

Conestoga College

3rd Year Project Report

E Z - C A \$ H: Feasibility Project

<http://www.zenwerx.com/ezcash>

Authors:

Bojan Korousic and Sean Carpenter

Prepared For:

George Bulik, Valdis Cers, Bob Coons, Rudy Hofer, Leslie McLaughlin

April, 2003

Table of Contents

I.	Introduction	3
II.	Background Research	4
III.	Hardware	5
	A. Phone/Modem/Cable	5
	B. Communication	6
	C. MC68HC11 Project Board	7
IV.	Software	7
	A. i) Checking For New Messages	7
	ii) Receiving Message From The Phone	8
	B. Decoding The Message	9
	C. Deleting The Message	9
	D. Database Program	10
	E. Transmitting To The Micro	10
	F. The Micro Program	11
V.	The Vending Machine	11
VI.	Final Results	12
VII.	Conclusions	13
VIII.	Bibliography	14
IX.	Appendices	15
X.	Computer Disk inside front cover	

EZ-CASH is an end to end wireless payment solution for purchasing products or services using a cell phone

Sean Carpenter Year III, Telecom Engineering, Bojan Korousic Year III, Telecom Engineering

Abstract - The feasibility project "EZ-CASH" emulates a service which allows the user to purchase desired item from the vending machine using his/her cell phone. The Consumer does not need to have cash to make purchases and can avoid hassling with change. One day this technology may replace cash all together.

I. INTRODUCTION

This project started as initial idea from what was already done in Europe using cell phone's infrared port and vending machine. However, our approach was similar but different, in the way that it was more secure and it was available to broad number of users. The main advantage was that all cell phones ,3 years old and up, operated on digital networks, such as GSM, TDMA, or CDMA. This would imply that nearly all these mobile phones are able to send and receive short messages(known as SMS MT: Mobile Terminate) which is a key component , and must be available on the consumers mobile phone in order for our project to be

effective. The Basic idea of how this project works is as follows:

The user is required to send a text message, of SMS type, which will contain the code of the item they would like to purchase. The user will read the "code" of the desired item from the vending machine. Once the message is typed in, the user has to send the SMS text message to a phone number specified on the vending machine. A built in GMS modem/phone is located within our mock vending machine which receives the users message. When a new message is received it is decoded by our main program which is constantly running in the background. The decoded message is then stored in a database along with the price of the item. The next step is to "release" the item to the user. This is done by lighting up an LED corresponding to the users selection. This simulates the release mechanism of real vending machine. See Fig.1.

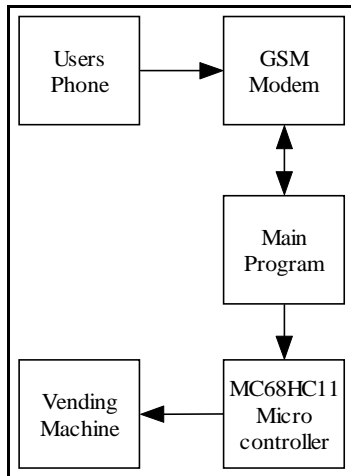


Fig.1. – Basic project flowchart

II. BACKGROUND RESEARCH

One of the main concerns that was brought up was would this project be feasible in the real world, would the consumer actually use it if it were available? To prove the concept a random survey of students and professionals in the age range of 15 to 50 was conducted, this was felt to be the target market. [See Appendix 1 - Survey Results]. Of the people surveyed ,74.3% use their cell phone on a daily basis. Of the 74.3% who use their cell phone , 78% said that they would purchase a product from a vending machine using their cell phone. In general, this survey gave a positive indication that people would use this service, so the decision to proceed with the project was made.

Other main topics of concern relating to project design and constraints were the operation of GSM modems and SMS hardware; Interfacing terminal equipment

with a GSM modem which is defined within the GSM 07:05 and 07:07 standards; The Short Message data structure known as a Protocol Data Unit(PDU) and finally; Familiarity with the micro code and architecture of the MC68 HC11 microcontroller project board.

At the beginning of the research phase it was known that the cell phone was being used as a wireless modem, and to communicate with modems in general it is required to use an AT command set. But since a standard set doesn't support all the commands needed to communicate with a cell phone, an extended set of commands was introduced by Nokia. Specifically the SMS command subset, which is needed to read, list, store and delete messages. [See Appendix 2]. During the next research phase, it was found that only GSM phones with a built in hardware modem will support the extended set of AT commands. This was an important step since at the beginning of the of the research phase a TDMA phone was being used to test these commands, and it wasn't responding. This is because the SMS subset is defined by GSM standards, therefore only phones compliant with these standards will support them. Once a GSM phone was acquired communication was established and the retrieval of SMS messages from the phone successful. The next step was to separate and decode these messages, which are stored in PDU format. See Fig.2.

```

07917150979603F0 04 0B916174820790F0 00 00 30209071606400 03 C1180C

07917150979603F0 - SCA, Network operators service centre number
04 - ID, Identification flag
0B916174820790F0 - OA, Senders phone number
00 - PID, Protocol identifier
00 - DCS, Data coding scheme
30209071606400 - SCTS, Date and time the message was sent
03 - UDL, User data length
C1180C - UD, Compressed user data

```

Fig.2. – SMS message in PDU format

Once the message was separated into its specific parts it was noted that for project purposes only the OA, SCTS, UDL and UD were needed, the rest was irrelevant. To decode these parts of the message familiarity with the PDU format was required. Finally basic knowledge of micro coding was known, but specific knowledge of the MC68HC11 commands and architecture was not. Therefore research of the commands and architecture of the chip and project board were needed. This was one of the simpler tasks since Conestoga College teaches various courses on this microcontroller and many resources were available.

III. Hardware

There are a variety of different terminals and means to receive a short message from a mobile phone, this is because GSM standards committees did not standardize the mobile phone interfaces. Therefore a wide range of terminals with a variety of interfaces that support SMS are available.

Several options and considerations were taken into account when choosing the hardware for this project.

A. Phone/Modem/Cable

Three main specifications were mandatory when choosing a phone. The first was that it had to support the GSM 07:05 and 07:07 standards as previously mentioned. Secondly it needed to have a built in hardware modem to be able to receive SMS messages. And lastly it needed to have serial connectivity to interface with a standard Personal Computer RS-232 port. When a search was conducted for a cell phone with these capabilities there were several models available. All of which were in the price range of \$300-\$650. Contact was made with Rogers AT&T Wireless who agreed to donate a Nokia 6310i GSM cell phone, which has all the features mentioned above. See Fig.3. This cell phone is one of the most important pieces of this project, since without a GSM cell phone this project would not be possible.



Fig.3. – Nokia 6310i to find a cable that matched the compressed serial output of the Nokia 6310i cell phone. The cable chosen was the Nokia DLR-3P serial cable, which has a

The next piece of hardware is the serial data cable. It is used to transfer data to and from the cell phone. There are a variety of data cables and connectors on the market. It was necessary

proprietary connector on one end and an RS-232 connector on the other. See Fig.3. This cable was able to interface directly with compressed serial output of the phone. See Fig.4.

