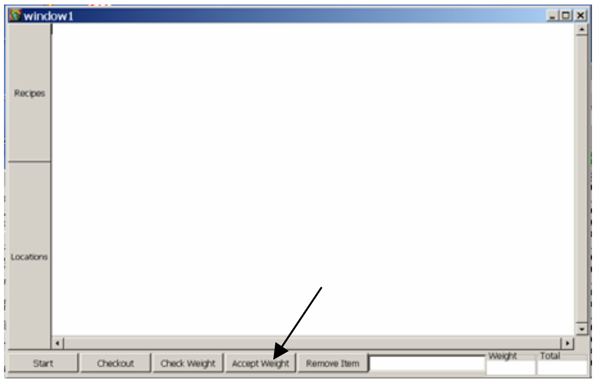
2. Add the desired produce to the digital scale.



3. Click on the button labeled “Check Weight” with the touchpad mouse.



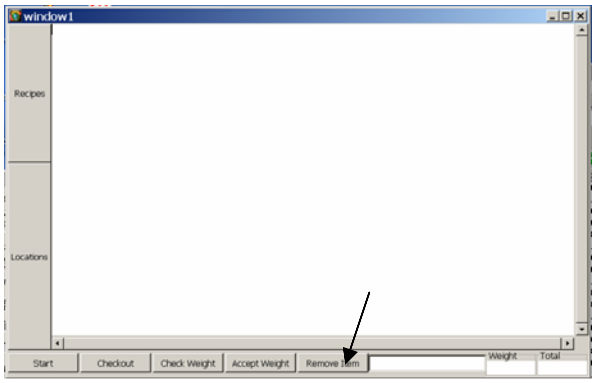
4. If the desired weight was added click on the button labeled “Accept Weight” with the touchpad mouse. The item and its cost have been added to the grocery list.



5. If desired weight was not accomplished return to step two and proceed again.
6. To continue shopping click on the desired button with the touchpad mouse.

Removing a Previously Scanned Item

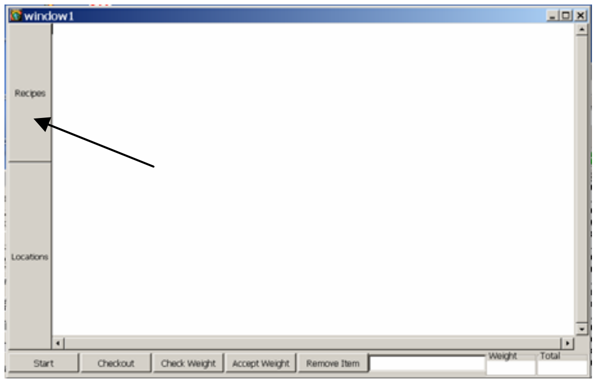
1. Click on the button labeled “Remove Item” with the touchpad mouse.



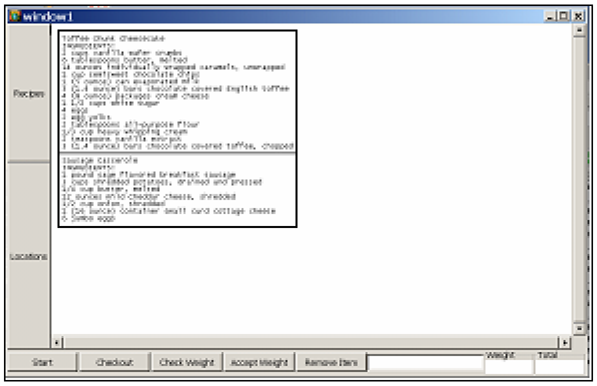
2. Scan the item’s barcode that you wish to remove with the barcode reader.
3. Remove the item from your cart.
4. The grocery list and total have now been updated.
5. To continue shopping click on the desired button with the touchpad mouse.

Recipe Look Up

1. Click on the button labeled “Recipes” with the touchpad mouse.



2. This will then pull up the recipe look up table.
3. Click on the recipe that you wish to see with the touchpad mouse and the ingredients will be displayed.



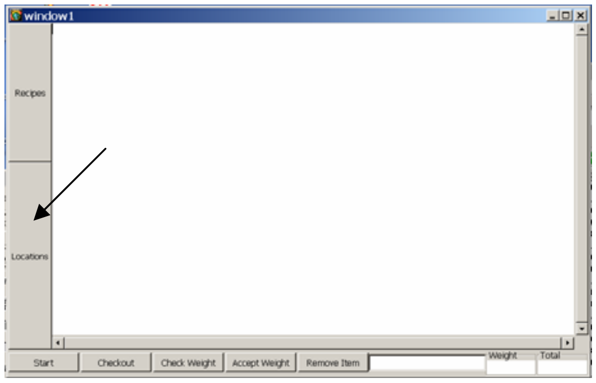
```
Toffee Chunk Cheesecake
INGREDIENTS:
1 cups vanilla wafer crumbs
6 tablespoons butter, melted
14 ounces individually wrapped caramels, unwrapped
1 cup semisweet chocolate chips
1 (5 ounce) can evaporated milk
1 (1.4 ounce) bars chocolate covered English toffee
4 (8 ounce) packages cream cheese
1 1/2 cups white sugar
4 eggs
1 egg yolks
1 tablespoons all-purpose flour
1/3 cup heavy whipping cream
1 teaspoons vanilla extract
1 (1.4 ounce) bars chocolate covered toffee, chopped
```

```
Sausage Casserole
INGREDIENTS:
1 pound sage flavored breakfast sausage
1 cups shredded potatoes, drained and pressed
1/4 cup butter, melted
12 ounces mild Cheddar cheese, shredded
1/2 cup onion, shredded
1 (16 ounce) container small curd cottage cheese
6 jumbo eggs
```

4. To continue shopping click on the desired button with the touchpad mouse.

Item Locations

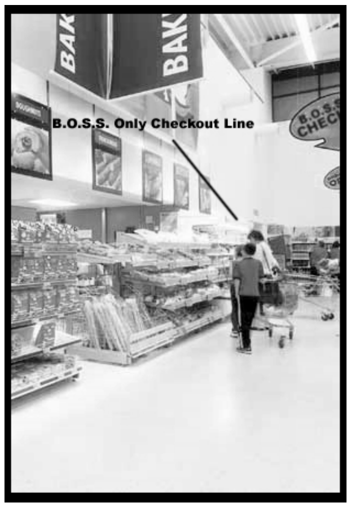
1. Click on the button labeled "Locations" with the touchpad mouse.



