EnginuiTech
Simon Fraser University
Burnaby, BC
V5A 1S9
Enginui-Tech@sfu.ca

March 12, 1999

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, BC
V5A 1S9

The attached document, Remote Automated Vending Statistics (RAVS) System Design Specification, specifies the operation and implementational details of the RAVS system. The RAVS System is designed to automatically monitor the status of a vending machine and periodically inform the servicer of its status.

The RAVS design specification outlines the major components of the RAVS system and then discusses the two major components - the Monitoring Unit (MU) and the Host system. This design specification includes the theory of operation, schematic diagrams, mechanical diagrams, and the design process for the RAVS system.

EnginuiTech was founded by Bill Moats, Shane Schneider, Nestor Siu, and Brad Oldham -four creative and dedicated third year engineering students. We would be happy to answer any questions or concerns you may have regarding our project proposal. I can be contacted via e-mail at wmoats@sfu.ca or by phone at 534-1584.

Sincerely,

Bill Moats, Team Manager
EnginuiTech

Enclosure: *Design specification*

EnginuiTech

Design specification

Submitted by	EnginuiTech
Contact	Bill Moats School of Engineering Science Simon Fraser University wmoats@sfu.ca
Submitted to	Andrew Rawicz School of Engineering Science Simon Fraser University Steve Whitmore School of Engineering Science Simon Fraser University

March 12, 1999

Executive Summary

It seems as though everywhere you look there is a vending machine selling everything from drinks to sandwiches. These pop machines require constant filling and maintenance and are routinely empty of product for a substantial amount of time causing the machines to be abused (think how many times you have seen someone hammering on a pop machine because it ate their quarter or doesn't have the drink they want). Valuable time and money is also wasted by suppliers by having to routinely check vending machines although they may not be empty.

To remedy this problem, EnginuiTech is developing the Remote Automated Vending Statistics (RAVS) System. This device will monitor the status of a single or a group of pop vending machines and report necessary information to the service personnel. The system will also generate sales statistics, which can be used to determine peak usage times and product approval.

This document introduces and defines the design of the RAVS System and its sub-components.

Table Of Contents

List of Figures

List of Tables

1. Introduction

How many times have you reached to put your money in a vending machine and noticed that the sold-out light is on for the product you want? Public areas are becoming filled with vending machines of all descriptions, which require a considerably large amount of time to constantly monitor and fill.

This document specifies the design of the Remote Automated Vending Statistics (RAVS) System developed by Enginuitech which will automatically monitor the status of the vending machine and report information to the service personnel when the machine requires filling. This system would save time and money for the servicer of many pop machines by allowing the scheduling of servicing and immediate re-filling of machines – so fewer customers would be turned away by empty machines.

This primary purpose of this project is to prove the concept is feasible by developing a prototype system on one pop machine and one host. It is also the intention of EnginuiTech to develop a proof of concept of a system such that it would be potentially marketable and flexible enough to be implemented with multiple vending machines in the future.

2. System Overview

The Remote Automated Vending Statistics (RAVS) System is intended to automatically send e-mail messages to a servicer when a pop machine runs low on pop or the internal temperature rises above a programmed limit. The system is divided into two separate entities: the Monitoring Unit (MU) and the Host as shown in Figure 1.

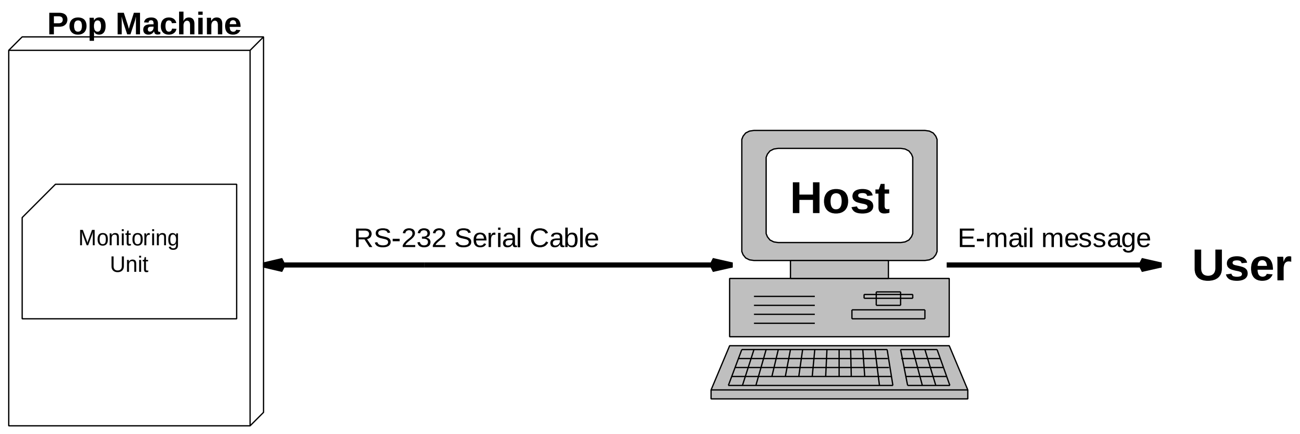


Figure 1: RAVS System Overview

The primary role of the Monitoring Unit (MU) is to record the levels of pop in each bin, the internal temperature of the pop machine, and the customer selections (including the time of the selection). The monitoring unit must then buffer this data and report this information to the host on request or at periodic intervals. The Host computer is responsible for receiving and logging the information from the monitoring unit and generating e-mail reports to the user when any pop levels are low or the interior temperature level rises above a programmable level. The schematic diagrams and other related technical documents are included in the Appendices.

As this project is intended to prove the concept is feasible, one MU and one host system are to be connected together by means of a serial cable. The Vending Information Protocol (VIP) is used for communication between these systems (described in Section 5.1) and will remain adaptable such that the communications medium between the monitoring unit and the host could be changed with little or no change to the existing systems.

The following sections outline the design of the monitoring unit and the host.

3. Monitoring Unit

3.1 System Architecture

The high-level system diagram for the Monitoring Unit is shown in Figure 2.

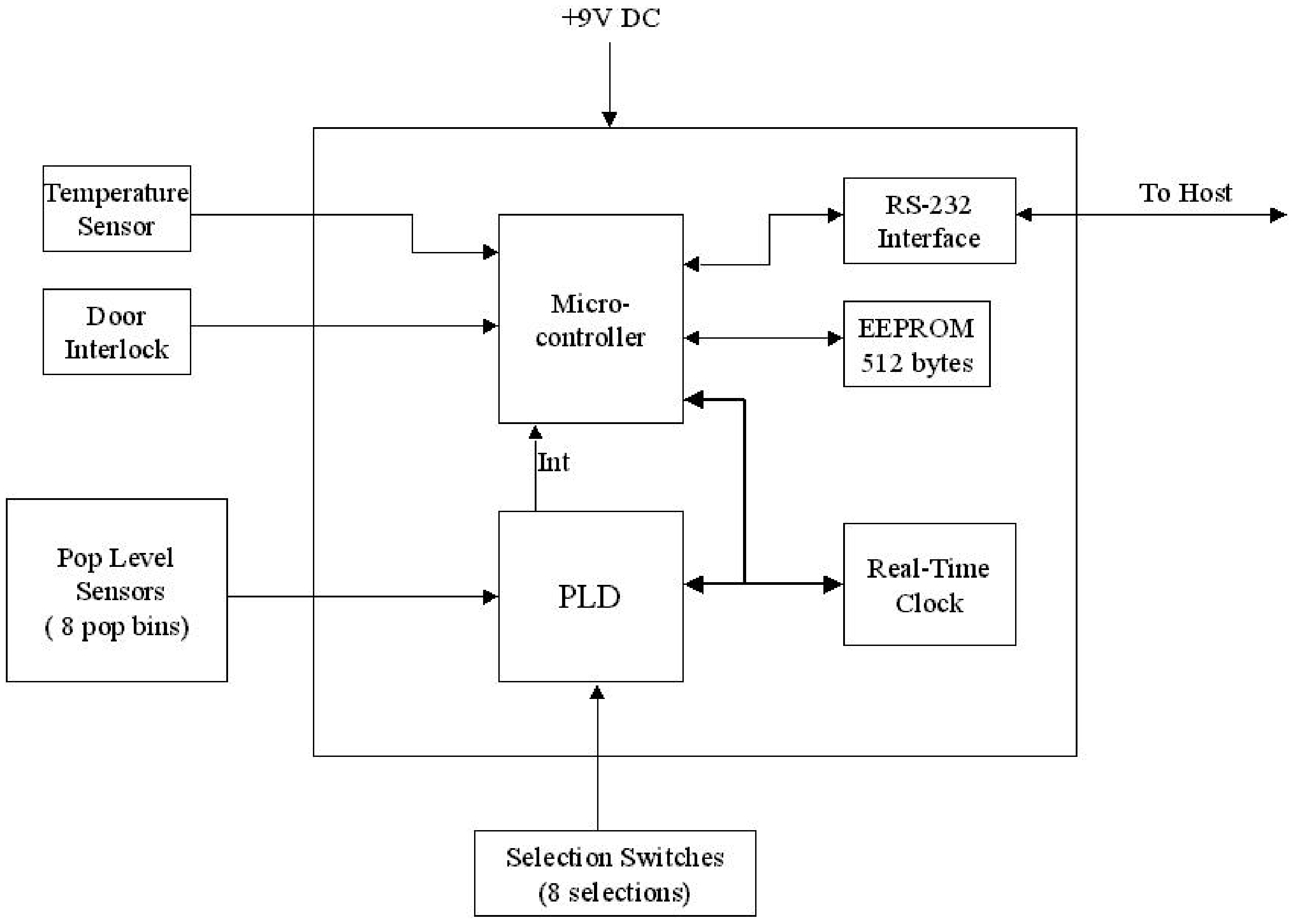


Figure 2: Monitoring Unit System Overview

To accomplish the tasks defined in the RAVS system functional specification, the Monitoring Unit (MU) design consists of a microcontroller, an EEPROM, an RS-232 serial interface, a real time clock (RTC), a Programmable Logic Device (PLD), a door interlock, a temperature sensor, pop selection sensors, and pop-level sensors.