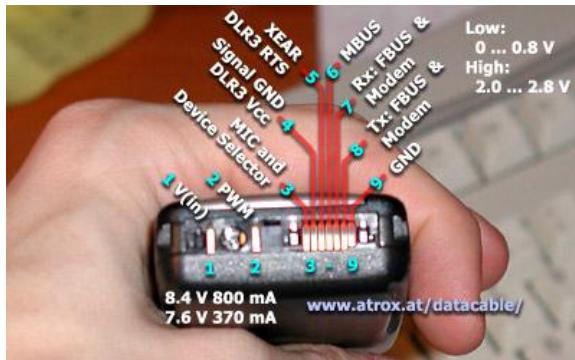


Fig.4. - Connector Layout on 6310i Nokia

B. Communication

Once the phone and data cable were selected the next step was to try and communicate with it through the use of AT commands. It was assumed that since the phone had an RS-232 interface it would be possible to connect it straight to the PC's comport and test its response to these commands. This was not the case. Further testing was done and it was noticed that the output of the cell phone was TTL levels (0-5V). It makes sense that connecting the phone directly to the comport wouldn't work since standard RS-232 levels are $\pm 25V$. Therefore it was necessary to convert the TTL levels to RS-232 levels when sending data to the PC and to convert RS-232 levels to TTL levels when sending data to the phone. This was accomplished by using the DLR-3P serial cable and a straight through serial cable both in NULL modem

configurations with a MAX232 level converter circuit in between. A NULL modem configuration is when the Data Terminal Ready (DTR) and Data Set Ready (DSR) are connected together and the Request To Send (RTS) and Clear To Send (CTS) are connected together. See Fig.5.

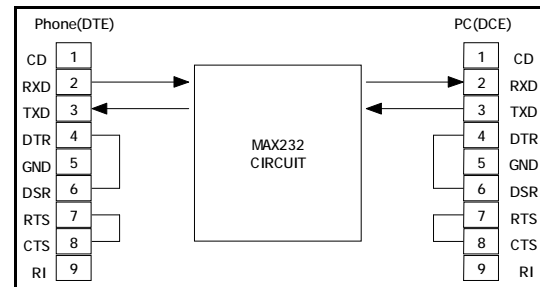


Fig.5. – NULL Modem Setup

The MAX232 is a dual transmitter/receiver that is capable of supplying RS-232 voltage levels from a 5 volt TTL input or provide TTL levels from an RS-232 input. This chip alone did not enabled communication between the phone and the PC. Problems occurred since in the RS-232 standard, the signal levels are negative. For example, when the data from the PC's serial port gets converted to TTL levels it is still negative, therefore the data is not recognized by the phone. Once this problem was realized a 74LS04 inverter was added to the circuit to invert the data to its proper levels. [See Appendix 3 - schematic]. With the communication hardware finished it was then possible to establish a link between the PC and the phone. This was tested by using a simple terminal emulator program called PROCOMM. Which in chat mode enables AT commands to be sent to the phone and receive its responses.

C. MC68HC11 Project Board

Choosing the right Microcontroller Project Platform was an important decision. The project design required a board which has at least one built in RS-232 9-pin serial connection, an LCD screen, and a minimum of 8 outputs. Three boards were taken in to consideration and they were: AMD SD186ES, PIC 16F84 Development board and MC68HC11. Firstly, the PIC 16F84 was considered since of its low cost , small instruction set and familiarity from previous studies. However, it was not powerful enough for this application. Secondly, AMD SD186ES was researched and it proved to have the best qualities for project implementation. Due to the high cost of the board and unfamiliarity with its instruction set, it proved not to be the best choice. Lastly, the MC68HC11 met all the desired requirements and more. In addition to the core microprocessor it includes a serial communications interface which supports RS-232 data, elaborate timers, LCD driver and display, 32k bytes of EPROM and 8 bit input/output registers. It is also supported by an excellent reference manual and data book for each of the basic devices which made it very easy to use. And most importantly it was free.

IV. Software

Software was the main and most important part of the project. This is because design of this project required it to be end-to-end automated system with little or no user assistance. To achieve this, two major programs were needed. One, whose main function is to continuously communicate with the phone and the other to control the mock vending machine. [See Appendix 4 – Flow Charts & Code]. Originally , it was thought that both of these functions would be programmed using micro code. However, as project design progressed it made more sense to use Borland C++ to communicate with the phone and assembly language to control the mock vending machine. Familiarity with the C++ and assembly programming languages from the previous years of study was of great help.

A. i) Checking For New Messages

When the user sends a message containing the desired item to number displayed on the vending machine it is received and stored in the phones memory. To gain access to these messages it was necessary to send AT+CMGR=0, which is a specific AT command to read a new message. To test the response of this command a terminal program called Procomm was used in chat mode. The line settings used to communicate with the phone were 9600 baud, no parity, 8 data bits, 1 stop bit via comport 1. The response obtained from

this command has four parts: the echoed command, memory location, the message in PDU format and the conformation OK. See

Fig.6.

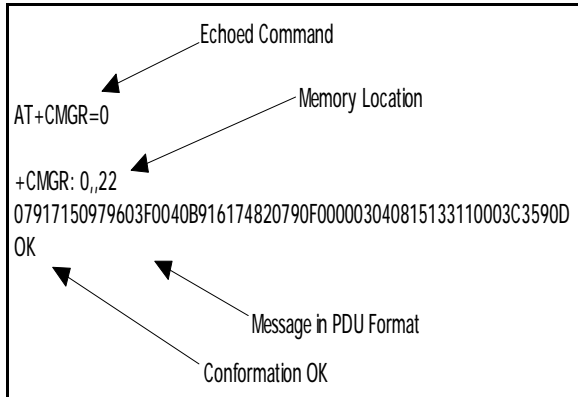


Fig.6. – AT command and phone response

Knowing how this worked, a program to automate the process of checking the phone for new messages was written. First of all the line settings of the comport and a buffer to temporarily store the message had to be set up. Once that was completed the actual AT command had to be transmitted. This was accomplished by sending one character of the command at a time followed by the ASCII equivalent of a carriage return (CR) stating that it was the end. This was executed every 0.5 seconds. It was necessary to have a short delay between the execution of this command due to the phones inability to handle multiple requests. If it was any faster the would phone crash and reset itself.

A. ii) Receiving Message From The Phone

After the command to retrieve new messages was sent to the phone it would

respond with either of two messages. An error message if no new messages were present, See Fig.7.

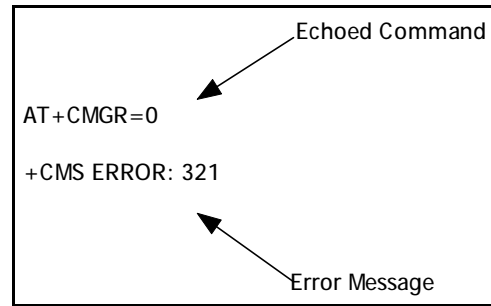


Fig.7. – Error response

or the response shown in Fig.5. When these messages were generated they were stored in the temporary buffer. For the purposes of the program the echoed command was not important, so it was ignored. The program would then read the second line of the response one character at a time. If the 4th character was "S" , it knew there was an error and it would continue to check for new messages. Else it would ignore the second line and start to read the PDU formatted message. As mentioned before this format can be separated into specific parts, only four of which were of interest. They are:

OA, SCTS, UDL, UD. The program would then skip 9 bytes (SCA & ID), store the 8 bytes of the OA, skip the next 2 bytes (PID & DCS), store the 7 bytes of the SCTS, store the 1 byte of the UDL and finally store the 3 bytes of the UD. Now that the important parts of the messages were stored in memory they could be decoded.

B. Decoding The Message

As mentioned before the message was stored in PDU format, thus making decoding the message one of the more difficult tasks. This is because the PDU format just looks like a long hexadecimal string, but not all the data is represented in the same manner. For example all of the data up to the UDL is stored as eight bit ASCII characters where the eighth bit is padded with a zero. Then the UDL is stored as a hex value, but represents the decimal number of bytes in the user data. According to ETSI specifications, an SMS message can be up to 140 bytes long. The usual GSM alphabet only requires 7 bits per character, allowing up to 160 characters of packed UD to be sent.

Decoding the OA and SCTS was easy. They were stored as a hex value but in groups of two characters. In order to get the correct values each set of two characters needed to be flipped. See Fig.8. The decoding process was done using C++, refer to appendix 4 functions getOA() and getTime() .