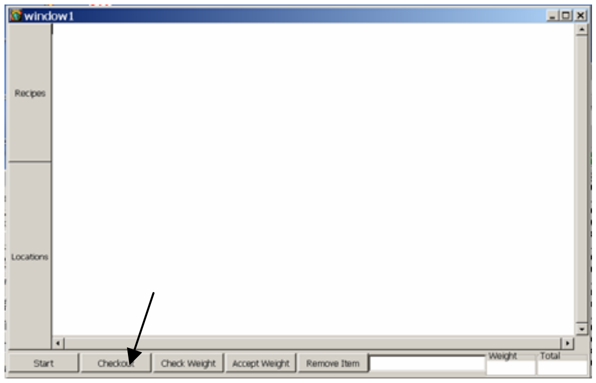
2. This will then pull up a list where you can select the item that you are looking to locate with the touchpad mouse.

Checking Out

1. When finished shopping proceed to the B.O.S.S. only designated line.



2. Once the cashier tells you to, click on the button labeled "Checkout" with the touchpad mouse.



3. The cashier will now ask you for your choice of payment.
4. You will now be instructed to return the B.O.S.S. unit and pick up your deposit.

Appendix III

Callbacks.c

```
/*
 * Brian Momeyer, Fall 2004
 * This is where the magic happens, the backend, the callbacks...
 * These assign functions to all the widgets in the interface.
 *
  */

#define TRUE 1
#define UPC_MAX_LENGTH 20
#define DESCRIPTION_MAX_LENGTH 90
#define      LINE_MAX_LENGTH 120
#define PRICE_MAX_LENGTH 7

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <gtk/gtk.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>  //libs for Unix file I/O
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#include "callbacks.h"  //Glade stuff
#include "interface.h"
#include "support.h"

const char DATABASE_NAME[] = "barcode_database.txt";
#define MAX_ITEMS 100 //change later, or not, whatever
#define ITEM_LENGTH 100 //seems reasonable
//const char SESSION_DATABASE_NAME[] = "session_database.txt";
        /*Hard drive is going out, I probably should use an array to hold the session list */
//FILE *session_database;

float priceLookup(char[]);
int descriptionLookup(int); //return pointer to a character array
int locationLookup(int);


char session_list[MAX_ITEMS][ITEM_LENGTH];
char description[DESCRIPTION_MAX_LENGTH];
char location[2];
char upc[UPC_MAX_LENGTH];
char temp_string[10];
char text_view_buffer[100];
float price, total = 0;
int active_session = 0; //set this bad boy to 1 to keep active files from being overwritten, etc
int remove_item_mode = 0;
int item_count = 0;

void text_view_update(GtkEntry *entry)
{
        int i = 0, c = 0, r = 0;
        while (r < item_count)
        {
                while (c < sizeof(session_list[r]))
                {
//                      if ( remove_item_mode == 1 && c == 0  && r == (item_count -1) )  //changes the output for an item
removal
//                      {
//                              text_view_buffer[i]='-';
//                              i++;
//                              c++;
//                              continue;
//                      }
                        if (session_list[r][c] == '\0')  //these string terminators make me want to
                                                // strangle someone, perhaps Bill
                        {
                                text_view_buffer[i]=' ';
                                i++;
                                c++;
                                break;
                        }
                        text_view_buffer[i]=session_list[r][c];
                        printf("TBV %c",text_view_buffer[i]);
                        i++;
                        c++;
                }
                text_view_buffer[i] = '\n';

                i++;
                r++;
                c=0;

        }

        text_view_buffer[i] = '\0';

        GtkWidget *text_view = lookup_widget(GTK_WIDGET(entry), "textview1");
        GtkTextBuffer *buffer;
//      text_view = g_object_new (GTK_TYPE_TEXT_VIEW, "havoc rules",NULL);
        buffer = gtk_text_buffer_new (NULL);
        gtk_text_buffer_set_text(buffer, text_view_buffer, -1);
//      gboolean gtk_text_view_forward_display_line(GtkTextView *text_view, GtkTextIter *iter);
//      gtk_text_buffer_set_text(buffer, "second line\n third line\n fourth line\n", -1);
        gtk_text_view_set_buffer (GTK_TEXT_VIEW (text_view), buffer);
        printf("TEXT VIEW BUFFER IS %s\n", text_view_buffer);
//      gtk_entry_set_text(GTK_ENTRY(total_display), "0.00");
//      printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
}
float priceLookup( char upc_string[] )
{
        char line_string[LINE_MAX_LENGTH], dbase_upc[UPC_MAX_LENGTH],
dbase_desc[DESCRIPTION_MAX_LENGTH], dbase_loc[2],
                dbase_price[7];
        char nextchar;
        int found = 0;
        int i; //index for characters on any single line in database
        int pn = 0; //index for character position in dbase_price
        int dn = 0; // " " description
        int ln = 0; // " " location
        int tabcount = 0;
//DEBUG
        printf("FROM PRICELOOKUP UPC string is %s\n",upc_string);
//DEBUG
        FILE *upc_database;
        upc_database = fopen("barcode_database.txt","r");
        while( (fgets (line_string , 120, upc_database) != NULL) && found == 0 )
        {
                i=0;
                pn=0;
                while(TRUE) //no conditions, will just use a break statement. Is this good? I don't know.
                {
                        nextchar = line_string[i]; //grab a char, put it into a buffer
                        if ( nextchar == '\t' )
                        {
                                dbase_upc[pn] = '\0'; //use this to close the strings
                                i++;
                                pn = 0;
                                break;
                        }
                        else
                        {
                                dbase_upc[pn] = nextchar;
```

```
				}
				i++;
				pn++;
			}
			if ( strcmp(dbase_upc, upc_string) == 0 )
			{
				printf("FROM PRICELOOKUP, MATCH!!!!\n");
				found = 1;
				if (remove_item_mode == 0)
					strcpy(session_list[item_count-1], strcat(line_string,"")); //kludgy, but works
				break;
			}
			else
			{
