NC State University
ECE 480 Senior Design

Electronic Chemical Detector

Robert Johnson Matthew Youngblood

for
Dr. A. J. Attar
Appealing Products, Inc.
Raleigh, NC

May 12, 2003

Table of Contents

Executive Summary

The purpose of this project was to create a portable and small gas detector which can be used in the field to produce reliable information on one's exposure to gas (dosage), the concentration of the gas in the air, the duration of exposure, and other relevant data.

The detector operates off a card which is composed roughly of a fibrous material which will change color logarithmically with the presence of a certain gas. A photo sensor will detect this color change, and deliver a frequency signal to the CPU for analysis. The CPU will perform the desired operations on this signal and produce a number of different measures, depending on the device's current application.

The detector has an LCD and three buttons (select, enter, reset). The LCD prompts the user through a very intuitive calibration/initialization process and then displays dose and concentration information periodically.

The detector will alert the user through the use of visual and audible alarms if the dose or concentration exceeds a pre-determined level of safety. These levels of safety and also some of the terms which are used to compute dose and concentration will be reprogrammable in the field.

The detector also stores dose and concentration history in a 4Mb SRAM chip. This history is available for downloading through a serial connection using a program compiled in Visual Basic which allows for graphical visualization of the stored information.

The project was not fully completed to specifications. There is still a small bug in the memory read/write code. Due to time constraints, there was no attempt made to develop the field re-programmable feature.

Press Release

Colorimetric Technologies is pleased to announce a revolution in chemical gas protection safety in these times of uncertainty and unrest. The Color-Sense Personal Gas Detector is a standalone product which operates off of a nine volt battery and is sized to fit inside a shirt pocket or on a belt. It can detect up to 40 different gasses ranging from nerve agents to carbon monoxide.

The detector has an LCD and three buttons. The LCD prompts the user through a very intuitive calibration/initialization process and then displays dose and concentration information periodically.

The detector operates by monitoring an associated dosimeter card which will change color based on the presence of a certain gas. The detector electronically monitors the color change of the dosimeter. While such dosimeter cards can be used independently of the detector, the electronic detector has many advantages.

The detector will alert the user through the use of visual and audible alarms if the dose or concentration exceeds a pre-determined level of safety. These levels of safety and also some of the terms which are used to compute dose and concentration will be reprogrammable in the field.

The detector also stores dose and concentration history in a 4Mb SRAM chip. This history is available for downloading through a serial connection using a program which allows for graphical visualization of the stored information.

This product fulfills a huge demand in the market today for affordable personal, commercial, industrial or military solution to protect those who may be at risk of toxic chemical exposure or who simply want the peace of mind that can be afforded for them and their family with this simple device.

Project Report

1 Introduction

The purpose of this project was to create a portable and small gas detector which can be used in the field to produce reliable information on one's exposure to gas (dosage), the concentration of the gas in the air, the duration of exposure, and other relevant data.

The detector operates off a card which is composed roughly of a fibrous material which will change color logarithmically with the presence of a certain gas. A photo sensor will detect this color change, and deliver a frequency signal to the CPU for analysis. The CPU will perform the desired operations on this signal and produce a number of different measures, depending on the device's current application.

The detector has an LCD and three buttons. The LCD prompts the user through a very intuitive calibration/initialization process and then displays dose and concentration information periodically.

The detector will alert the user through the use of visual and audible alarms if the dose or concentration exceeds a pre-determined level of safety. These levels of safety and also some of the terms which are used to compute dose and concentration will be reprogrammable in the field.

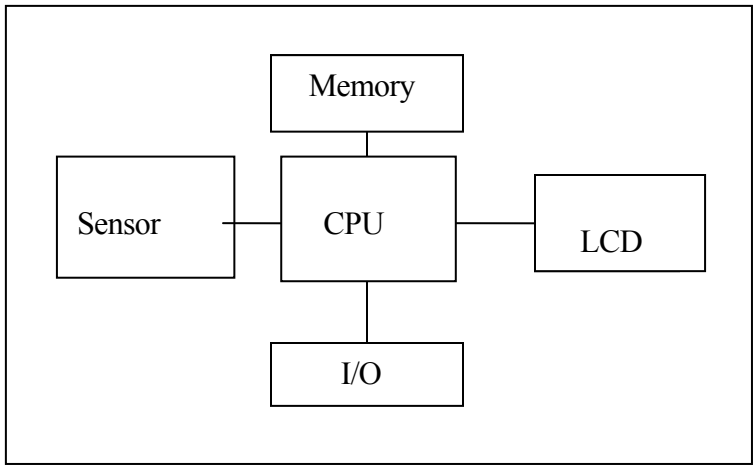
The detector also stores dose and concentration history in a 4MB SDRAM chip. This history is available for downloading through a serial connection using a program compiled in Visual Basic which allows for graphical visualization of the stored information.