OA - Received Data	SCTS - Received Data
51 91 27 19 55 F3	20 21 52 61 32 43 00
OA - Decoded Data	SCTS - Decoded Data
15 19 72 91 55 3F	02 12 25 16 23 34 00
User's Number	Date and Time
1-519-729-1553	02 12 25 16 23 34
	YR MN DD HH MM SS

Fig.8. – Received/Decoded Data

Decoding the UD proved to be a much more difficult task. It was assumed that the user data would not exceed 3 bytes per message,

since the user can only purchase one item at a time. This was a limitation set by design constraints. First of all, the groups of two characters had to be flipped, then converted from hex to binary equivalent. This had to be done since 7-bit ASCII characters were compressed into 8-bit characters; The least significant bit(LSB) of the first character was shifted into the most significant bit(MSB) of next character. Once converted into string of binary characters , 7-bits were shifted out representing the first character of the message. Once done, it was converted back to hex equivalent. This was done for all characters in this message. See Fig.9. Refer to appendix 4, functions unpack() and chartohex() .

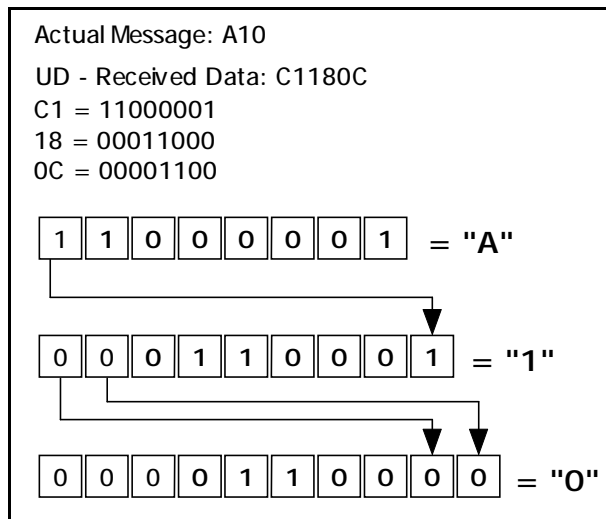


Fig.9. – Decompressed User Data

C. Deleting The Message

There are several reasons why deleting the message immediately after it was read was necessary. First of all, the inbox of Nokia 6310i has a maximum storage capacity of 150 SMS messages. No testing was done to see

what would occur if the phones inbox was at maximum capacity. Therefore , it was decided to delete the message right after it was read in the buffer. Secondly, deleting the message reduces the probability of reading the same message twice. The process of deleting the message is similar to checking for new message, except different AT command was used. AT+CMGD=1, which returns a confirmation response. See Fig.10.

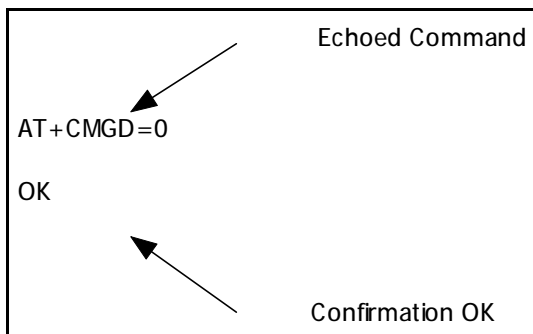


Fig.10. – Delete Command and Response

Any information returned from this command was not significant. So, it was read in the buffer and flushed.

D. Database Program

It was necessary to have a database to keep track of all transactions that took place. The database takes the previously decoded information and stores it in a log file for billing purposes. The way the data is stored is predefined within the program. It stores the users number, date and time the message was sent, the message sent and cost of the item. The cost is determined by comparing the users message to a list of predefined messages. If the users message

matches any of the predefined messages a cost is returned relating to the selected item. If the message does not match any of the predefined messages a cost of zero will be returned.

E. Transmitting To The Micro

To transmit to the microcontroller a few parameters needed to be setup. First, comport2 had to be initialized, since comport1 was already in use. The line settings used were the same as comport1, 9600 baud, no parity, 8 data bits, and 1 stop bit. From the project design it was only necessary to transmit data to the microcontroller, nothing had to be received. Therefore only a transmit buffer had to be setup.

Transmitting data to the microcontroller was not done the same way as transmitting to the phone. The only data that needed to be sent was the three byte user message corresponding to the item selected. There were eight possible messages that could be sent the microcontroller, A05, A10, B15, B20, C25, C30, D35 and D40. Therefore the user data was then compared to these messages. If the data did not match any of these predefined messages then the string " Invalid Choice" was stored in the transmit buffer, where it waited to be read by the microcontroller. If the user data matched any of the predefined messages then an integer number corresponding to message was placed in the buffer followed by the actual message. See Fig.11. The integer number was needed

by the micro program so it could distinguish between each of the messages.

<u>Integer</u>	<u>Message</u>	<u>Buffer</u>
1	A05	1A05
2	A10	2A10
3	B15	3B15
4	B20	4B20
5	C25	5C25
6	C30	6C30
7	D35	7D35
8	D40	8D40

Fig.11. – Integer and corresponding message

F. The Micro Program

The micro code is a simple program that was designed to turn a desired output high and to display messages on the two line LCD screen. It works in such a way that it continuously checks for new messages in the transmit buffer. When a new message is found, it reads the message one character at a time. The first character, which indicates the desired output to turn on, is pushed onto the stack. The rest of the message is then displayed on the second line of the LCD screen. Once this step is complete, the character is pulled from the stack and compared to a list of hex values from 1-8. When the corresponding value is found the program jumps to the subroutine which turns that output high. A delay is then introduced to keep the desired output high for approximately five seconds. After the delay has expired, the LCD screen is cleared and a "Thank You" message is displayed. This message is present for approximately two seconds. Once this is complete, the LCD

is cleared again , allowing for a new message to be displayed, and the output is reset to its original state(low). The program then jumps to the main loop and continues to check for new messages.

This program was loaded into the RAM of the MC68HC11 since it was much easier than taking the chip out every time it needed to be reprogrammed. This saved a considerable amount of time. The program was assembled using a program called Ultra Edit V.32-b and loaded/executed into the microcontrollers RAM through Procomm by the use of the BUFFALO monitor program.

V. The Vending Machine

The initial idea of this project was to purchase a product from a vending machine. Several companies were contacted inquiring about the possibility of obtaining schematics of a vending machine. This was to prove that the project could be easily interfaced to a real vending machine and control the release mechanism. Unfortunately no responses were received after several tries. This led to only one solution, build a mock vending machine that simulates the release of a product purchased by the user.[See Appendix 5] As mentioned previously the microcontroller only had eight outputs, so these outputs were used to light up an LED. The LED turning on simulates the release mechanism of the vending machine when it is activated. Also most vending machines have a LCD screen which informs the user of the product they

purchased. Fortunately a LCD screen was already incorporated into the design of this project and was easily implemented into the vending machine design. Fake items, with the actual code and prices corresponding to the defined values in the database program, were then placed inside. Finally a clear acrylic sheet was placed in front to complete the design. See Fig.12.



Fig. 12. - Mock Vending Machine

VI. Final Results

Now that the project design was finished, it was possible to test the operation of the complete product. Various real life situations were conducted to see how the project would handle them.

Test #1: User Error

The most common error that the system might encounter in the real situation would be user error. There are several possibilities of such an occurrence. For example, a mistyped item CODE, message sent to wrong number, or improper use of phone. The first and probably most common would be if the user typed in the wrong message. This may include the message being mistyped by one or several characters, or if the correct message was sent but extra characters were added by mistake. If the message was mistyped by one or more characters the program receives and decodes the message properly, but the program does not recognize the item code. In this case the users number, the time the message was sent and the item code are stored in the database with a price of \$0.00. Since it does not recognize it as one of the predefined values the string " Invalid Choice" is sent and displayed on the microcontroller. The user will still get charged for the message sent by their service provider. The next possibility might be if the user sends the correct message but accidentally adds extra characters to the end. In this case the main program will only read the first three characters since the predefined user length is three and the rest of the message gets discarded. Therefore the program works properly and decodes the message stores it in the database and turns on the corresponding LED.