				for (i=0; i < pn; i++) //ZERO OUT STRINGS OR STRCMP() WONT WORK
					dbase_upc[i] = '\0';
			}

		}

		if (found == 0)
			printf("Item not in database, please contact a manager.\n");
		else  //if the item was found in database
		{
			pn=0;
			while(TRUE) //no conditions, will just use a break statement. Is this good? I don't know.
			{
				nextchar = line_string[i]; //grab a char, put it into a buffer
				if ( nextchar == '\n' )
				{
					dbase_price[pn] = '\0';
					description[dn] = '\0';
					break;
				}
				if ( nextchar == '\t' && tabcount < 2)  //we want to skip two fields
				{
					tabcount++;
					i++; //skip to next character
					continue;  //skip this field since it ends in a tab
				}
				if (tabcount == 0)
				{
					description[dn] = nextchar;
					dn++;
				}
				if (tabcount == 2)
				{
					dbase_price[pn] = nextchar;
					pn++;
				}
				i++;
			}
		}
		rewind( upc_database);
		return atof(dbase_price);

	}



	void CloseDialog (GtkWidget *widget, gpointer data)
	{
		gtk_widget_destroy(GTK_WIDGET(data));
	}

	void ClosingDialog (GtkWidget *widget, gpointer data)
	{
		gtk_grab_remove(GTK_WIDGET (widget));
	}
```

```
void
popup (char *szMessage)
{
		static GtkWidget *label;
		GtkWidget *button;
		GtkWidget *dialog_window;

		dialog_window = gtk_dialog_new();

		gtk_signal_connect (GTK_OBJECT (dialog_window), "destroy", GTK_SIGNAL_FUNC(ClosingDialog),
&dialog_window);

		gtk_window_set_title(GTK_WINDOW (dialog_window), "System Message");

		gtk_container_border_width(GTK_CONTAINER (dialog_window), 5);

		label = gtk_label_new(szMessage);
		gtk_misc_set_padding(GTK_MISC(label), 10, 10);

		gtk_box_pack_start(GTK_BOX (GTK_DIALOG ( dialog_window) ->vbox), label, TRUE, TRUE, 0);

		gtk_widget_show(label);

		button = gtk_button_new_with_label("Ok");

		gtk_signal_connect(GTK_OBJECT(button), "clicked", GTK_SIGNAL_FUNC(CloseDialog), dialog_window);

		GTK_WIDGET_SET_FLAGS(button, GTK_CAN_DEFAULT);

		gtk_box_pack_start(GTK_BOX(GTK_DIALOG(dialog_window) ->action_area), button, TRUE, TRUE, 0);

		gtk_widget_grab_default(button);

		gtk_widget_show(button);

		gtk_widget_show(dialog_window);

		gtk_grab_add(dialog_window);

}

void
on_start_clicked			(GtkButton	*button,
				gpointer	user_data)
{
		if (active_session == 1)
		{		popup("There is currently an active shopping session!\n");
		}
		else
		{


				active_session = 1;
				GtkWidget *total_display = lookup_widget(GTK_WIDGET(button), "total");
		//		gtk_entry_set_text(GTK_ENTRY(total_display), "JIGGA!");
				popup("Enjoy shopping with The Boss!");
				gtk_entry_set_text(GTK_ENTRY(total_display), "0.00");

		}
}

void
on_checkout_clicked		(GtkButton	*button,
				gpointer	user_data)
{
		int i=0;
		while (i<item_count)
		{
				printf("\n %s \n", session_list[i]);
				i++;
		}
}


void
on_accept_clicked                      (GtkButton       *button,
                                        gpointer         user_data)
{

}


void
on_remove_clicked                      (GtkButton       *button,
                                        gpointer         user_data)
{
		if (item_count == 0)
				popup("You have no items to remove.");
		else
				remove_item_mode = 1;
}


void
on_checkweight_clicked                 (GtkButton       *button,
                                        gpointer         user_data)
{

}

void
on_UPCentry_activate                   (GtkEntry        *entry,
                                        gpointer         user_data)
{
		int r = 0;
		int c = 0;
		int i = 0;
		float price;
		const gchar *entry_text;
		entry_text = gtk_entry_get_text (GTK_ENTRY (entry));
		int entry_text_size = sizeof(entry_text);

		char list_upc[20];
		printf("ENTRY TEXT IS %s", entry_text);
		//DEBUG
		printf("REMOVE ITEM MODE IS %i\n",remove_item_mode);
		if (remove_item_mode == 1 && item_count > 0)
		{
				item_count--; // use this to make array shifting easier
		//DEBUG
				printf("item count is %i\n",item_count);
				while (r < sizeof(session_list)/sizeof(session_list[r])) //gives length of row, rho, roe
				{
						while (c < sizeof(session_list[r]))
						{
								if (session_list[r][c] == '\t')  //WE ONLY WANT THE UPC HERE!!!! STOP AT A
TAB!!!
								{    //   printf("BREAKING \n");
										c++;
										list_upc[i]='\0';
										break;
								}
								list_upc[i] = session_list[r][c];
								c++;
								i++;
						}

						if (strcmp(list_upc, entry_text) == 0)
						{
								printf("MATCH!!!!\n"); //DEBUG
//              total = total - priceLookup(entry_text);
//              text_view_update(entry);
                while(r < sizeof(session_list)/sizeof(session_list[r]))
                {
                    strcpy(session_list[r], session_list[r+1]);
                    r++;
                }
                total = total - priceLookup(entry_text);
                text_view_update(entry);
                break;
            }
            r++;
            c = 0;
            i = 0;
        }

        remove_item_mode = 0;
//DEUG
        printf("ater while loop : %s \n ", list_upc);
    }
    else
    {
    //  strcpy(session_list[item_count], entry_text);
        item_count++;
        total = total + priceLookup(entry_text);
        text_view_update(entry);
        //DEBUG
        printf("item count is %i\n entry item is %s\n total is %3.2f\n",item_count,session_list[item_count], total);
        r++;  // goddamnit
    }

    printf("TOTAL IS %3.2f\n", total);
    sprintf(temp_string, "%3.2f", total);
    GtkWidget *total_display = lookup_widget(GTK_WIDGET(entry), "total");
    gtk_entry_set_text(GTK_ENTRY(total_display), temp_string ); //refresh the total display frame

//  price = priceLookup(entry_text);
//DEBUG

}
```