The internal operations of the monitoring unit are performed by the microcontroller - which is selected to be the PIC 16C74A (made by Microchip, http://ww.microchip.com). This microcontroller has an internal Universal Synchronous/Asynchronous Receiver Transmitter (USART) which is used to communicate with the host system via serial cable. An RS-232 driver/ receiver is present to convert the voltage levels from the +5V and 0V signals produced by the microcontroller to those present in serial communications with the computer.

In order to store the necessary configuration parameters and the sensor logs in a non-volatile fashion (the data is not lost over power downs), an EEPROM module is connected to the microcontroller. A size of 512 bytes was selected to allow enough space for the data buffer and the configuration parameters maintained by the microcontroller. See Section 3.1.1 for the details of this data structure, as well as memory management and mapping.

A real time clock is included in the design to allow the microcontroller to be able to store the time of day when a customer purchases a product. The real time clock can also be programmed to generate interrupt signals at periodic intervals during a day which will be used by the microcontroller to control when the sensors are read and when the data log is downloaded to the host system.

The Programmable Logic Device (PLD) is included in the design to easily interface the microcontroller to the sensors and to generate interrupt signals. By using a PLD to implement the sensor interface, the number of sensor inputs can be changed easily with only minor changes to the microcontroller software and other components in the system.

3.1.1 Memory Mapping and Management

For every sale, a sales record containing the bin number and time of sale will be generated and temporary stored on the MU until the buffer reaches a watermark or the host requests a download. Each record will be composed of four bytes: bin number, day of month, hour, and minute. These sales records will be stored in a circular First In First Out (FIFO) list and have a limit of 15 records. Any sales above the 15 records will be lost, but will be indicated by an *overflow flag*. A watermark will be set at 8 records, after which time the MU will attempt to establish communications with the host. This low watermark will allow the freedom to add seven more records in the buffer without causing the overflow in case communications can not be established immediately. Once a record has been successfully downloaded to the host, it will be deleted from the buffer.

The FIFO buffer will be able to hold a maximum of 15 records, but because of the nature of a circular FIFO buffer, an extra blank record must be used to indicate if the buffer is full. This means 64 bytes will be allocated from the non-volatile RAM to implement the buffer. In addition, a copy of the head and tail index pointers, as well as the overflow flag must also be stored on the non-volatile RAM, each consisting of a byte.

The FIFO buffer has the benefit of keeping the sales records in order and also allows the removal and addition of records at the same time. As the pointers approach the top of the memory buffer allocated to the FIFO, they will rollover and reset to the bottom of the buffer ready to move up again. To make use of a natural rollover with binary numbers, sixteen record slots will be used. Since the index pointers are 8 bits long (1 byte) and 16 positions is only represented by 4 bits, a mask map will be used to keep the 4 least significant bits and remove the overflow bits that result from incrementing the pointers. This allows the index pointers to always contain a value between 0 and 15.
Figure 3 gives examples of the FIFO buffer.

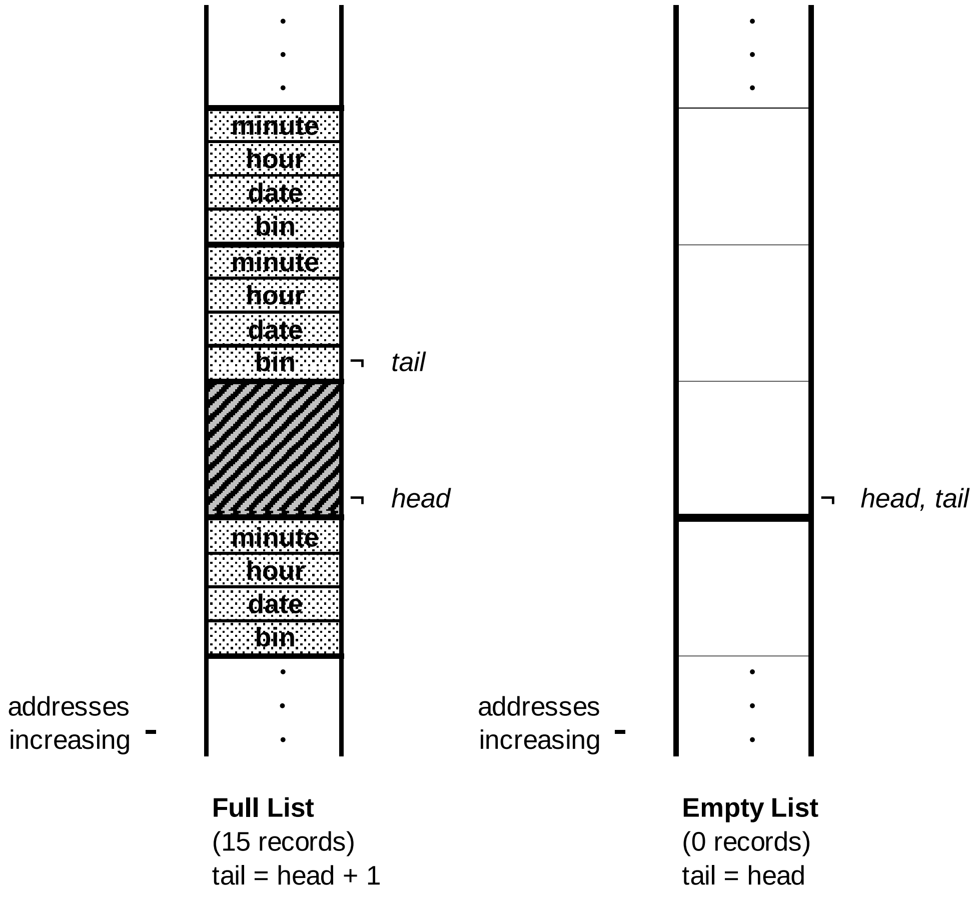


Figure 3: Examples of End Conditions for the FIFO Buffer

The variables and constants outlined in Table 1 will be used to implement the FIFO buffer.

Table 1: Constants and variables used by the FIFO buffer

Constant	Value	Purpose
`Base_address`	*To be determined*	Base address of the FIFO buffer in which the index head and tail pointers will be added to find the individual records.
`cp_head`	*to be determined*	Address of saved copy of the head pointer in non-volatile RAM.
`cp_tail`	*to be determined*	Address of saved copy of the tail pointer in non-volatile RAM.
`buff_overflow`	*to be determined*	Address of overflow flag in non-volitale RAM.
`index_map`	b'00111100	Mask map to remove the overflow bits and to make use of pointer rollovers.
`index_inc`	b'00000100	Used to increment the index pointers by four bytes (the size of 1 record).
Local Variable	**Purpose**	
`head`	Local (PIC) copy of the head pointer to be used in address manipulation.	
`tail`	Local (PIC) copy of the tail pointer to be used in address manipulation.	

3.1.2 PLD Interface Design

A Programmable Logic Device (PLD) will be used to integrate the various chips and sensors of the MU with the microcontroller. The PLD will also be responsible for managing all the incoming interrupt signals from various sources and generate a single interrupt and vector for the microcontroller.

The microcontroller will communicate with the various chips and sensors by sending a specific address to the PLD using an address bus. The three most significant bits will represent which chip or set of sensors the microcontroller wants to communicate with, while the four least significant will represent either an address or operand to be implemented. Given a complete address, the PLD will set the appropriate chip selects for the chips, implement the desired operand, or return the requested data via a data bus to the microcontroller. There will be three components the PLD will interface with for the microcontroller: the real time clock, the quantity sensors, and the selection switch interrupts.

After receiving an interrupt signal, the microcontroller will request the interrupt vector from the PLD and determine the source of the interrupt. Figure 4 gives a logic circuit for implementing the single interrupt signal.