Secondly if the user sent the CODE to the wrong number or used the phone improperly then message will never be received by this

system. This can not be corrected and is therefore one limitation of the project.

Test #2: Multiple Messages

Another concern would be what happens when multiple messages are received in the phone's memory. This means that one or more users may be trying to send a message at the same time or due to a lag in the SMS central network messages sent previously maybe be received at a later time. The chances of two users sending a message at the same time is very unlikely to happen, since it is presumed that each user will stand in front of vending machine when sending a message and that the next user will not attempt to send a message until the first user has left. But, there is still a probability that this may happen. Therefore, system was tested under the condition where three different messages from three different users were sent simultaneously. The result was as expected, all messages got processed in the order they were received. This is because the system is designed to read only one message at a time, process it, delete it and read next new message.

Test #3: Power Failure

The last concern was what would happen to the project if there was sudden loss of power. To test this the power bar that the PC and the microcontroller were plugged into was reset. It was inevitable that everything would be shut down, but what

would happen upon restarting the system? It turned out that the microcontroller program started back up with no problems once it was burned into the ROM. The phone program had to be reinitialized but the previous data stored in the database was still there. This is probably the best scenario that could have been hoped for.

VII. Conclusions

SMS messaging is one of the most widely available features on today's cell phones. Because of its ease of use and inexpensive service rates, millions of messages are sent each day. This amount of traffic allows for a huge market of SMS based services. For example: SMS chat, receiving local news and weather reports, downloading phone logos via SMS, and purchasing ring tones. However, not many services that support SMS are available to purchase consumer goods. This is the main reason why the "EZ-CA\$H" project was chosen. It proves that SMS would and could be used by consumers to purchase products. As technology continues to progress and get easier to use more products will become available. Right now people use their cell phones to do everything from surf the internet to organize their day. It is an integral part of their everyday life. So the next logical question is what's next? In the not so distant future it is hoped that this product may pave the way for a world without cash and or credit cards.

VIII. BIBLIOGRAPHY

- [1] Peter D. Hiscocks
68HC11 Microcontroller Construction and Technical Manual
Ryerson Polytechnic Institute, 1998

- [2] Claudio Lanconelli and Alberto Ricci Bitti
Tiny Planet-A One-Bit, Wireless I/O Port
Circuit Cellar, Issue 142, May 2002

- [3] Joan Slavat
Office Supervisor - A control System Based on SMS
Circuit Cellar, Issue 146, September 2002

- [4] Nokia Mobile Phones
Nokia 6090 AT- Command Set And Interfaces Version 1.31
Nokia Mobile Phones, March 27th 2000

- [5] Simon Buckingham
YES 2 SMS
<http://www.yes2sms.com>

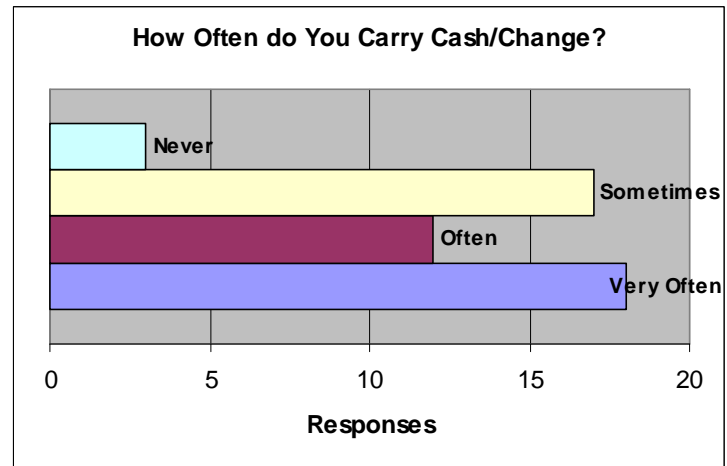
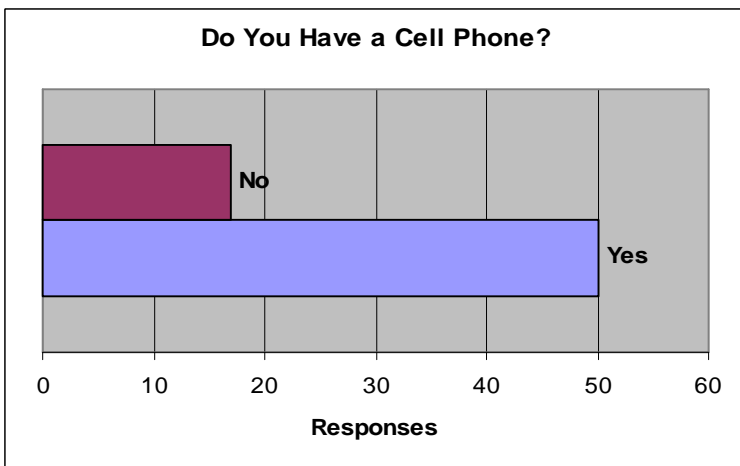
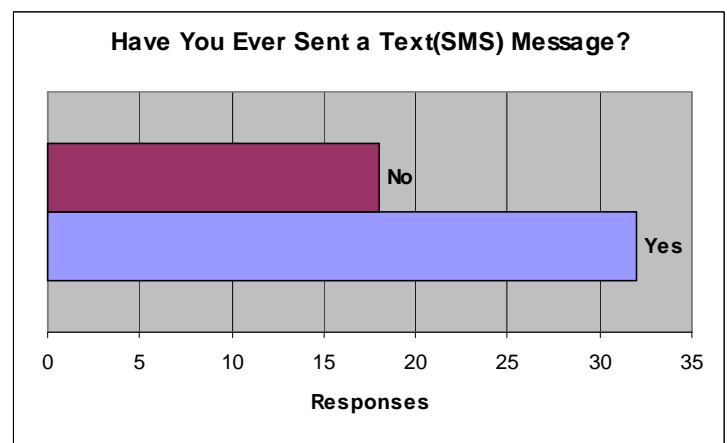
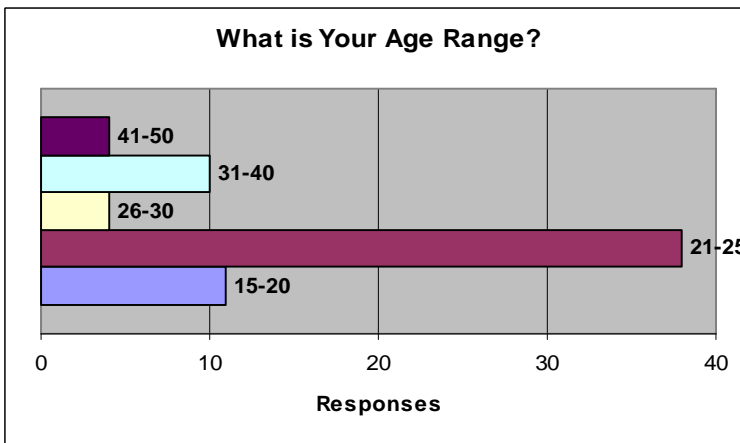
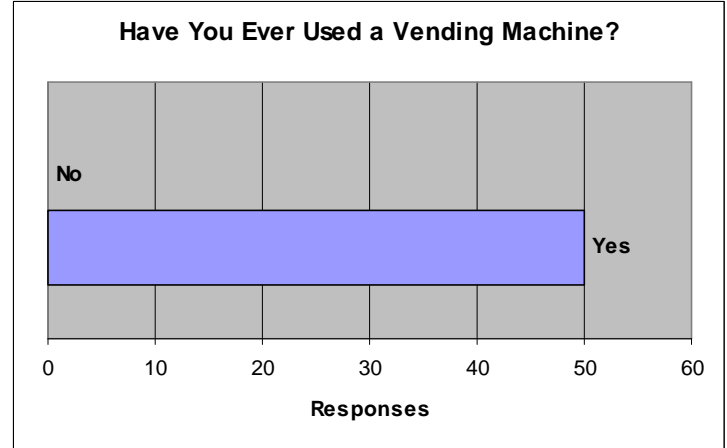
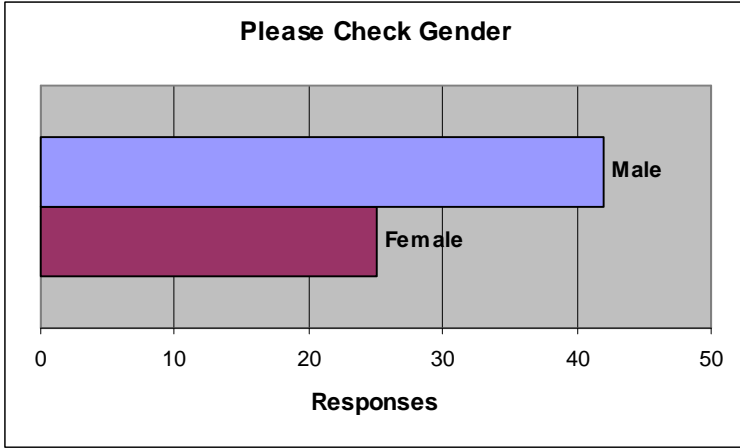
- [6] SMS Development Kit , Version 2.0
<http://junk.republika.pl/mbus/mbus-toc.html>
Date Accessed: September 20th, 2002