Interface.c

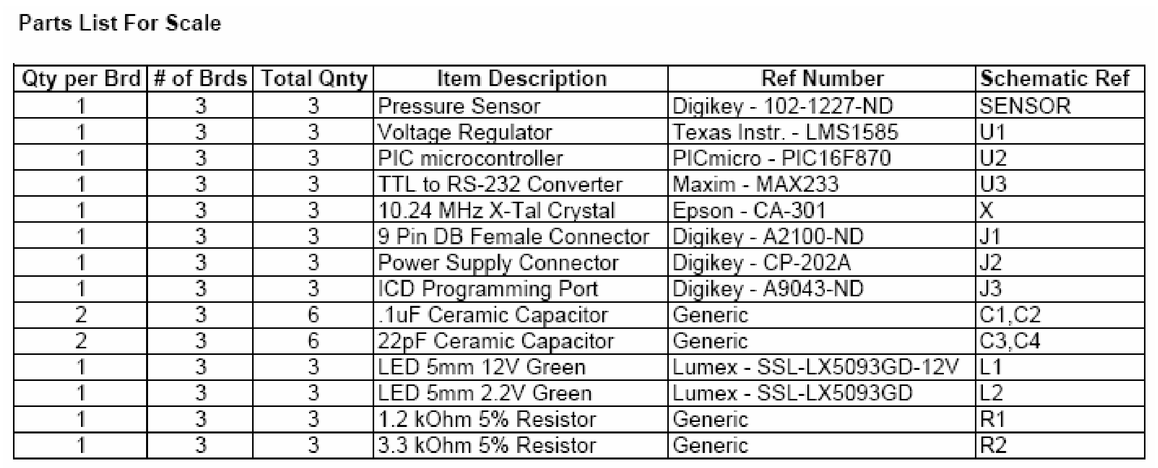
```
/*
 * interface!!!
 */

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>

#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>

#include "callbacks.h"
#include "interface.h"
#include "support.h"

#define GLADE_HOOKUP_OBJECT(component,widget,name) \
  g_object_set_data_full (G_OBJECT (component), name, \
    gtk_widget_ref (widget), (GDestroyNotify) gtk_widget_unref)

#define GLADE_HOOKUP_OBJECT_NO_REF(component,widget,name) \
  g_object_set_data (G_OBJECT (component), name, widget)
```

```
GtkWidget*
create_window1 (void)
{
  GtkWidget *window1;
  GtkWidget *vbox1;
  GtkWidget *scrolledwindow2;
  GtkWidget *textview1;
  GtkWidget *hbox1;
  GtkWidget *hbox2;
  GtkWidget *hbuttonbox3;
  GtkWidget *button1;
  GtkWidget *button2;
  GtkWidget *button3;
  GtkWidget *button4;
  GtkWidget *button5;
  GtkWidget *entry1;
  GtkWidget *eventbox1;
  GtkWidget *frame1;
  GtkWidget *weight;
  GtkWidget *label1;
  GtkWidget *frame2;
  GtkWidget *total;
  GtkWidget *label2;

  window1 = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window1), _("window1"));

  vbox1 = gtk_vbox_new (FALSE, 0);
  gtk_widget_show (vbox1);
  gtk_container_add (GTK_CONTAINER (window1), vbox1);

  scrolledwindow2 = gtk_scrolled_window_new (NULL, NULL);
  gtk_widget_show (scrolledwindow2);
  gtk_box_pack_start (GTK_BOX (vbox1), scrolledwindow2, TRUE, TRUE, 0);
  gtk_widget_set_size_request (scrolledwindow2, 622, 357);

  textview1 = gtk_text_view_new ();
  gtk_widget_show (textview1);
  gtk_container_add (GTK_CONTAINER (scrolledwindow2), textview1);
  gtk_widget_set_size_request (textview1, 495, 351);
  gtk_text_view_set_editable (GTK_TEXT_VIEW (textview1), FALSE);
  gtk_text_view_set_justification (GTK_TEXT_VIEW (textview1), GTK_JUSTIFY_FILL);
  gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (textview1), GTK_WRAP_CHAR);
  gtk_text_view_set_cursor_visible (GTK_TEXT_VIEW (textview1), FALSE);

  hbox1 = gtk_hbox_new (FALSE, 0);
  gtk_widget_show (hbox1);
  gtk_box_pack_start (GTK_BOX (vbox1), hbox1, TRUE, TRUE, 0);

  hbox2 = gtk_hbox_new (FALSE, 0);
  gtk_widget_show (hbox2);
  gtk_box_pack_start (GTK_BOX (hbox1), hbox2, TRUE, TRUE, 0);

  hbuttonbox3 = gtk_hbutton_box_new ();
  gtk_widget_show (hbuttonbox3);
  gtk_box_pack_start (GTK_BOX (hbox2), hbuttonbox3, TRUE, TRUE, 0);

  button1 = gtk_button_new_with_mnemonic (_("Start"));
  gtk_widget_show (button1);
  gtk_container_add (GTK_CONTAINER (hbuttonbox3), button1);
  GTK_WIDGET_SET_FLAGS (button1, GTK_CAN_DEFAULT);

  button2 = gtk_button_new_with_mnemonic (_("Checkout"));
  gtk_widget_show (button2);
  gtk_container_add (GTK_CONTAINER (hbuttonbox3), button2);
  GTK_WIDGET_SET_FLAGS (button2, GTK_CAN_DEFAULT);

  button3 = gtk_button_new_with_mnemonic (_("Check Weight"));
  gtk_widget_show (button3);
```

```
gtk_container_add (GTK_CONTAINER (hbuttonbox3), button3);
GTK_WIDGET_SET_FLAGS (button3, GTK_CAN_DEFAULT);

button4 = gtk_button_new_with_mnemonic (_("Accept Weight"));
gtk_widget_show (button4);
gtk_container_add (GTK_CONTAINER (hbuttonbox3), button4);
GTK_WIDGET_SET_FLAGS (button4, GTK_CAN_DEFAULT);

button5 = gtk_button_new_with_mnemonic (_("Remove Item"));
gtk_widget_show (button5);
gtk_container_add (GTK_CONTAINER (hbuttonbox3), button5);
GTK_WIDGET_SET_FLAGS (button5, GTK_CAN_DEFAULT);

entry1 = gtk_entry_new ();
gtk_widget_show (entry1);
gtk_box_pack_start (GTK_BOX (hbox2), entry1, TRUE, TRUE, 0);

eventbox1 = gtk_event_box_new ();
gtk_widget_show (eventbox1);
gtk_box_pack_start (GTK_BOX (hbox1), eventbox1, TRUE, TRUE, 0);

frame1 = gtk_frame_new (NULL);
gtk_widget_show (frame1);
gtk_container_add (GTK_CONTAINER (eventbox1), frame1);

weight = gtk_entry_new ();
gtk_widget_show (weight);
gtk_container_add (GTK_CONTAINER (frame1), weight);
gtk_editable_set_editable (GTK_EDITABLE (weight), FALSE);
gtk_entry_set_has_frame (GTK_ENTRY (weight), FALSE);
gtk_entry_set_width_chars (GTK_ENTRY (weight), 4);

label1 = gtk_label_new (_("Weight"));
gtk_widget_show (label1);
gtk_frame_set_label_widget (GTK_FRAME (frame1), label1);

frame2 = gtk_frame_new (NULL);
gtk_widget_show (frame2);
gtk_box_pack_start (GTK_BOX (hbox1), frame2, TRUE, TRUE, 0);

total = gtk_entry_new ();
gtk_widget_show (total);
gtk_container_add (GTK_CONTAINER (frame2), total);
gtk_editable_set_editable (GTK_EDITABLE (total), FALSE);
gtk_entry_set_has_frame (GTK_ENTRY (total), FALSE);
gtk_entry_set_width_chars (GTK_ENTRY (total), 6);

label2 = gtk_label_new (_("Total"));
gtk_widget_show (label2);
gtk_frame_set_label_widget (GTK_FRAME (frame2), label2);

g_signal_connect ((gpointer) button1, "clicked",
                G_CALLBACK (on_start_clicked),
                NULL);
g_signal_connect ((gpointer) button2, "clicked",
                G_CALLBACK (on_checkout_clicked),
                NULL);
g_signal_connect ((gpointer) button3, "activate",
                G_CALLBACK (on_checkweight_clicked),
                NULL);
g_signal_connect ((gpointer) button4, "clicked",
                G_CALLBACK (on_accept_clicked),
                NULL);
g_signal_connect ((gpointer) button5, "clicked",
                G_CALLBACK (on_remove_clicked),
                NULL);
g_signal_connect ((gpointer) entry1, "activate",
                G_CALLBACK (on_UPCentry_activate),
                NULL);
```

```
  /* Store pointers to all widgets, for use by lookup_widget(). */
  GLADE_HOOKUP_OBJECT_NO_REF (window1, window1, "window1");
  GLADE_HOOKUP_OBJECT (window1, vbox1, "vbox1");
  GLADE_HOOKUP_OBJECT (window1, scrolledwindow2, "scrolledwindow2");
  GLADE_HOOKUP_OBJECT (window1, textview1, "textview1");
  GLADE_HOOKUP_OBJECT (window1, hbox1, "hbox1");
  GLADE_HOOKUP_OBJECT (window1, hbox2, "hbox2");
  GLADE_HOOKUP_OBJECT (window1, hbuttonbox3, "hbuttonbox3");
  GLADE_HOOKUP_OBJECT (window1, button1, "button1");
  GLADE_HOOKUP_OBJECT (window1, button2, "button2");
  GLADE_HOOKUP_OBJECT (window1, button3, "button3");
  GLADE_HOOKUP_OBJECT (window1, button4, "button4");
  GLADE_HOOKUP_OBJECT (window1, button5, "button5");
  GLADE_HOOKUP_OBJECT (window1, entry1, "entry1");
  GLADE_HOOKUP_OBJECT (window1, eventbox1, "eventbox1");
  GLADE_HOOKUP_OBJECT (window1, frame1, "frame1");
  GLADE_HOOKUP_OBJECT (window1, weight, "weight");
  GLADE_HOOKUP_OBJECT (window1, label1, "label1");
  GLADE_HOOKUP_OBJECT (window1, frame2, "frame2");
  GLADE_HOOKUP_OBJECT (window1, total, "total");
  GLADE_HOOKUP_OBJECT (window1, label2, "label2");

  return window1;
}
```