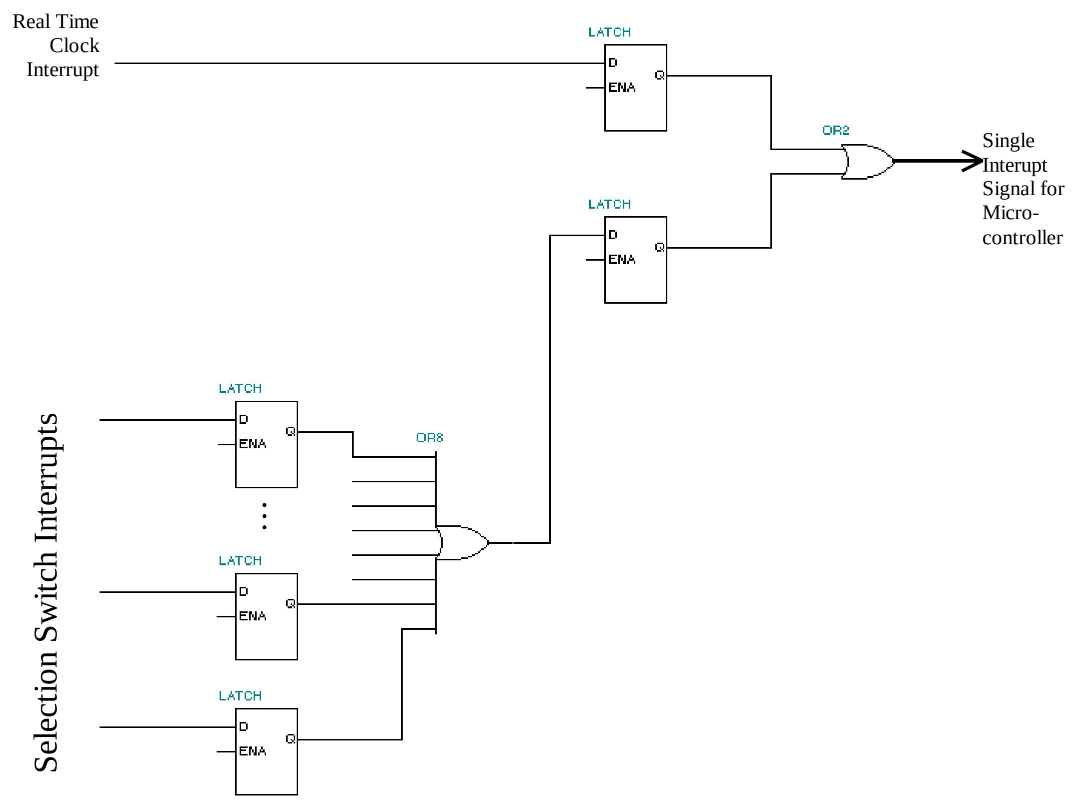

Real Time Clock Interrupt
LATCH
D
ENA
Q
OR2
Single Interupt Signal for Micro-controller
OR8
Selection Switch Interrupts

Figure 4: Interrupt Logic

3.1.3 Algorithmic Design

The software in the PIC microcontroller performs many event-driven tasks that must be able to be performed simultaneously. These tasks include:

- Receiving and processing instructions from the host over the serial port connection
- Processing purchases made by the customer and logging this data in memory in accordance with the data structure defined in Section 3.1.1.

- Downloading information to the host system

- Measuring and comparing the quantities of product in the pop machine at periodic intervals
- Monitoring the internal temperature of the pop machine

To make efficient use of the microcontroller's processing ability, an interrupt driven system will be utilized where most processing performed by the micro is initiated by an interrupt (from the PLD or an internal interrupt).

The real-time clock will generate a periodic (programmable) interrupt signal to indicate when the information in the data log must be downloaded and when the sensors must be read. The selection sensors will also generate an interrupt signal used to add a purchase to the data log. When a new character has been read from the host serial port an interrupt will be generated in order to read the character into the processing routine.

The high-level software design for the monitoring unit is shown in Figure 3.

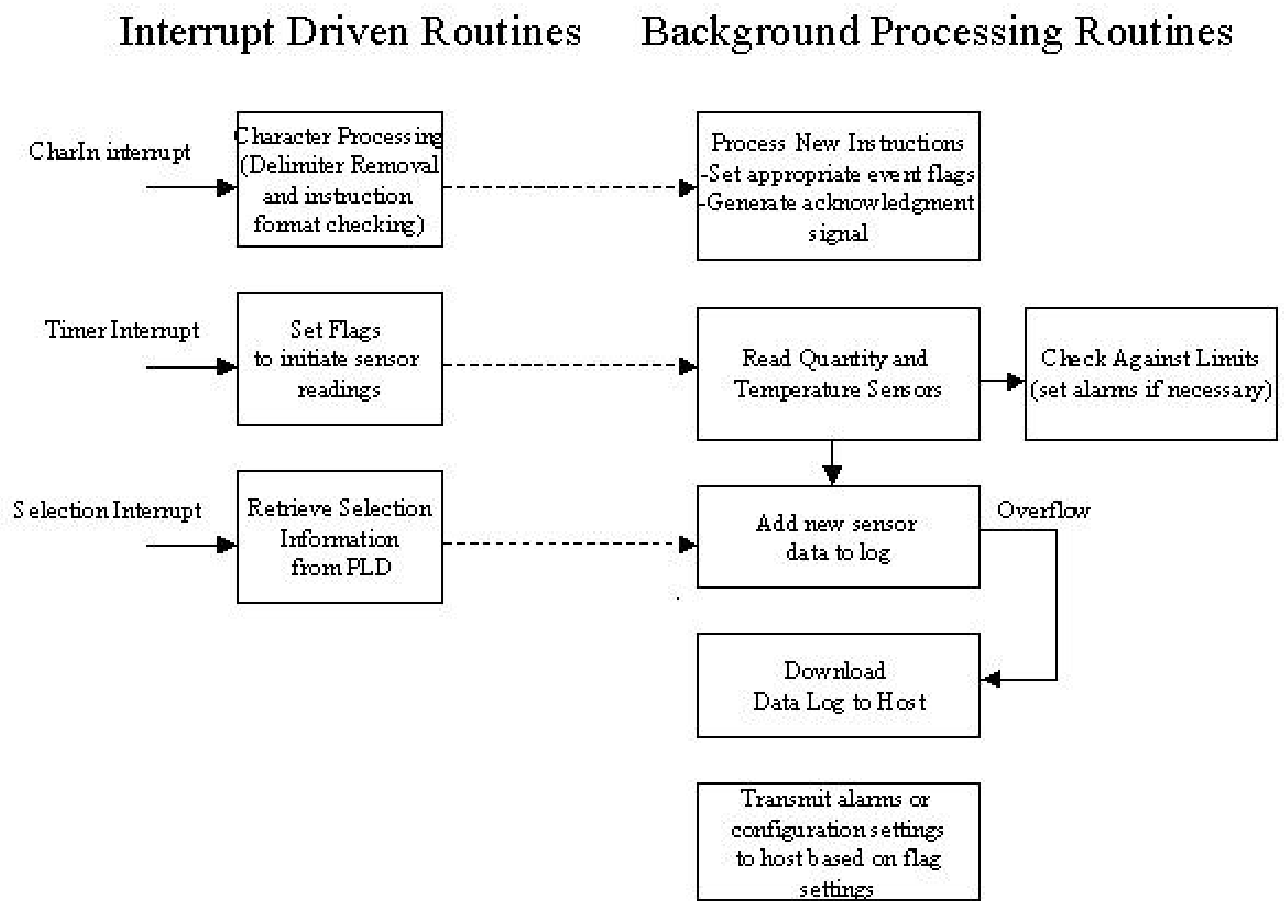

Interrupt Driven Routines
Background Processing Routines
CharIn interrupt
Character Processing (Delimiter Removal and instruction format checking)
Process New Instructions -Set appropriate event flags -Generate acknowledgment signal
Timer Interrupt
Set Flags to initiate sensor readings
Read Quantity and Temperature Sensors
Check Against Limits (set alarms if necessary)
Selection Interrupt
Retrieve Selection Information from PLD
Add new sensor data to log
Overflow
Download Data Log to Host
Transmit alarms or configuration settings to host based on flag settings

Figure 5: High Level Software Design for the Monitoring Unit

The software is separated into the event driven (time critical) functions and the background processing functions. The background processing tasks perform the majority of the processing required by the monitoring unit, such as the processing of new instructions, reading the quantity and temperature sensors, logging new data, and downloading information to the host. The interrupt routines set status flags to activated the appropriate background task and perform some low level processing such as the character processing (delimiter removal routine) which are required by the Vending Information Protocol (see Section 5.1).

3.1.4 Instruction Processing

The software must process instructions from the host in an efficient manner and be compliant with the Vending Information Protocol (VIP) (see Section 5.1).
This protocol uses variable length character delimited instructions, which are fomatted as follows:

<open delimiter> <opcode> <operands> <close delimiter>

The instruction processing software will provide data “transparency” routines to reduce the number of processing cycles performed by the background processing. This “transparency” routine will buffer every incoming character from the host and allow only properly delimited instructions of the correct length to pass to the processing routine. This method of instruction processing reduces the chances of random input data being interpreted as an instruction code and allows this communications system to be easily upgraded to a larger network with multiple vending machines. The name transparency is used to indicate the fact that the end system routines are not aware of the implementation of the delimiters and thus the protocol is transparent.

Once a proper instruction has been detected, a processing flag is set to allow the background routine to process the new instruction. To avoid overwriting a new instruction before it has been processed the character interrupt will be disabled while the new instruction is being processed. The data “transparency” state machine is shown in Figure 6.