- [7] IEEE
IEEE 802 Standards
<http://standards.ieee.org/getieee802/download/802.11b-1999.pdf>
Date Accessed: October 2nd, 2002

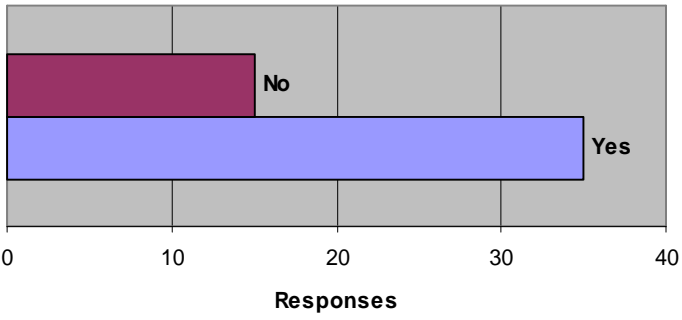
IX. Appendices

Appendix 1

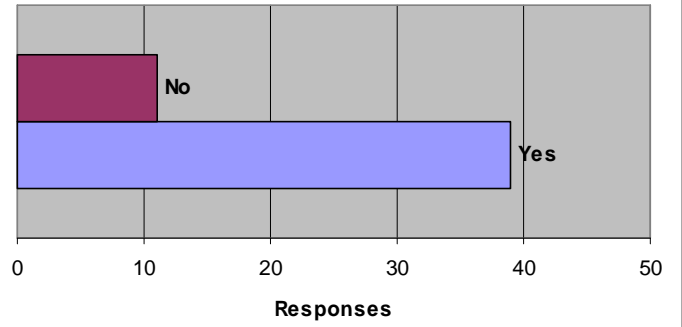
3rd Year Project Market Research Results