Appendix IV



Parts List For Scale

Qty per Brd	# of Brds	Total Qnty	Item Description	Ref Number	Schematic Ref
1	3	3	Pressure Sensor	Digikey - 102-1227-ND	SENSOR
1	3	3	Voltage Regulator	Texas Instr. - LMS1585	U1
1	3	3	PIC microcontroller	PICmicro - PIC16F870	U2
1	3	3	TTL to RS-232 Converter	Maxim - MAX233	U3
1	3	3	10.24 MHz X-Tal Crystal	Epson - CA-301	X
1	3	3	9 Pin DB Female Connector	Digikey - A2100-ND	J1
1	3	3	Power Supply Connector	Digikey - CP-202A	J2
1	3	3	ICD Programming Port	Digikey - A9043-ND	J3
2	3	6	.1uF Ceramic Capacitor	Generic	C1,C2
2	3	6	22pF Ceramic Capacitor	Generic	C3,C4
1	3	3	LED 5mm 12V Green	Lumex - SSL-LX5093GD-12V	L1
1	3	3	LED 5mm 2.2V Green	Lumex - SSL-LX5093GD	L2
1	3	3	1.2 kOhm 5% Resistor	Generic	R1
1	3	3	3.3 kOhm 5% Resistor	Generic	R2

```
//Source code for BOSS Project Digital Scale

#include<16f870.h>
#fuses HS,NOWDT,PUT,NOPROTECT,NODEBUG,NOBROWNOUT,NOLVP,NOCPD,NOWRT
#use delay(clock=10240000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
void main()
{
int i,min,max,value;
float sum,avg;
printf("Sampling:");
setup_port_a( ALL_ANALOG);
setup_adc( ADC_CLOCK_INTERNAL);
set_adc_channel(0);
do{
min=255;
max=0;
value=0;
sum=0.0;
avg=0.0;
for(i=1;i<=10;++i){
delay_ms(100);
value = Read_ADC();
sum = sum + (float)value;
if(value<min)
min = value;
if(value>max)
max = value;
}
avg = sum/10.0;
printf("\fMIN:%u MAX:%u AVG:%u\r",min,max,(int)avg);
}while(TRUE);
}
```

Source code for BOSS Project Digital Scale

```
//////// Standard Header file for the PIC16F870 device ////////////////
#device PIC16F870
#nolist
//////// Program memory: 2048x14 Data RAM: 128 Stack: 8
```