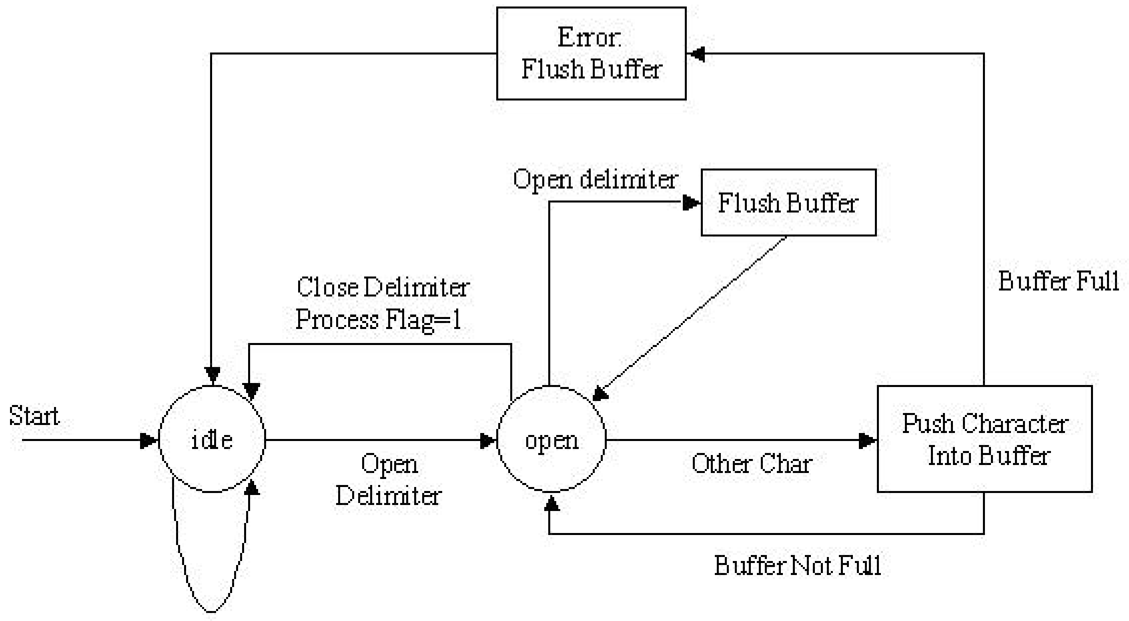


Figure 6: Data “Transparency” State Machine for the Instruction Processor

Once the instruction processing flag has been set, the background processing task will process the new instruction.

3.2 Sensor Design

3.2.1 Quantity Sensors

The quantity sensors measure the volume of pop in a given bin by detecting the presence of a pop can at discrete levels. For the implementation of this project there will be 3 sensors per bin. The sensors are placed at positions shown in Figure 7.

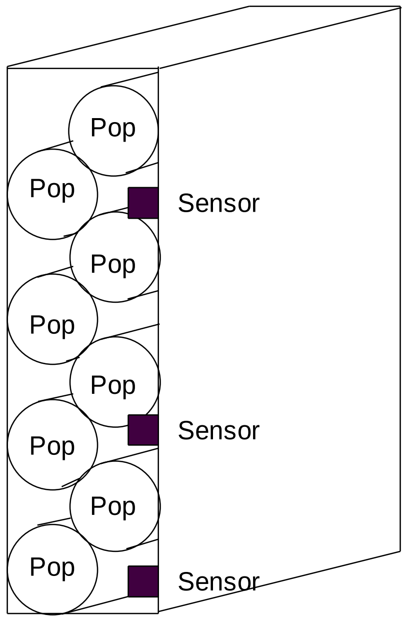


Figure 7: Row assembly

The sensors will measure the pop level in a quantized fashion (nearly empty, part full, fairly full). The sensors we considered for sensing pop are shown in Table 2.

Table 2: Sensor selection

Sensor	Pro's	Con's
Latches, Mechanical	Limited electronic worries	Interferes with machine loading
Break Sensors, Mechanical	Limited electronic worries	Interferes with machine loading
Break Sensors, Electronic	No mechanical break down	Long distance required, lot of electronic hardware, high noise pickup do to distance
Ultrasonic Sensors, Electronic	Measure absolute quantity, easy to install	Echoes caused by confined space

Reflective Sensors, Electronic	No mechanical break down, Low noise	Electronic hardware, noise rejection

The reflective sensors will be implemented do to the minimal amount of installation into the pop machine as well as the limited noise pick-up. The ultrasonic sensors would have been preferred but the echoes in the pop machine would cause too much interference and cross talk. The cost of the reflective sensors was lower then long distance optic sensors, which lead to choice of reflective sensors over break sensors (electronic). The interference of the mechanical sensors with the loading of the pop machine lead to there dismissal.
The pop cans will be sensed using the OP8706A Infrared Reflective Object Sensors (made by OPTEK http://www.optekinc.com). The sensors emit infrared light (IR), via an light emitting diode (LED), towards the pop in the bin. If there is a pop can in the bin the light is reflected back to a phototransistor where the signal is converted to current. The reflective object sensor is shown in Figure 8.

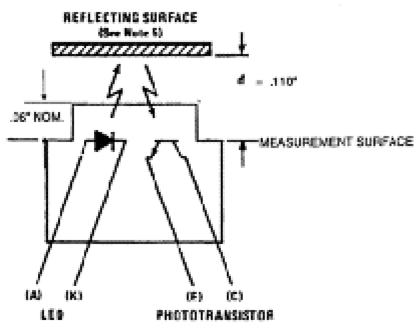


Figure 8: Reflected object[1]

To increase the range of the sensor, the voltage waveform into the LED is a square wave with frequency of approximately 1 kHz. This small signal is amplified by an very high gain amplifier and then smoothed into a DC signal by means of an envelope follower. This DC signal is then compared to a reference voltage by means of an analog comparator to convert the sensor output to a 1-bit digital value (refer to Figure 9).

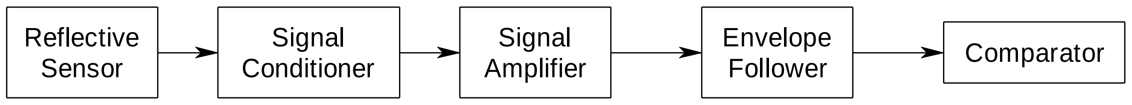


Figure 9: Sensing Circuit Flow Chart

The circuit diagram for the sensor circuit is shown in Figure 10.

[1] Taken from page 2 of the OPB706A data sheet, OPTEK Inc. (www.optekinc.com)

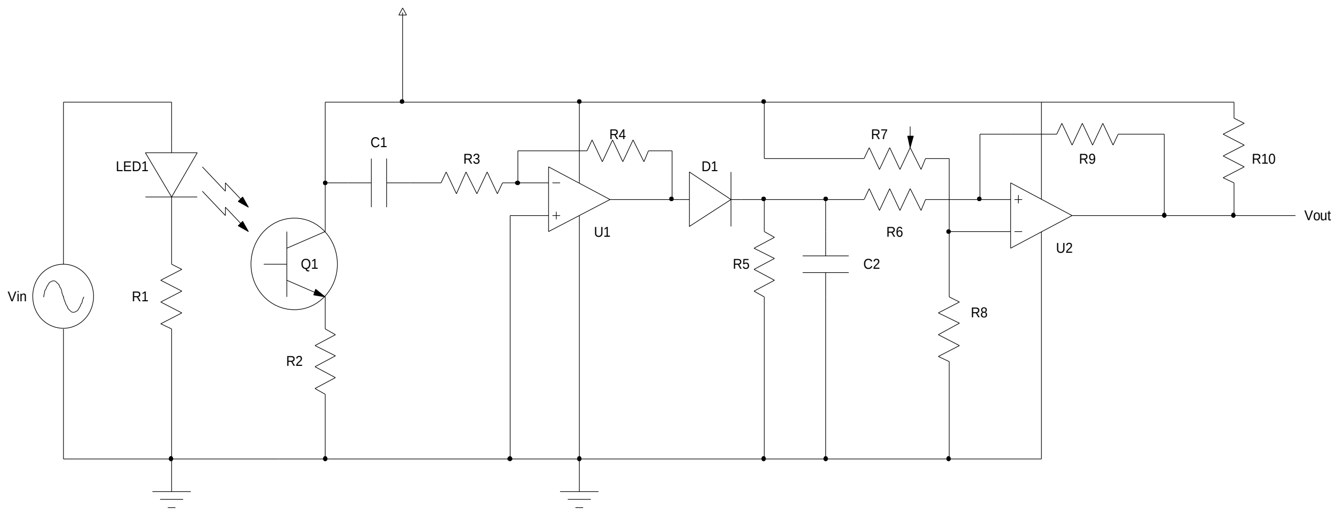


Figure 10: IR Sensing Circuit

The capacitor C1 is used to remove the DC signal from the output of the reflective object sensor. This signal is then amplified with an Op Amp with very high gain - calculated by the formula $\frac{R4}{R3} \approx 10^4$. The envelope follower smooths the sinusoidal output of the amplifier to a relativly constant DC value. The envelope follower consists of a diode, resistor and a capacitor with a time constant of $\frac{1}{R5 \times C2}$. The comparator is used to convert this DC voltage to TTl digital signal by comparing to the reference voltage (Vref) of $\frac{R7}{R7+R8}Vcc$ and the %hysterisis is $\frac{R6}{R9}$.

3.2.2 Temperature Sensor

One temperature sensor will be installed inside the pop machine to measure the internal temperature of where the pop is stored.

The temperature sensor will be a single voltage supply centigrade temperature sensor that converts the current temperature into a low voltage signal. This low voltage signal will then be amplified to allow a voltage swing between 0V and ~5V using a non-inverting op-amp configuration. This voltage signal is converted to a digital number using the A/D converter in the microcontroller. Figure 11 show the temperature sensor circuit.