**Would You Ever Consider Buying Something
With Your Cell Phone?**



**Would Purchase a Product From a Vending
Machine With Cell Phone**



Appendix 2

Extended Set of GSM AT Commands for SMS

AT+CSMS: Select Message Service

AT+CMMS: More Messages To Send

AT+CPMS: Preferred Message Storage

AT+CMGF: Message Format

AT+CSCA: Service Center Address

AT+CSMP: Set Text Mode Parameters

AT+CSDH: Show Text Mode Parameters

AT+CNMI: New Message indication to TE

AT+CMGL: List Messages

AT+CMGR: Read Message

AT+CMGS: Send Message

AT+CMGC: Send Command

AT+CMGW: Write Message to Memory

AT+CMGD: Delete Message

AT+CMSS: Send Message From Storage

AT+CSCB: Select CBM Type

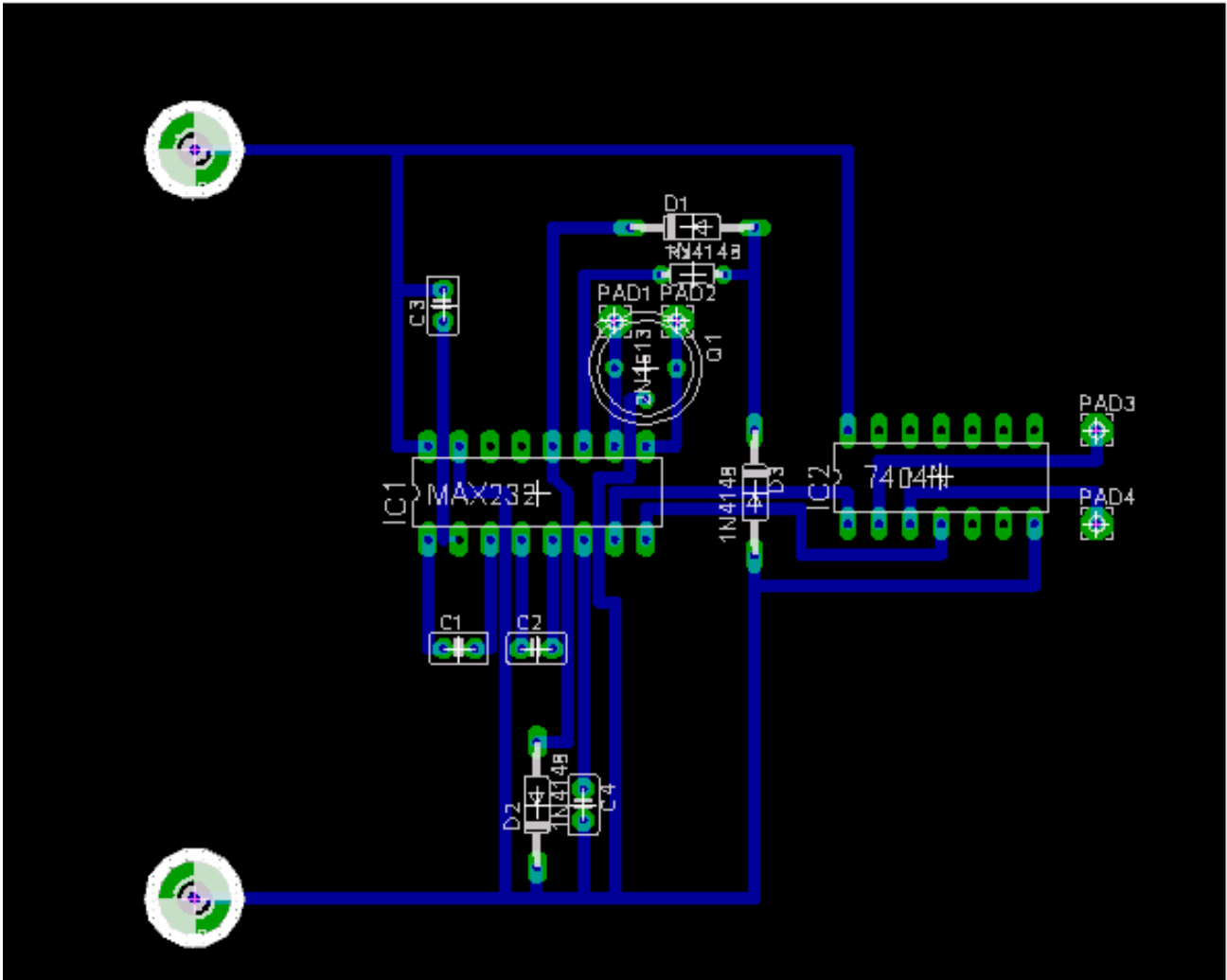
AT+CSAS: Save Settings

AT+CRES: Restore Settings

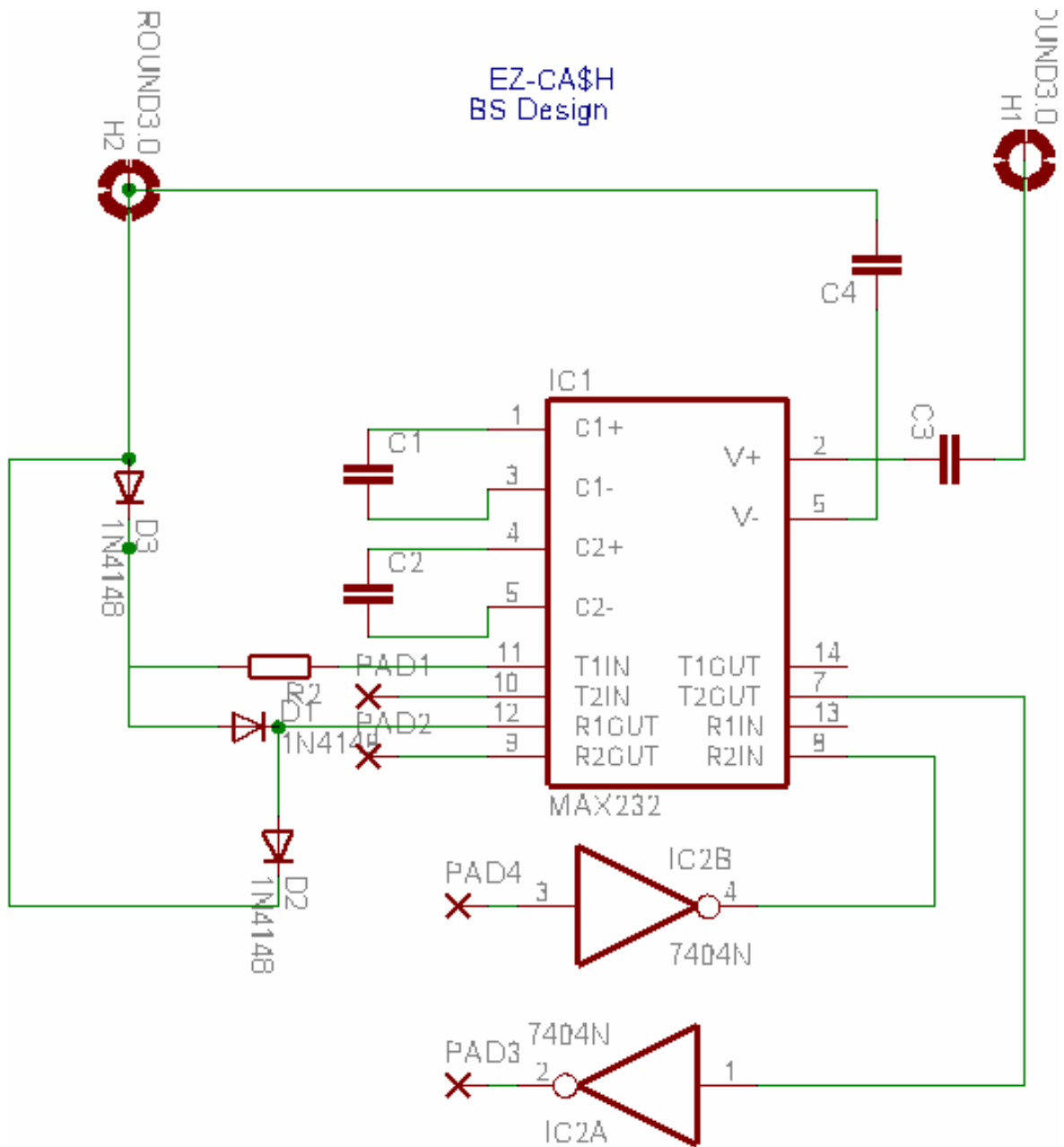
AT+CNMA: New Message ACK To ME/TA

Appendix 3

MAX 232 Schematic & Board Layout



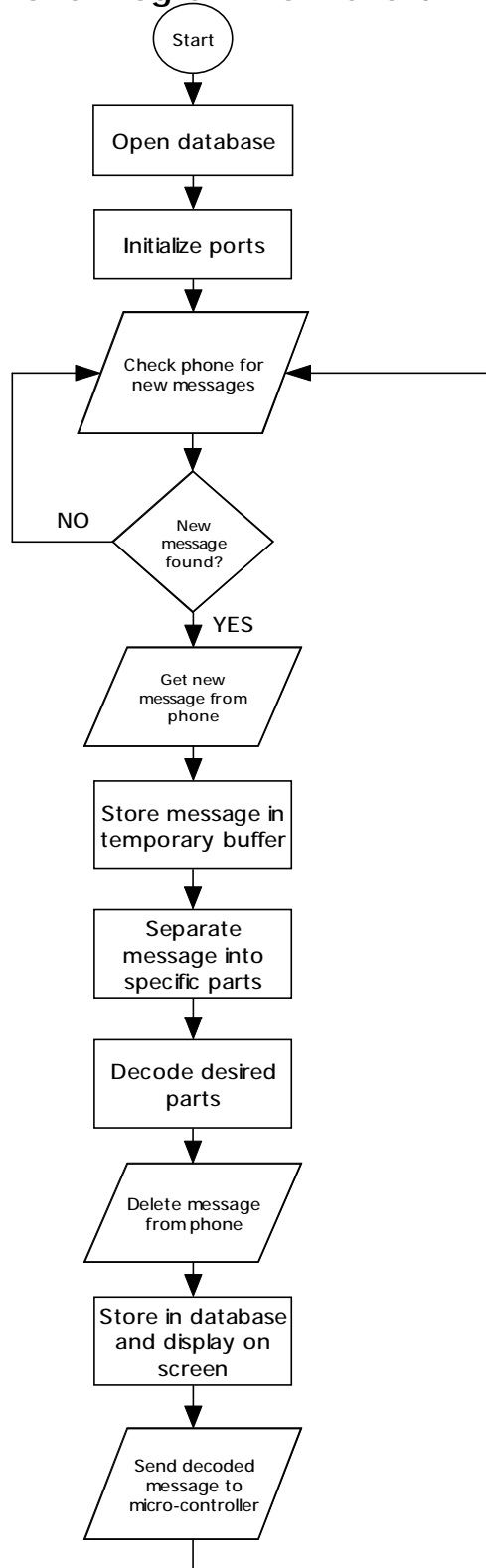
Max232 Board Layout



Max232 Schematic

Appendix 4 - Program Flow Charts & Main Code

Phone Program Flow Chart



Main Loop

```
void MainLoop()
{
Res_t *r;
float cst;
int Result;
    r =AllocateResponse();
    Initialise_DB();
    while(!kbhit())
        {
        Result = Get_Nokia_Result(r);
        if (Result != INVALID)
            {
            Tx_Nokia_Result();
            Unpack_Nokia(r);
            cst = Get_Cost(r);
            Store_Message(r,cst);
            TX_Item(r);
            }
        }
    Terminate_Coms();
    Update_DB();
    exit (0);
}
```