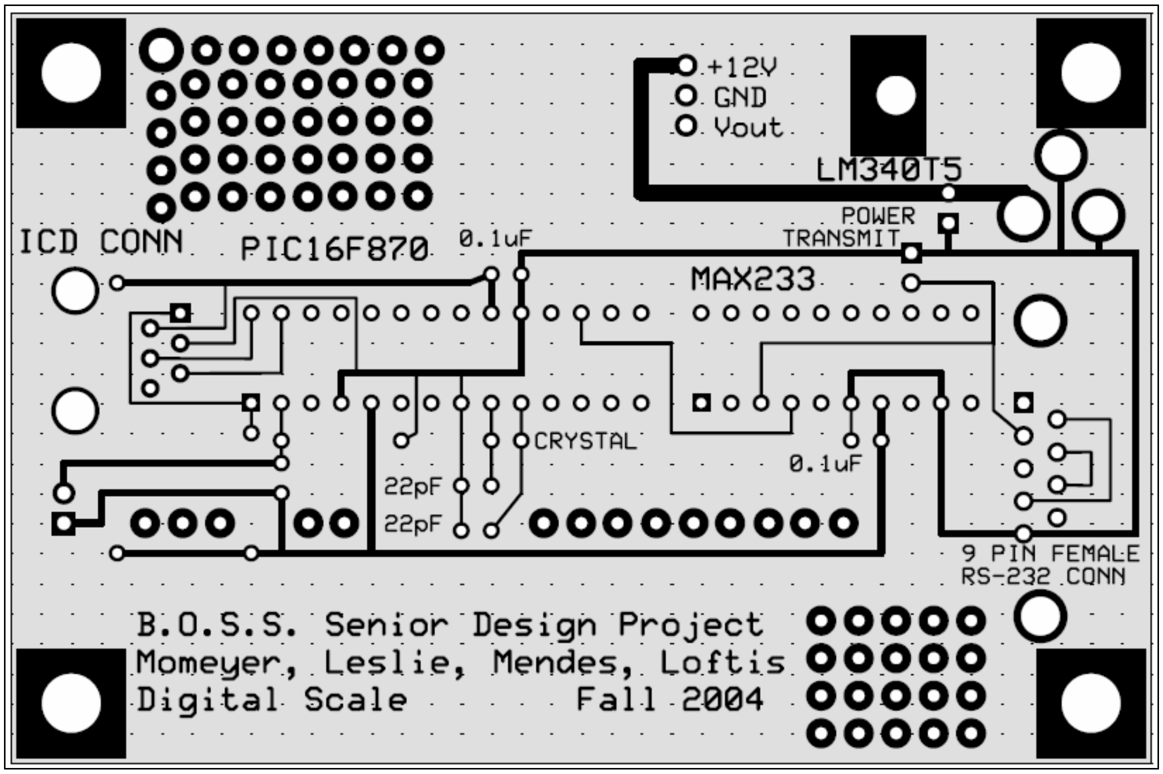
```
//////// I/O: 22 Analog Pins: 5
//////// Data EEPROM: 64
//////// C Scratch area: 20 ID Location: 2000
//////// Fuses: LP,XT,HS,RC,NOWDT,WDT,NOPUT,PUT,PROTECT,NOPROTECT,DEBUG
//////// Fuses: NODEBUG,NOBROWNOUT,BROWNOUT,LVP,NOLVP,CPD,NOCPD,WRT,NOWRT
////////
/////////////////////////////////////////////////////////////////// I/O
// Discrete I/O Functions: SET_TRIS_x(), OUTPUT_x(), INPUT_x(),
// PORT_B_PULLUPS(), INPUT(),
// OUTPUT_LOW(), OUTPUT_HIGH(),
// OUTPUT_FLOAT(), OUTPUT_BIT()
// Constants used to identify pins in the above are:
#define PIN_A0 40
#define PIN_A1 41
#define PIN_A2 42
#define PIN_A3 43
#define PIN_A4 44
#define PIN_A5 45
#define PIN_B0 48
#define PIN_B1 49
#define PIN_B2 50
#define PIN_B3 51
#define PIN_B4 52
#define PIN_B5 53
#define PIN_B6 54
#define PIN_B7 55
#define PIN_C0 56
#define PIN_C1 57
#define PIN_C2 58
#define PIN_C3 59
#define PIN_C4 60
#define PIN_C5 61
#define PIN_C6 62
#define PIN_C7 63
/////////////////////////////////////////////////////////////////// Useful
defines
#define FALSE 0
#define TRUE 1
#define BYTE int
#define BOOLEAN short int
#define getc getch
#define fgetc getch
#define getchar getch
#define putc putchar
#define fputc putchar
#define fgets gets
#define fputs puts
/////////////////////////////////////////////////////////////////// Control
// Control Functions: RESET_CPU(), SLEEP(), RESTART_CAUSE()
// Constants returned from RESTART_CAUSE() are:
#define WDT_FROM_SLEEP 0
#define WDT_TIMEOUT 8
#define MCLR_FROM_SLEEP 16
#define NORMAL_POWER_UP 24
/////////////////////////////////////////////////////////////////// Timer 0
// Timer 0 (AKA RTCC)Functions: SETUP_COUNTERS() or SETUP_TIMER0(),
// SET_TIMER0() or SET_RTCC(),
// GET_TIMER0() or GET_RTCC()
// Constants used for SETUP_TIMER0() are:
#define RTCC_INTERNAL 0
#define RTCC_EXT_L_TO_H 32
#define RTCC_EXT_H_TO_L 48
#define RTCC_DIV_1 8
1
#define RTCC_DIV_2 0
#define RTCC_DIV_4 1
#define RTCC_DIV_8 2
#define RTCC_DIV_16 3
```

```
#define RTCC_DIV_32 4
#define RTCC_DIV_64 5
#define RTCC_DIV_128 6
#define RTCC_DIV_256 7
#define RTCC_8_BIT 0
// Constants used for SETUP_COUNTERS() are the above
// constants for the 1st param and the following for
// the 2nd param:
////////////////////////////////////////////////////////////////// WDT
// Watch Dog Timer Functions: SETUP_WDT() or SETUP_COUNTERS() (see above)
// RESTART_WDT()
//
#define WDT_18MS 8
#define WDT_36MS 9
#define WDT_72MS 10
#define WDT_144MS 11
#define WDT_288MS 12
#define WDT_576MS 13
#define WDT_1152MS 14
#define WDT_2304MS 15
////////////////////////////////////////////////////////////////// Timer 1
// Timer 1 Functions: SETUP_TIMER_1, GET_TIMER1, SET_TIMER1
// Constants used for SETUP_TIMER_1() are:
// (or (via |) together constants from each group)
#define T1_DISABLED 0
#define T1_INTERNAL 0x85
#define T1_EXTERNAL 0x87
#define T1_EXTERNAL_SYNC 0x83
#define T1_CLK_OUT 8
#define T1_DIV_BY_1 0
#define T1_DIV_BY_2 0x10
#define T1_DIV_BY_4 0x20
#define T1_DIV_BY_8 0x30
////////////////////////////////////////////////////////////////// Timer 2
// Timer 2 Functions: SETUP_TIMER_2, GET_TIMER2, SET_TIMER2
// Constants used for SETUP_TIMER_2() are:
#define T2_DISABLED 0
#define T2_DIV_BY_1 4
#define T2_DIV_BY_4 5
#define T2_DIV_BY_16 6
////////////////////////////////////////////////////////////////// CCP
// CCP Functions: SETUP_CCPx, SET_PWMx_DUTY
// CCP Variables: CCP_x, CCP_x_LOW, CCP_x_HIGH
// Constants used for SETUP_CCPx() are:
#define CCP_OFF 0
#define CCP_CAPTURE_FE 4
#define CCP_CAPTURE_RE 5
#define CCP_CAPTURE_DIV_4 6
#define CCP_CAPTURE_DIV_16 7
#define CCP_COMPARE_SET_ON_MATCH 8
#define CCP_COMPARE_CLR_ON_MATCH 9
#define CCP_COMPARE_INT 0xA
#define CCP_COMPARE_RESET_TIMER 0xB
#define CCP_PWM 0xC
#define CCP_PWM_PLUS_1 0x1c
#define CCP_PWM_PLUS_2 0x2c
#define CCP_PWM_PLUS_3 0x3c
long CCP_1;
#byte CCP_1 = 0x15
#byte CCP_1_LOW= 0x15
#byte CCP_1_HIGH= 0x16
////////////////////////////////////////////////////////////////// PSP
// PSP Functions: SETUP_PSP, PSP_INPUT_FULL(), PSP_OUTPUT_FULL(),
2
// PSP_OVERFLOW(), INPUT_D(), OUTPUT_D()
// PSP Variables: PSP_DATA
// Constants used in SETUP_PSP() are:
#define PSP_ENABLED 0x10
```