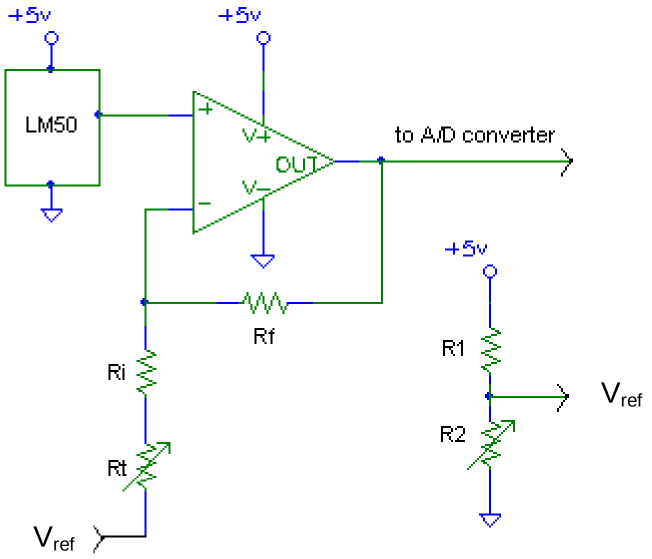


Figure 11: Temperature Sensor Circuit

This temperature sensor will be able to sense temperatures from -10°C to +50°C. The reference voltage, Vref, is used to eliminate the DC offset the generated by the temperature sensor, which would waste bits in the converter reducing resolution. Both the reference voltage and the gain of the amplifier can be calibrated using the two trim pots Rt and R2.

3.2.3 Interlocks / Selection Sensor

When the door of the pop machine is open, external light can enter the system which could potentially interfere with the operation of the infrared quantity sensors. Also the internal temperature would change quite dramatically. To avoid producing invalid results due to the door of the pop machine being open, a switch will be mounted on the door fixture such that the microcontroller can detect when the door is open. When the door is sensed to be open, the microcontroller will suspend all sensor measurements.

In order to sense what product a customer has purchased, a series of switches are to be mounted inside the vending mechanism such that the switch is closed when a purchase is made. An alternative would be to connect these switches to the selection switches on the front of the vending machine.

Both the door interlock and the selection switches will produce “digital” signals by utilizing the circuit shown in Figure 12. These signals will be routed into the PLD where the internal logic will perform all the necessary de-bouncing and interrupt generation.

When the switch is open, the output will be pulled up to +5V through the pull-up resistor. When the switch is closed, the output will be shunted to ground (0V).

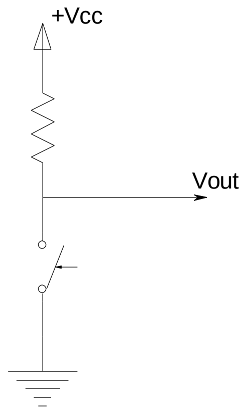


Figure 12: Selection Sensor and Door Interlock Sensor

3.3 Component Selection

3.3.1 Microcontroller

Because of the cost consideration and the low resolution sensor data required to be processed, the monitoring unit will utilize an 8-bit microcontroller. The Enginuitech design team investigated the use of Microchip's PIC 16C7x series microcontroller, and the Motorola's HC11 microcontroller.
The features that we desired for the microcontroller were

- USART module on board
- Large number of I/O pins
- Non-Volatile Program Memory
- Internal A/D converter
- Multiple Interrupt Sources (external and internal)
- Simple Instruction Set
- Serial Peripheral Interface (SPI) compatible
- Excellent Development Environment (MACRO assembler)
- Simple Packaging (DIP)

These features will allow a large number of the Monitoring Unit's features to be easily implemented the development of the software easier. Both the HC11 and the PIC have the majority of the necessary features. We selected the Microchip PIC16C74A microcontroller over the HC11 because of the large number of I/O pins (33) in a DIP package, the RISC architecture which includes 25 instructions, and the excellent MPLAB4 development which provides a free macro assembler and linker. Another advantage was that Enginuitech already possessed the necessary tools to program and erase the PIC series microcontroller. An advantage of the PIC microcontroller is that many of the chips in the PIC series are code and pin compatible which means that if more resources are needed later in the development, a larger chip can be substituted.

3.3.2 PLD

The Altera MAX7000S EPM7128SLC84-7 will be used as the PLD interface chip because of its cost, availability, and the familiarity EnguiTech has with its design environment. The one drawback this chip does have, it lacks a large number of I/O pins. This required the number of bins to be monitored from 16 bins to 8 bins.

3.3.3 Temperature Sensor

The temperature sensor circuit is shown in Figure 11. The LM50B temperature sensor was selected because of its single supply capability and its sensitivity range (-40°C to 125°C). This sensor was also selected because of its low price.

3.3.4 Memory

To store all the configuration parameters and the data log generated by the microcontroller we investigated several different methods of data storage.
We considered the use of

- Static RAM
- Electrically Erasable Programmable Read Only Memory (EEPROM)
- FLASH memory

We found that the use of static RAM would result in the loss of the data log and configuration parameters every time the power was cycled. The Xicor X25043 serial EEPROM was selected over several other devices due to its small package (DIP8), its low pin requirement (4 pins as opposed to 11 pins in some FLASH devices), and its other features such as built in watchdog timer and brown-out reset capability. It also does not loose any data over power cycles

3.3.5 Real Time Clock

We considered several different real-time-clocks for use in the monitoring unit. We narrowed the selection to the DALLAS DS12887 and the National Semiconductor DP8573A. The DALLA DS12887 was selected because it is a complete modular design (everything necessary to run including oscillators is included in the package), it has an internal battery backup that can last for ten years without external power, and it uses a multiplexed data/address bus. These features limit the number of clock updates to a minimum and make the device easy to integrate into the design.

3.3.6 RS-232 Driver

To convert between the standard +5V and 0V signals present in the monitoring unit to the higher voltages used in the RS-232 serial port we selected the Analog Devices ADM232 - an “off-the shelf” RS-232 driver/receiver able to generate the necessary voltages from only a+5V DC supply. An advantage is its compatability with the MAX232 driver/receiver.

3.4 Power Supply

In accordance with CSA regulations, the power requirements of the monitoring unit have been changed from 110VAC to +9VDC – which also prevents any unnecessary shock hazard the system could pose.

Thus, the system’s power supply consists of a voltage regulator and associated filter capacitors. The power supply circuit is shown in Figure 13.

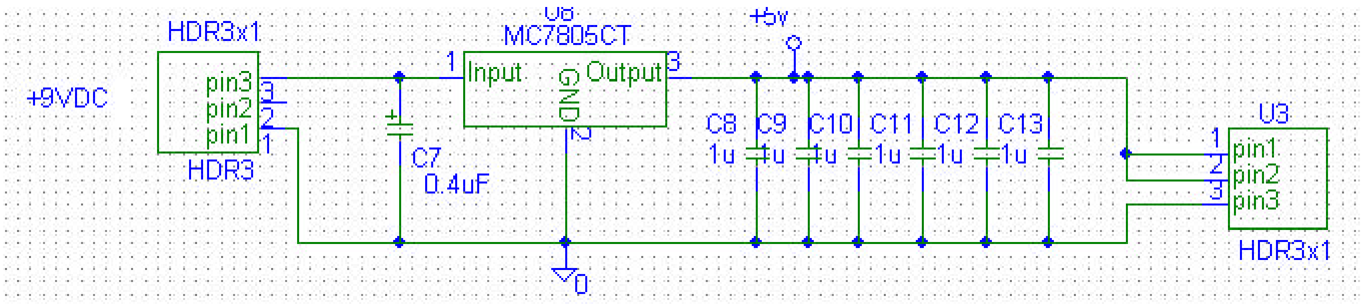


Figure 13 Power Supply Circuit

3.5 Mechanical Design

In accordance with the RAVS functional specification, the case of the monitoring unit is an aluminum box with dimensions 15cm by 10 cm by 8 cm. The aluminum will provide some shielding against Electro-Magnetic Interference (EMI) which could be generated by the cooling system and the many motors present in a pop machine.

The monitoring unit uses a DB-9 Male for RS232 communication, a DB-9 Female connector for the selection sensors and a DB-25 Female connector for the quantity sensors. These connectors were selected because of their high availability and the ability to group all of the sensor signals through one common connector.

3.6 User Interface

On the exterior of the monitoring unit case there is a power switch and several status LEDs. These LEDs indicate

- The status of the transmit line to the host (TX)
- The status of the receive line from the host (RX)
- The status of the power to the unit

Several LEDs will be present on the interior of the case for the purposes of debugging. The mechanical drawings for the monitoring unit case are included in Appendix

4. Host

The host is responsible for handling alarms or events generated by the MU. It is also responsible for downloading and storing any data logs that the MU buffers. Furthermore, the host is responsible for setting the MU configuration parameters. The host logs all data logs and events generated by the MU so that a status report may be generated from them.

For the purpose of this project, we will be developing our program on the Windows 95 platform. This operating system has a very large established user base, giving us access to many pre-developed software libraries, as well as many sophisticated software development tools to aid us in the software development process.

The C++ programming language will be used because it is a language familiar to the members of EnginuiTech.