Transmitting To Phone

```
void TX_Command(int cmd,char param)
{
    switch (cmd)
        {
        case CMGR :
            {
            SioPuts(COM1,CMGRSTRG,8);
            SioPutc(COM1,param);
            SioPutc(COM1,13);
            break;
            }
        case CMGD :
            {
            SioPuts(COM1,CMGDSTRG,8);
            SioPutc(COM1,param);
            SioPutc(COM1,13);
            break;
            }
        case CMGL :
            {
            SioPuts(COM1,CMGLSTRG,8);
            SioPutc(COM1,param);
            SioPutc(COM1,13);
            break;
            }
        }
}
```

Receiving From The Phone

```
int RX_Response(Res_t *r)
{
    static char Response[256];
    static char Total[4];
    time_t timer, now;
    char c,t;
    int j,count,tot, offset;
    j = count = tot = offset = 0;
    t = 1;
    timer = time(NULL);
    while (t!=0) /* while character doesnt equal 0 read character */
        { /* from com port one at a time */
            c = SioGetc(COM1,0);
            if (c >0) /* if character is greater than 0 print character */
                { /* to screen */
                    //printf("%c",c);
                    Response[count]=c;
                    count++;
                }
            now = time(NULL);
            if (now >= timer+1)
                {
                    t = 0;
                }
        }
    count =0;
    c =0;
    while(c != 13)
        {
            c = Response[count];
            count++;
        };
    c = Response[count];
    if (c == 13)
        {
            count++;
            c = Response[count];
            count = count+4;
            if (Response[count] == 'S')
                return (INVALID);
            while(c != 13)
                {
                    c = Response[count];
                    count++;
                };
            offset =0;
            while(c != 13)
                {
                    c = Response[count];

                    Total[offset]=Response[count];
                    count++;
                }
        }
}
```

```

        offset++;
    };
    Total[offset] = '\n';
    tot = atoi(Total);
    count++;
    for (j=0;j<16;j++)
    {
        r->SCA[j] = Response[count];
        count++;
    }
    r->SCA[j]= NULL;
    for (j=0;j<2;j++)
    {
        r->ID[j] = Response[count];
        count++;
    }
    r->ID[j]= NULL;
    for (j=0;j<16;j++)
    {
        r->OrgAdd[j] = Response[count];
        count++;
    }
    r->OrgAdd[j]= NULL;
    for (j=0;j<2;j++)
    {
        r->ProtocolID[j] = Response[count];
        count++;
    }
    r->ProtocolID[j]= NULL;
    for (j=0;j<2;j++)
    {
        r->DCS[j] = Response[count];
        count++;
    }
    r->DCS[j]= NULL;
    for (j=0;j<14;j++)
    {
        r->SCTS[j] = Response[count];
        count++;
    }
    r->SCTS[j]= NULL;
    for (j=0;j<2;j++)
    {
        r->DataLen[j] = Response[count];
        count++;
    }
    r->DataLen[j]= NULL;
    tot = 2*atoi(r->DataLen);
    for (j=0;j<tot;j++)
    {
        r->Data[j] = Response[count];
        count++;
    }

```

```

        r->Data[j]= NULL;

        flushall();
        return VALID;
    }
    return INVALID;
} /* End of function */

```

Decoding The Message

```
void Unpack_Nokia(Res_t *r)
```

```

{
char Unpacked[4] = "";
char Packed[3] = "";
int j;
    for (j=0;j<3;j++)
    {
        Packed[0]=r->Data[(2*j)];
        Packed[1]=r->Data[(2*j)+1];
        Unpacked[j] = chartohex(Packed);
    }
    unpack(Unpacked, r->Decoded); // Decode data
    strcpy(r->OrgAdd, getOA(r->OrgAdd));
    strcpy(r->SCTS, getTime(r->SCTS));
}

```

```
char* getOA ( char* s)
```

```

{
    char* OA = "";
    int j = 0;
    for (j=0;j<11; j+=2)
    {
        OA[j]=s[j+5];
        OA[j+1]=s[j+4];
    }
    OA[11]='\0';
    return OA;
}

```

```
char* getTime (char* s)
```

```

{
    char* Time = "";
    int j;
    for (j=0;j<15;j+=2)
    {
        Time[j]=s[j+1];
        Time[j+1]=s[j];
    }
    Time[14]='\0';
    return Time;
}

```

```

void unpack(char *s,char *d)
{
int c,bc,i,j,N;
unsigned char mout,min;

    N = 0;
    while (s[N] != NULL)
        {
            d[N] = s[N];
            N++;
        }
    N--;
    c = 1;
    bc = 0;
    for (j=0;j<N+c;j++)
        {
            min = 0;
            if ((d[j] & 0x80)==0x80)
                min = 0x01;
            d[j] = d[j] & 0x7F;
            for (i=j+1;i<N+c;i++)
                {
                    mout = 0;
                    if ((d[i] & 0x80)==0x80)
                        mout = 0x01;
                    d[i] = (d[i] << 1)|min;
                    min = mout;
                }
            bc++;
            if (bc == 8)
                {
                    bc = 0;
                    c++;
                }
        }
    d[N+c] = NULL;
}

```

```

short int chartohex(char data[])
{
int number = 0;
int i = 0;
char* hexdigits = "0123456789ABCDEF";
for (i=0;i<16;i++)
{
    if (data[1]==hexdigits[i])
    {
        number=i;
    }
}
for (i=0;i<16;i++)
{
    if (data[0]==hexdigits[i])

```

```

        {
            number+=i+(15*i);
        }
    }
    return (number);
}

```

Deleting The Message

```
void DEL_Response(void)
```

```

{
    static char Response[256];
    static char Total[4];
    time_t timer, now;
    char c,t;
    int j,count,tot, offset;
    j = count = tot = offset = 0;
    t = 1;
    timer = time(NULL);

    while (t!=0)          /* while character doesnt equal 0 read character */
        {                /* from com port one at a time */
            c = SioGetc(COM1,0);
            if (c >0)     /* if character is greater than 0 print character */
                {        /* to screen */
                    //      printf("%c",c);
                    Response[count]=c;
                    count++;
                }
            now = time(NULL);
            if (now >= timer+1)
                {
                    t = 0;
                }
        }

    count =0;
    c =0;
    while(c != 13)
        {
            //      c = Response[count];
            printf("%c ",c);
            count++;
        };

    c = Response[count];
    if (c == 13)
        {
            count++;
            c = Response[count];
            while(c != 13)
                {
                    //      c = Response[count];
                    printf("%c",c);
                }
        }
}

```

```

        count++;
    };
    offset =0;
    while(c != 13)
    {
//      c = Response[count];
      printf("%c",c);
      Total[offset]=Response[count];
      count++;
      offset++;
    };
    Total[offset] = '\n';
    tot = atoi(Total);
    count++;
  }
  fflush();
  return;
}

```

Transmit Message To Microcontroller

```

void TX_Item(Res_t *r)
{
  int len,j;
  int k = 0;
  char i;
  char *invalid = " Invalid Choice";
  if(strcmp(r->Decoded, "A05")==0)
  {
    i = '1';
    k=1;
  }
  else if(strcmp(r->Decoded, "A10")==0)
  {
    i = '2';
    k=1;
  }
  else if(strcmp(r->Decoded, "B15")==0)
  {
    i = '3';
    k=1;
  }
  else if(strcmp(r->Decoded, "B20")==0)
  {
    i = '4';
    k=1;
  }
  else if(strcmp(r->Decoded, "C25")==0)
  {
    i = '5';
    k=1;
  }
}