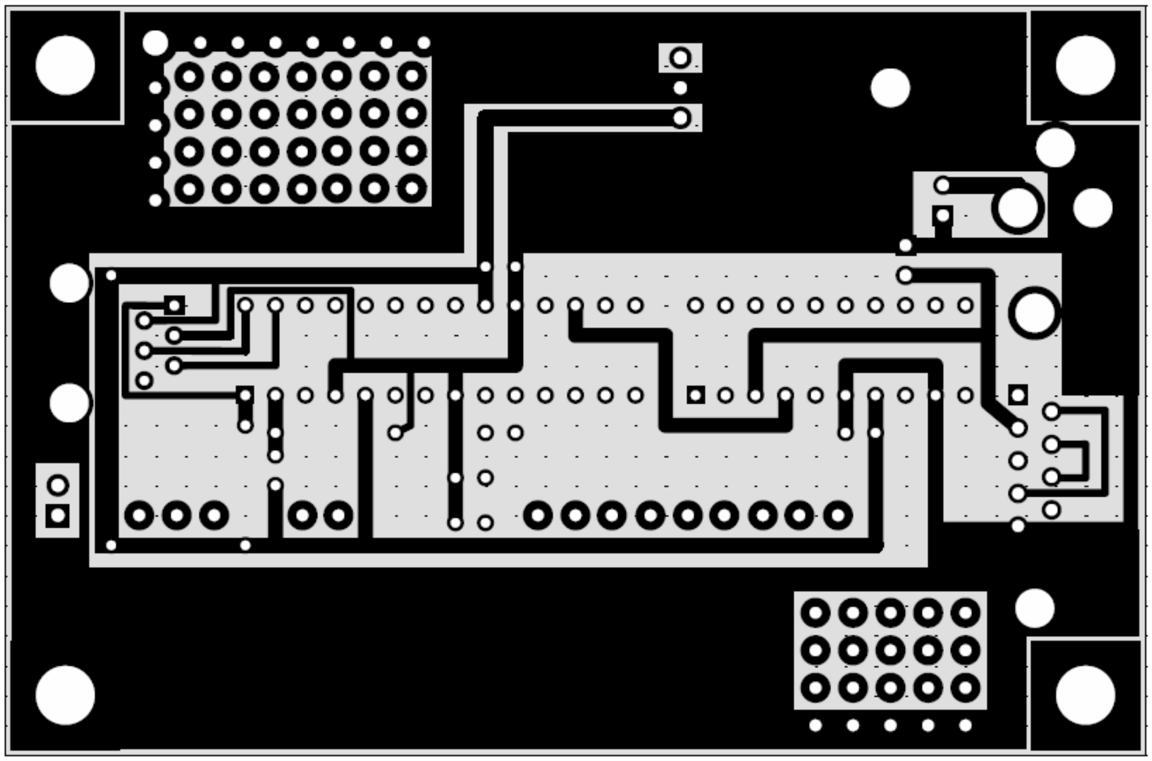
```
#define PSP_DISABLED 0
#byte PSP_DATA= 8
////////////////////////////////////////////////////////////////// SPI
// SPI Functions: SETUP_SPI, SPI_WRITE, SPI_READ, SPI_DATA_IN
// Constants used in SETUP_SSP() are:
#define SPI_MASTER 0x20
#define SPI_SLAVE 0x24
#define SPI_L_TO_H 0
#define SPI_H_TO_L 0x10
#define SPI_CLK_DIV_4 0
#define SPI_CLK_DIV_16 1
#define SPI_CLK_DIV_64 2
#define SPI_CLK_T2 3
#define SPI_SS_DISABLED 1
////////////////////////////////////////////////////////////////// UART
// Constants used in setup_uart() are:
// FALSE - Turn UART off
// TRUE - Turn UART on
#define UART_ADDRESS 2
#define UART_DATA 4
// TRUE - Turn UART on
////////////////////////////////////////////////////////////////// ADC
// ADC Functions: SETUP_ADC(), SETUP_ADC_PORTS() (aka SETUP_PORT_A),
// SET_ADC_CHANNEL(), READ_ADC()
// Constants used in SETUP_ADC_PORTS() are:
#define NO_ANALOGS 0x86 // None
#define ALL_ANALOG 0x80 // A0 A1 A2 A3 A5 E0 E1 E2 Ref=Vdd
#define ANALOG_RA3_REF 0x81 // A0 A1 A2 A5 E0 E1 E2 Ref=A3
#define A_ANALOG 0x82 // A0 A1 A2 A3 A5 Ref=Vdd
#define A_ANALOG_RA3_REF 0x83 // A0 A1 A2 A5 Ref=A3
#define RA0_RA1_RA3_ANALOG 0x84 // A0 A1 A3 Ref=Vdd
#define RA0_RA1_ANALOG_RA3_REF 0x85 // A0 A1 Ref=A3
#define ANALOG_RA3_RA2_REF 0x88 // A0 A1 A5 E0 E1 E2 Ref=A2,A3
#define ANALOG_NOT_RE1_RE2 0x89 // A0 A1 A2 A3 A5 E0 Ref=Vdd
#define ANALOG_NOT_RE1_RE2_REF_RA3 0x8A // A0 A1 A2 A5 E0 Ref=A3
#define ANALOG_NOT_RE1_RE2_REF_RA3_RA2 0x8B // A0 A1 A5 E0 Ref=A2,A3
#define A_ANALOG_RA3_RA2_REF 0x8C // A0 A1 A5 Ref=A2,A3
#define RA0_RA1_ANALOG_RA3_RA2_REF 0x8D // A0 A1 Ref=A2,A3
#define RA0_ANALOG 0x8E // A0
#define RA0_ANALOG_RA3_RA2_REF 0x8F // A0 Ref=A2,A3
// Constants used for SETUP_ADC() are:
#define ADC_OFF 0 // ADC Off
#define ADC_CLOCK_DIV_2 1
#define ADC_CLOCK_DIV_8 0x41
#define ADC_CLOCK_DIV_32 0x81
#define ADC_CLOCK_INTERNAL 0xc1 // Internal 2-6us
// Constants used in READ_ADC() are:
#define ADC_START_AND_READ 7 // This is the default if nothing is specified
#define ADC_START_ONLY 1
#define ADC_READ_ONLY 6
////////////////////////////////////////////////////////////////// INT
// Interrupt Functions: ENABLE_INTERRUPTS(), DISABLE_INTERRUPTS(),
// EXT_INT_EDGE()
//
// Constants used in EXT_INT_EDGE() are:
#define L_TO_H 0x40
#define H_TO_L 0
// Constants used in ENABLE/DISABLE_INTERRUPTS() are:
#define GLOBAL 0x0BC0
#define INT_RTCC 0x0B20
#define INT_RB 0x0B08
#define INT_EXT 0x0B10
#define INT_AD 0x8C40
#define INT_TBE 0x8C10
#define INT_RDA 0x8C20
#define INT_TIMER1 0x8C01
#define INT_TIMER2 0x8C02
3
```

```
#define INT_CCP1 0x8C04
#define INT_SSP 0x8C08
#define INT_PSP 0x8C80
#define INT_BUSCOL 0x8D08
#define INT_EEPROM 0x8D10
#define INT_TIMER0 0x0B20
#list
```