4.1 System Architecture

The architecture of the Host is depicted in Figure 14:

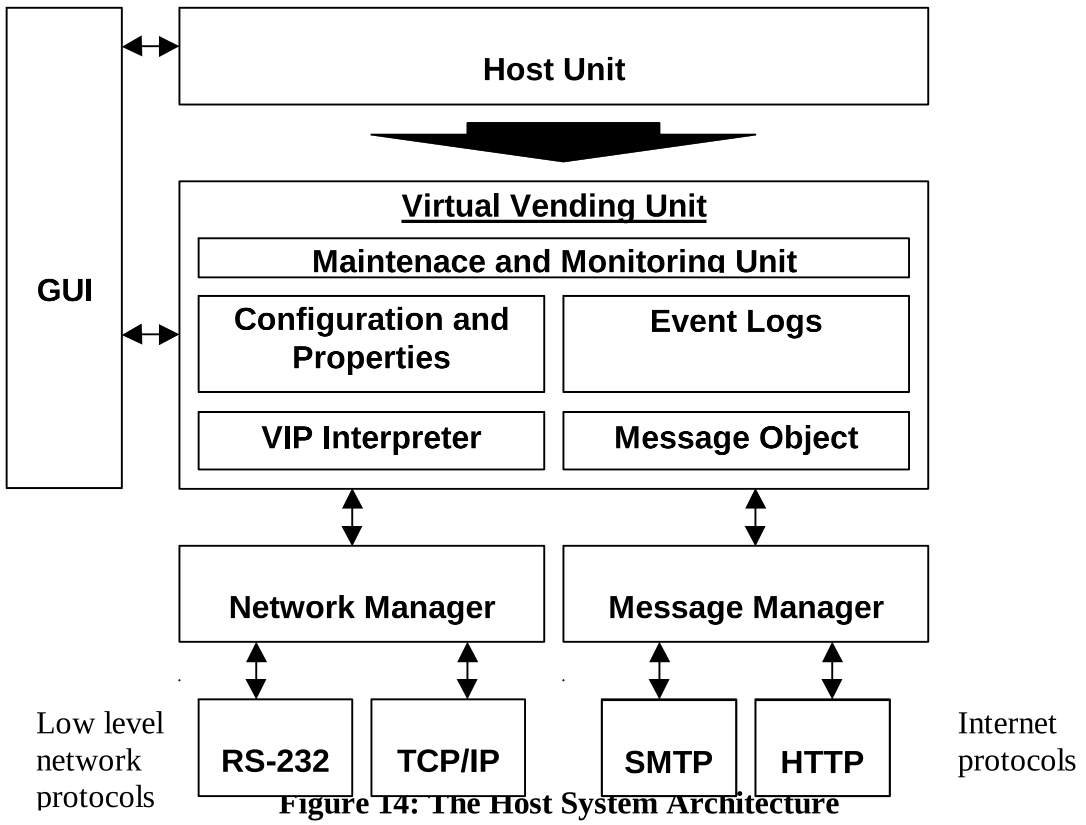


Figure 14: The Host System Architecture

Below is a description of each of the components of the architecture:

4.1.1 The Host Unit

This object acts as the main program of the system, incharge of instanciating all the major objects/components in the system which includes the Graphical User Interface (GUI), **Virtual Vending Unit** (**VVU**), **Network Manager** (**NM**), **Message Manager** (**MM**), and any low level network and Internet objects. It is also responsible for interconnecting all the objects together. For example, for each vending machine that needs to be controlled, the main program will create **Virtual Vending Unit** and register it to all the major permanent object dynamically setup the event handlers.

As a result, the **Host Unit** will be responsible for storing the following information:

- List of vending machines to be managed
- List of low level network objects to instantiate
- List of internet protocol modules to instantiate

4.1.2 The Virtual Vending Unit

The **Virtual Vending Unit** (**VVU**) represents a single vending machine that needs to be controlled. This self-contained unit will be responsible for handling and logging any events or alarms that may arrive, as well as perform maintenance tasks such as downloading sales data. Below is a list of internal objects that are controlled by the **Virtual Vending Unit**:

4.1.2.1 Maintenance and Monitoring Unit

The **Maintenance and Monitoring Unit** (**MMU**)is responsible for the execution of scheduled tasks. At the moment, this task includes the periodic update of the following MU status:

- Current Pop Levels
- Machine Temperature
- Sales Logs

The **Maintenance and Monitoring Unit** is also responsible for handling any alarms or events generated by the **VIP Interpreter** (see Table 3). Note that an alarm/event is only processed after the Monitoring Unit has established a session to the host, sent its requests, and closed the session. The **MMU** will then download all sales logs and status from the Monitoring Unit so that it can form the proper status messages to e-mail or page the vending machine personnel with. The messages are then sent, and the alarm/event will be logged. Except for the **ConnInd**, **DiscInd**, and **LogBufferLowInd** events, all events will be logged.

4.1.2.2 Configuration and Properties

The **Configuration and Properties** object is responsible for storing all the configuration and status information pertaining to the vending machine being monitored. The object is responsible for generating a **ChangeNotify** event to signal other objects such as the GUI which need to monitor the status of the vending machine. The following configuration information of the vending machine are stored:

- Vending Machine name
- Vending Machine ID
- Status update interval
- Valid temperature range
- Temperature out of range message (address, subject, message)
- Low pop level
- Low pop level message (address, subject, message)
- Log buffer overflow message (address, subject, message)
- Global alarm enable for all alarms

The following Pop machine status are stored:

- Door status
- Current pop levels
- Machine temperature

4.1.2.3 Event Logs

The **Event Logs** object is responsible for logging any events generated by the Monitoring Unit inside the vending machine. It is also responsibe for generating a **ChangeNotify** event to notify other objects that a log has been added. The following types of events will be logged:

- Pop Sale
- Temperature out of range
- Pop level low
- Monitoring Unit power up
- Log buffer full
- Log buffer overflow

4.1.2.4 VIP Interpreter

The **VIP Interpreter** is a network object requested from the **Network Manager** which is responsible for interpreting the **Vending Information Protocol** and generating appropriate events to handle a protocol session. Table 3 is a list of all the events generated by the **VIP Interpreter**:

Table 3: VIP Interpreter Events

MUI Event	**Event Description**
ConnInd	MU is opening a session to the host.
DiscInd	MU is closing its session to the host.
PowerUpInd	MU has just been powered up.
TempAlarmInd	Temperature in vending machine is out of valid range.
LevelLowInd	A bin quantity is getting low (or is below the minimum quantity level).
LogBufferLowInd	The sales log buffer is more the half full and should be downloaded.
LogBufferOverflowInd	The log buffer has overflown and lost the most reset sales log entry.

Except for the **ConnInd** and **DiscInd** events, all events are generated only after the Monitoring Unit has established a network session with the host, sent its requests, and closed the session.

4.1.3 Network Manager

The **Network Manager** (**NM**) is used to provide an abstract, high level interface to the different network resourses used to create a network session between the Host and the Monitoring Unit. This allows the host to support Monitoring Units connected to a variety of different network mediums and protocols such as RS-232 and TCP/IP.

The **Network Manager** is used to obtain a **VIP Interpreter** object which is used to communicate with a **single** Monitoring Unit using the **Vending Information Protocol**. When a packet of information arrives, the **Network Manager** is incharge of determining which Monitoring Unit sent the packet, and it then routes the data packet only to the **VIP Interpreter** that is supposed to receive it.

The **Host Unit** will be responsible for registering the low level network objects which the **Network Manager** will have access to.

4.1.4 Message Manager

The **Message Manager** is used to provide an abstract interface to different Internet protocols such as SMTP and HTTP which will be used to send out messages to alert vending machine personnel that a vending machine needs maintenance. Objects which need to send out message typically requests a **Message Object** from the **Message Manager** to perform the actual sending of the message.

Again, the **Host Unit** is incharge of creating the **Message Manager** as well as register any Internet protocol objects to the manager.

4.1.5 Low Level Network Protocols

The **Low Level Network Protocol** objects are used to provide access to different network or communication mediums such as RS-232 and TCP/IP. These objects will provide a simplified interface which will be used to integrate with the **Network Manager**. The interface will allow the **Network Manager** to intialize and close the network resource using an **init()** and **close()** method. Data can then be send using **write()** method, and it is received through a **DataInd** event. For the purpose of this project, we will only be implementing the RS-232 interface.

4.1.6 Internet Network Protocols

The **Internet Network Protocols** objects are used to provide access to Internet protocols such as SMTP and HTTP which can be used to send out e-mail messages as well as communicate with web based interfaces to pagers and digital cell phones. These objects will be used to provide an intance of a **Message Object** which is used to send out messages. For the purpose of this project, we'll be supporting the HTTP and SMTP protocols.

4.1.7 Message Object

The Message Object provides an interface into a specific Internet Protocol. This object is used to send out a message to alert the vending machine personnel of vending machine problems or status.

4.2 Data Structure Design

This section describes the configuration file used by the host and the format of the log files.

4.2.1 Configuration File

The configuration file will employ the format of .INI files originally used with Windows 3.1. This text based file format allows us to easily store all configuration parameters inside a single file while keeping configuration paramters for each different object in serparate sections. Also, the file format allows the user to modify the configuration manually when such a need arises. Using text characters also allow us to manually interpret configuration settings for debugging purposes.

The basic structure of the file consists of sections which are marked by a section header. The format of a section header is

```
[<Section Identifier>]
```

where <*`Section Identifier>`* is a text string used to identify the header. Inside each section, individual parameters can be stored using the following format:

`<field identifier>=<value>`

where ***`<field identifier>`*** is a text string used to identify the parameter, and *<value>* is the value of the parameter. Each field is separated by a new-line character. Blank lines are ignored. Lines which begin with a ";" character are treated as comments and ignored.

Each object that is instanciated by the **Host Unit** will be given its own section in the configuration file. Each time the object is instantiated, an object representing the configuration file, as well as a unique identifier string to identify the object will be passed into the parameter list. The object will then extract its parameters from the section designated by the indetifier for initialization.