```

```

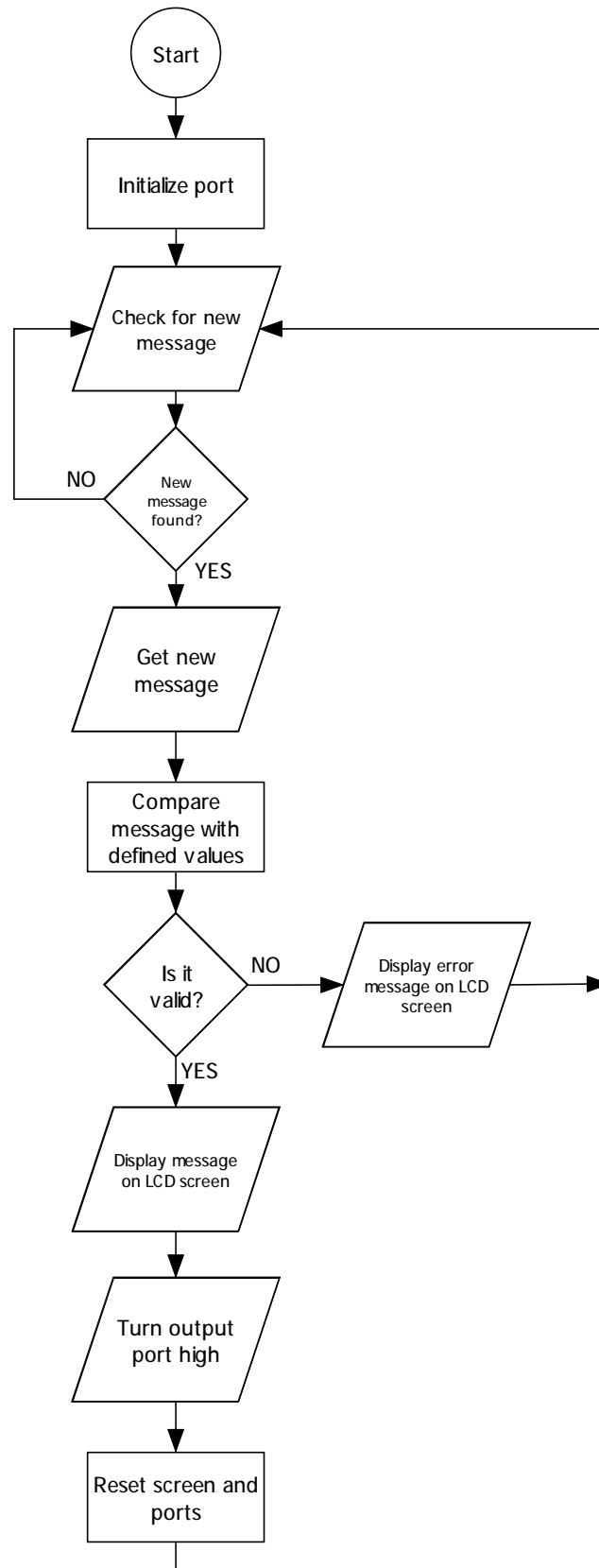
else if(strcmp(r->Decoded, "C30")==0)
    {
        i = '6';
        k=1;
    }
else if(strcmp(r->Decoded, "D35")==0)
    {
        i = '7';
        k=1;
    }
else if(strcmp(r->Decoded, "D40")==0)
    {
        i = '8';
        k=1;
    }

if(k == 1)
{
    len = strlen(r->Decoded);
    InitTX();
    PutTXc(i);
    for (j =1;j<7;j++)
        PutTXc(' ');
    for (j=0;j<len;j++)
        {
            PutTXc(r->Decoded[j]);
        }
    PutTXc(0);
}
else if(k == 0)
{
    len = strlen(Invalid);
    InitTX();
    PutTXc(' ');
    for (j=0;j<len;j++)
        {
            PutTXc(Invalid[j]);
        }
    PutTXc(0);
}
QueueTXMessage();
TXMessage();
}

```

*****For Full Project Code See Attached Disk*****

Micro Program Flow Chart



STATIONADD EQU \$01 Initial Station Address

INCLUDE "ALEV200.ASM"

***** MAIN ROUTINE *****

MAINP SEI A precaution! Make sure there are
LDS #STACKORG no interrupts while installing
JSR INITLCD Initialise LCD
JSR INITVAR Initialise general variables
JSR INITSCI Initialise SCI

JSR SCIRXON Turn RX SCI interrupt ON

CLI
LDY #\$1100
CLRA
STAA ,Y
MAINJMP2 LDX #EZCASH
JSR LCDTEXT

MAINJMP3 LDAA #LCDLN2 Display RX message on line 1
JSR GOTOLN
LDX #SELECT
JSR LCDTEXT
LDAA RXMESS Is there a received message
BEQ MAINJMP3 No!
JSR GETRXMSG Yes! Place it RX scratchpad

JSR CLRLINE

LDX #RXSCRPAD
INX Skip Length
INX Skip Address
LDAA ,X Load Scratchpad into Accum A
PSHA Push the first byte on to the stack
INX Increment X register
JSR LCDTEXT Jump to LCDTEXT
PULA Pull byte off the stack and into A
CMPA #\$31 Compare byte in A with values
BEQ DO1
CMPA #\$32
BEQ DO2
CMPA #\$33
BEQ DO3
CMPA #\$34
BEQ DO4
CMPA #\$35
BEQ DO5
CMPA #\$36
BEQ DO6
CMPA #\$37

```

        BEQ    DO7
        CMPA  #$38
        BEQ    DO8
        CLRB
        JMP    DONE
DO1     LDAB  #$01      Turn outputs "High" if match
        BRA   DONE
DO2     LDAB  #$02
        BRA   DONE
DO3     LDAB  #$04
        BRA   DONE
DO4     LDAB  #$08
        BRA   DONE
DO5     LDAB  #$10
        BRA   DONE
DO6     LDAB  #$20
        BRA   DONE
DO7     LDAB  #$40
        BRA   DONE
DO8     LDAB  #$80
        BRA   DONE

DONE    LDY   #$1100 Load Y with value indicating output
        STAB ,Y
        CLRA           Indicate that the RX pad is free
        STAA RXMSGRDY
        BRA   CLRSCR

CLRSCRJSR DELAY2
        JSR   CLRLINE      Clear LCD line 2
        LDX  #THANK
        JSR   LCDTEXT
        JSR   DELOOP
        JSR   CLRLINE      Clear LCD line 2
        LDAB #$00          Clear outputs
        LDY  #$1100
        STAB ,Y

***** Loop
        JMP  MAINJMP3      Loop

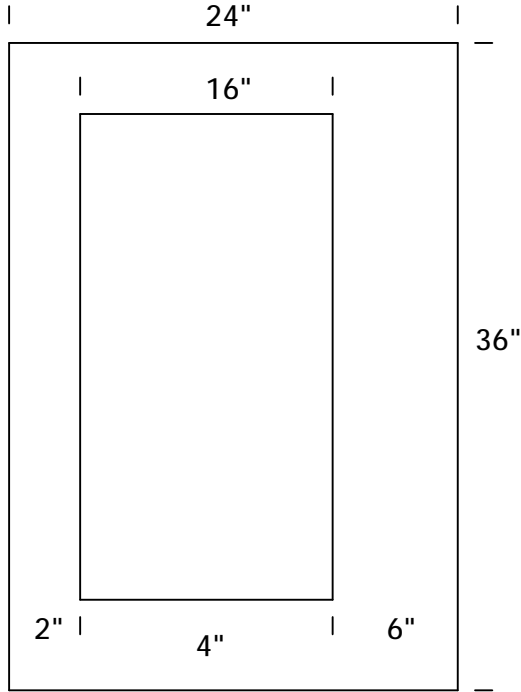
```

*****For Full Project Code See Attached Disk*****

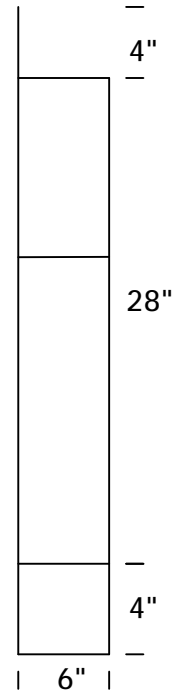
Appendix 5

Vending Machine Design

Front View



Side View



Top View