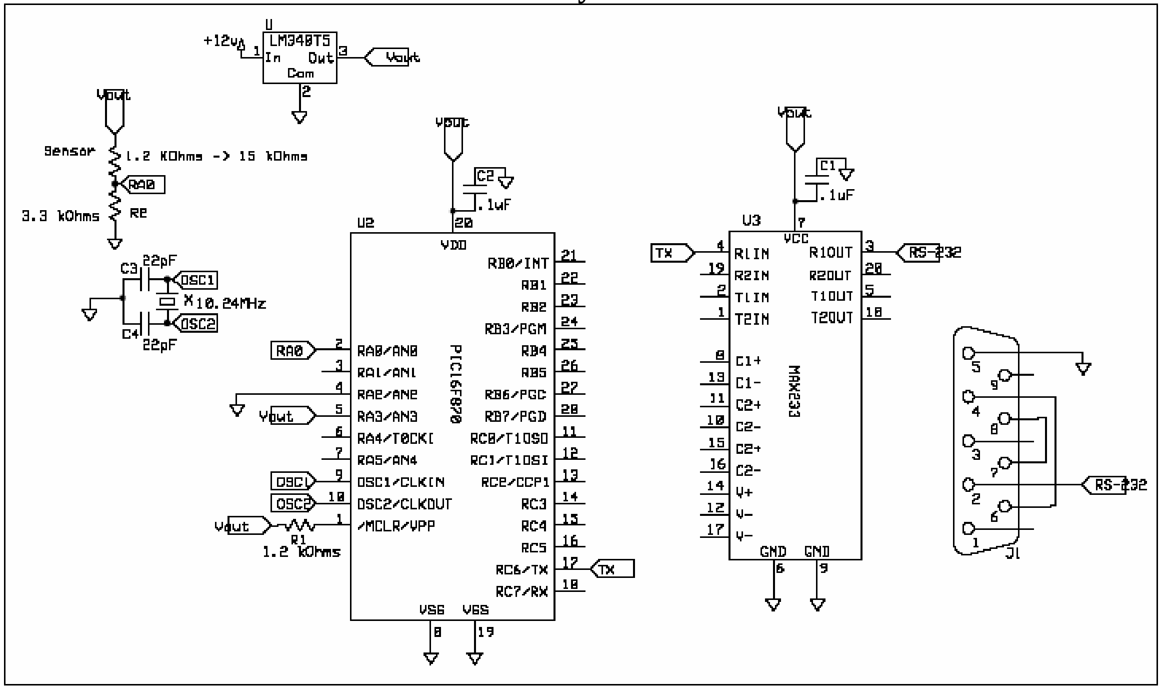
Standard Header file for the PIC16F870 device



Top Copper Layer of PCB



Bottom Layer of PCB



PCB Layout Schematic