4.2.2 Log File

A log file will be generated for each vending machine monitored. The log file will contain all major events and alarms generated by the Monitoring Unit. This file can later be used to generate status reports or sales reports which can be easily imported into third party programs like Excel. The format of each log entry is as follows:

*<yyyy>**/**<mm>**/**<dd> <hh>**:**<mm> <entry type> <parameters …>*

The types of events and alarms logged are listed in Table 4:

Table 4: Logged Events

Log Entry Type	**Parameter**
Bin_level_low	Bin number
Log_buffer_overflow	*None*
Temperature_out_of_range	Temperature
Power_up	*None*
Sale	Bin number

4.3 User Interface

4.3.1 GUI Design

For the purpose of this project, we will design the interface so that it can configure and monitor one vending machine. However, in future implementations, we will expand the

interface so that it can configure multiple vending machines. A screen shot of how the main program screen looks like is depicted in :
:

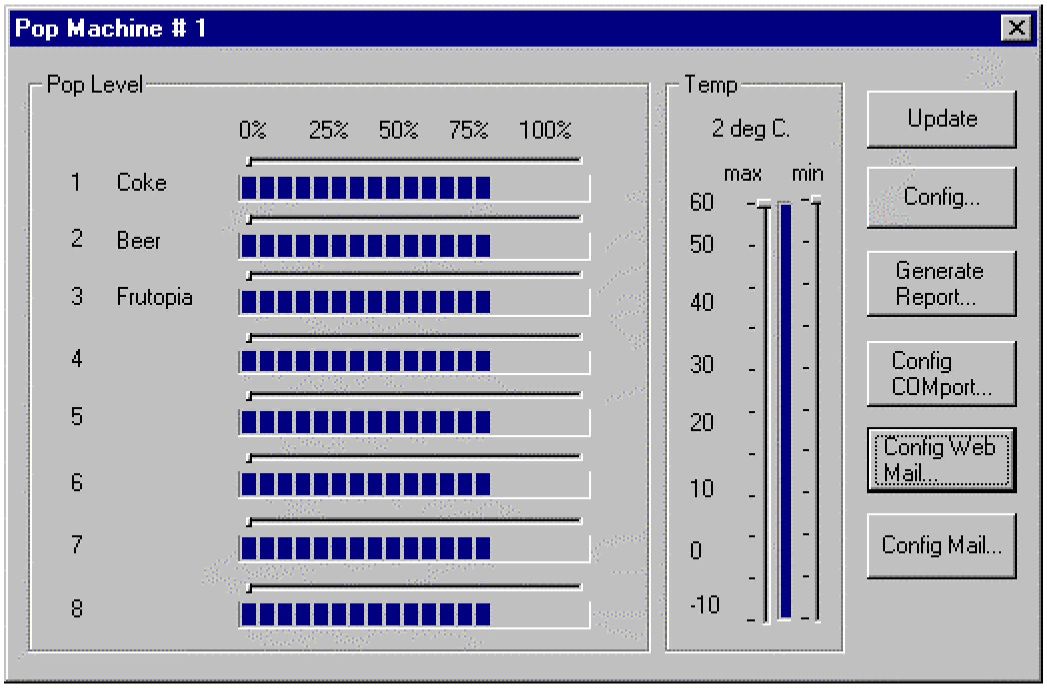


Figure 15: Program Main Screen GUI

This screen will allow the user to view the pop level and their minimum pop-levels. It will also allow them to view the current temperature inside the pop machine. An update button can be used used to force an uncheduled update of the status. Buttons are also provided to bring up the configuration screens to configure alarm settings, generate reports, configure the RS-232 port, configure the HTTP protocol, and configure the SMTP protocol.

4.3.2 Report Gereration

A user will be able to generate a comma delimited report file to be imported into Excel or another data base program. The user will be able to specify the date range, date format, and file name, the delimited file will be saved with. The delimited file will use the comma delimited format, with a carriage return between entries. Each record will have the following defaulted format:

<mm>/<dd>/<yyyy>,<hh>:<mm>,<bin #>

For now, this will be the only file format that will be available for the user.

5. Communications

Communications between the Host and the monitoring unit will take place over an RS-232 serial port connection. Using this medium in conjunction with the Vending Information Protocol (defined in the next section) allows the system to be easily converted to a different communications medium such as radio or standard computer network.

5.1 Vending Information Protocol (VIP)

The vending information protocol defines the set of instructions that can be used to communicate between the host and the monitoring unit as well as the procedures (protocol) used in the communications.

The vending information protocol utilizes variable length character delimited instruction set in the following format:

‘{‘ <Op-code> < operands> < close delimiter> ‘}’

where ‘{‘ is the open delimiter and ‘}’ is the close delimiter. Refer to Appendix B for a complete instruction set. The instructions can be of variable length up to a maximum of 8 characters. Any instruction exceeding this length will be ignored. The receiving units on either end of the communications link will be responsible for checking the validity of each instruction before allowing it to be processed by the application software. This validity check will:

- Insure the instruction contains one open and close delimiter in the positions specified above
- Insure the instruction has at least one character
- Insure the instruction contains no more than eight characters

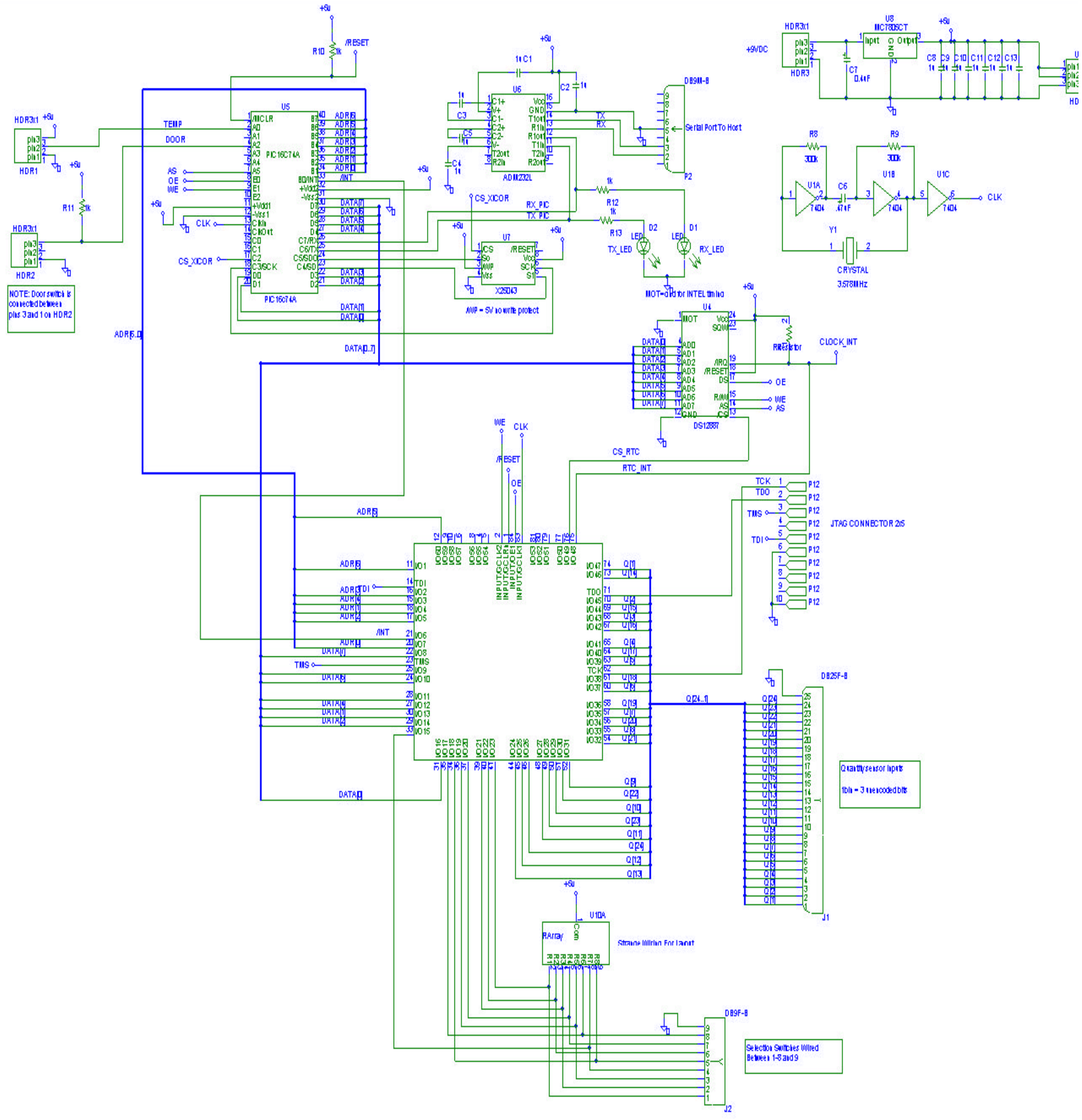
This helps to insure that any other noise or information being transmitted over the serial link will not be interpreted as an instruction.

Each sent command must be acknowledged with the {A} command before additional commands may be sent. If no response is received within 10 seconds, the command can be considered lost and may be re-sent.

All communications between the host and the monitoring unit are contained within sessions. When an entity has an instruction to send to the other it must first obtain a session by using the {O} command. Once the session has been obtained by means of the {AO} acknowledgment, the unit that opened the session becomes the “master” of the session. In this relationship the associated “slave” can only respond to instructions sent by the “master”. All sessions should be closed by the “master” using the {C} command. The VIP instruction set is listed in Appendix B.

Appendix A: Schematics

Monitoring Unit Schematics – See Next Page



U5 PIC16C74A
U6 ADM232L
DB9M-B
Serial Port To Host
U8 MC7805CT
+9VDC
HDR3x1
HDR1
HDR2
HDR3
NOTE: Door switch is connected between pins 3 and 1 on HDR2
U7 X25043
/WP = 5V no write protect
MOT=gnd for INTEL timing
U4 DS12887
CLOCK_INT
U1A 7404
U1B 7404
U1C 7404
CRYSTAL 3.578MHz
TX_LED
RX_LED
JTAG CONNECTOR 2x5
DB25F-B
Quantity sensor inputs
tbl = 3 unencoded bits
U10A
RArray
Strange Wiring For Layout
DB9F-B
Selection Switches Wired Between 1-8 and 9

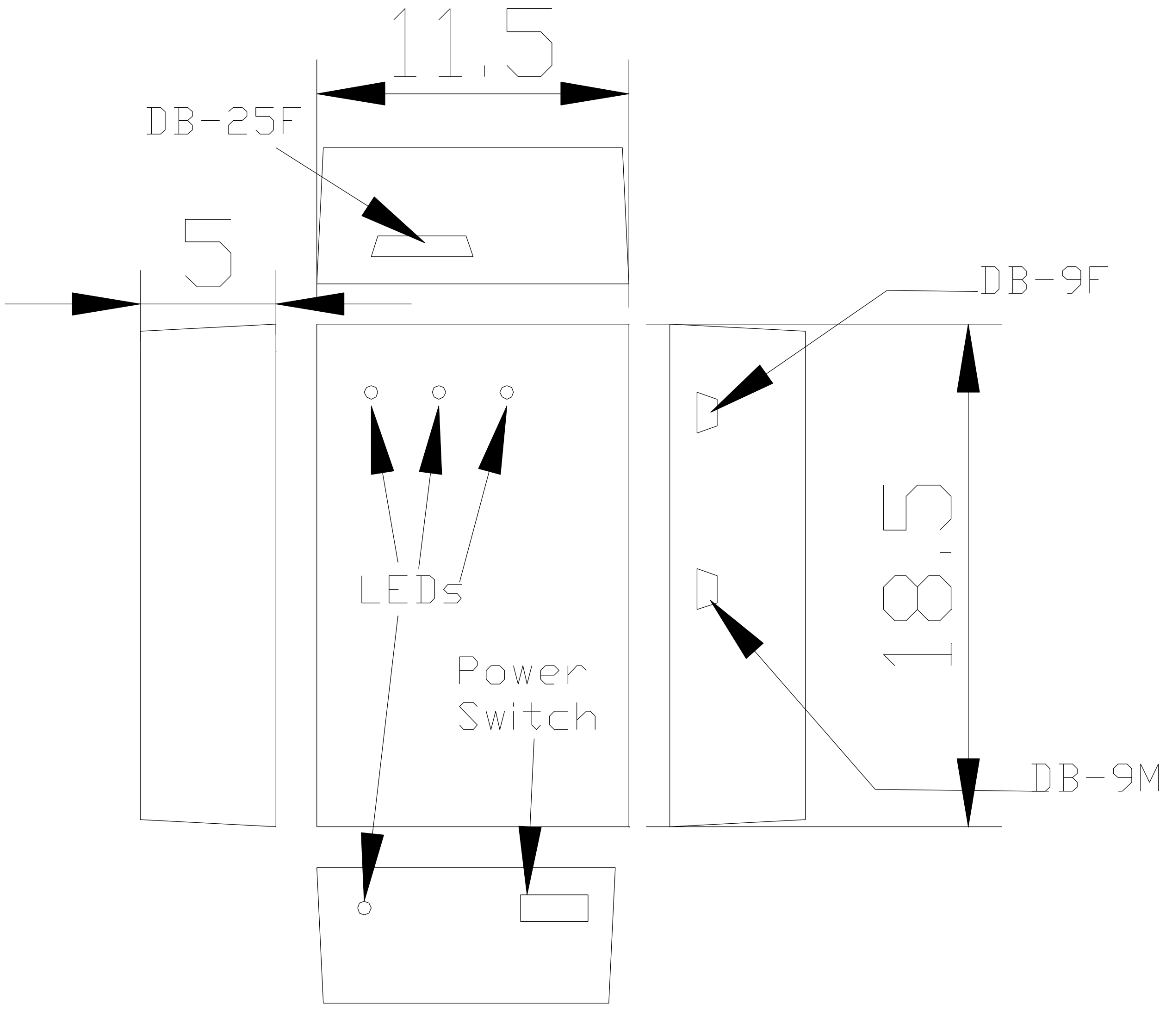
Appendix B: Vending Information Protocol (VIP) Instruction Set

Op-Code	Operand	Data Operands	Range	Description
A	Last Opcode successfully executed	None		**Acknowledgment** indicating that the last opcode sent was executed correctly: A reply must be received from the Monitoring Unit before any further instructions can be sent
E	Error Code	None		**Error** General error has occurred corresponding to the error number returned. See Error code definition
O	None	None		**Open Session** This command is use to open a new communications session. A new session must be opened between the Host and MU before any otther instructions will be executed. The entity that initiates this command will retain command over the session (see Protocol Definition)
C	None	None		**Close Session** This Closes the current communications session (see Protocol Definition)
R	**t** time (hours, minutes)	None		**Request Data** Instruction sent by Host to request data from a Monitoring Unit
		None		
	d selection data log	None		
	I time interval between reports	None		
	a alarm settings (on/off)	None		
	p pop level alarm setting	None		
	c temperature limits	None		
	y date (year, month, day)	None		

S	**t**	time (hours, minutes)	hours	0x00-0x17	**Set Data Field** Instruction sent by Host to MU to set configuratble paramters
			minutes	0x00-0x3B	
	I	time interval between reports	hours alarm	0x00-0x17 or 0xC0--0xFF	
			minutes alarm	0x00-0x3B or 0xC0-0xFF	
	a	alarm settings (on/off)	alarm byte	see alarm byte definition	
	p	pop level alarm setting	pop alarm level	0x00-0x1F	
	c	temperature limits	high temp limit	0x00-0x70	
			low temp limit	0x00-0x70	
	y	date (year, month, day)	year	0x00-0x63	
			month	0x01-0x1F	
			day	0x01-0x07	Sunday =1
Y	**t**	time (hours, minutes)	hours	0x00-0x17	**Reply with Data** This instruction is used by the MU to transfer requested data to the Host System
			minutes	0x00-0x3B	This instruction also serves as an acknowledgment to the R command
	I	time interval between reports	hours alarm	0x00-0x17 or 0xC0—0xFF	
			minutes alarm	0x00-0x3B or 0xC0-0xFF	
	a	alarm settings (on/off)	alarm byte	see alarm byte definition	
	p	pop alarm level	pop alarm level	0x00-0x03	
	c	temperature limits	high temp limit	0x00-0x70	
			low temp limit	0x00-0x70	
	y	date (year, month, day)	year	0x00-0x63	
			month	0x01-0x1F	
			day	0x01-0x07	

D	**None**		Bin Number month day hours minutes	0x00-0x0F 0x01-0x1F 0x01-0x07 0x00-0x17 0x00-0x3B	**Data Log Download** This instruction is used to transport a single selection data record from the Monitoring unit t the host. It must be acknowledged with AD before successive instructions can be sent. Sunday =1
P	MU ID	Monitoring Unit ID		0x00 - 0x7A	**Ping** Used to detect if a Monitoring unit with the specified ID is online This Command should be responded to only by the unit with the matching ID by the instruction {AP}
X	MU ID	None	None		**Reset MU** This instruction will cause a software reset in the MU with the specified ID
Z	MU ID	None	None		**MU Booted** This instruction is sent b an MU when it is first booted. It can be used by the Host to keep track of what Mus are online
L	**t**	temp out of range	temp	0x00-0x50	**Alarm Condition** An alarm condition has occurred which must be immediately reported to the host
	p	pop out of range	Bin Number Pop Level	0x00-0x0F 0x00-0x03	

Appendix C: Mechanical Drawings



All measurements are in cm.

Appendix D: Parts List For the Monitoring Unit

Description	Value	Quantity
Carbon Film Resistor, 5%	1k	96
Resistor SIP array, 9 pin	10k	1
PIC 16C74A microcontroller		1
DS12887 Real Time Clock		1
ADM232 RS-232 Driver/Receiver		1
X25043 512x8 serial EEPROM		1
MC7805CT 5V positive voltage reg.		1
Capacitor, Tantalum	0.1uF	7
Capacitor, Tantalum	0.1uF	5
Capacitor, Electrolytic	0.47uF	1
LED	Red	3
LED	Yellow	2
LED	Green	1
Altera EPM7128SLC84-7 EPLD		1
Panel mount power connector		1
DB-25 female connector		1
DB-9 male connector		1
DB-9 female connector		1
3-pin panel mount connector		2
Aluminum Box 18.5cmx11.5cmx5cm		1
LM50B temperature sensor		1
Single supply dual op-amp		1
Potentiometer (trim pot)	10k	10
Metal Film Resistor , 1%	10k	1
Metal Film Resistor, 1%	120k	1
Metal Film Resistor, 1%	56k	1
Normally open Single Pole Single Throw reed switches		9
OPB706A IR reflective object sensor		24
Single Supply High Gain Quad Op-Amp		8
LM339 Single Supply High Speed Quad Analog Comparator		8
1n4148 signal diode		24
Capacitor, 100nF		24
Capacitor, 100uF		24
Carbon Film Resistor, 5%	10M	24
Carbon Film Resistor, 5%	2k2	48
Carbon Film Resistor, 5%	100k	24