The project was not fully completed to specifications. There is still a small bug in the memory read/write code. Due to time constraints, there was no attempt made to develop the field re-programmable feature.

2 Background

This project involves a portable and small gas detector which can be used in the field to produce reliable information on one's exposure to gas (dosage), the concentration of the gas in the air, the duration of exposure, and other relevant data.

The detector is composed roughly of a fibrous material which will change color logarithmically with the presence of a certain gas. A photo sensor will detect this color change, and deliver a voltage signal to an amplifying circuit which will be sent to the CPU for analysis. The CPU will perform the desired operations on this signal and produce a number of different measures, depending on the device's current application. A diagram is shown below.



Detector Block Diagram

2.1.1 Governing Equation

Where α, β1, and k are constants that are unique to the gas and the sensors,

$$\text{Concentration}(t) = \frac{e^{\left(-\frac{\beta 1 + \alpha \ln(k)}{\alpha}\right)} e^{\left(\frac{\text{Voltage}(t)}{\alpha}\right)} \left(\frac{\partial}{\partial t}\text{Voltage}(t)\right)}{\alpha}$$

and

$$\text{Dosage}(t) = \int_0^{\infty} \frac{e^{\left(-\frac{\beta 1 + \alpha \ln(k)}{\alpha}\right)} e^{\left(\frac{\text{Voltage}(t)}{\alpha}\right)} \left(\frac{\partial}{\partial t}\text{Voltage}(t)\right)}{\alpha} dt$$

2.1.1.1 α, β1, and k

The constants α, β1, and k are dependant on the particular photo sensor and amplifier. They are pre-calculated and incorporated into the tables of calculation coefficients.

2.1.1.2 Voltage(t)

In this device, voltage(t) is only a placeholder in the general dose computation algorithm. It is required to convert the gas data tables from an older device to use in this implementation. Voltage(t) corresponds to linearly to frequency(t) produced by the sensor chip.

$$\text{Voltage}(t) := 240 - \frac{125}{221}\text{Frequency}(t)$$

This signal will vary logarithmically with color change.

3 Project Requirements

3.1.1 General Operation

- (R-01) The device will detect the presence of a gas and report to the user the concentration of that gas and the dose to which the user has been exposed.
- (R-02) The device will detect gas presence through the use of a card which changes color per presence of gas
- (R-03) The device will have an optional attachment that will allow for the determination of the permeability of fabrics and other materials to a certain gas.
- (R-04) The device will store an exposure history
- (R-05) The device will sound an alarm when the concentration or dose exceeds a certain level
- (R-06) The user will be able to input user/trial-specific data

3.1.2 Platform

- (R-07) The device will have an alpha-numeric display which will display dose and concentration data to the user
- (R-08) The device will use a microprocessor to calculate and monitor dose and concentration values.
- (R-09) The device will communicate with an external computer to compute permeability information
- (R-10) The device will be capable of being re-programmed in the field
- (R-11) The device's programmable components will be installed in sockets
- (R-12) Photo-sensors will be mounted within a light-proof box to monitor the card's color change
- (R-13) The circuitry will be implemented on a printed circuit board

3.1.3 Normal Operation

- (R-14) The device will display current dose and concentration of a specific gas

3.1.4 Permeability Operation

- (R-15) The device will output dose and concentration to an external computer
- (R-16) The external computer will compute permeability data

3.1.5 Alarm Operation

- (R-16) If the dose or concentration exceed a safe level, the device will alert the user with audible and visible warnings

3.1.6 User Interface

- (R-17) The device will have a keypad through which the user may enter information

3.1.7 Packaging

- (R-18) The device will be small enough to fit inside a shirt pocket
- (R-19) The device will be rugged enough to withstand a fall from three feet.
- (R-20) The device housing will be able to mate to an optional attachment that will allow for the determination of the permeability of fabrics and other materials to a certain gas.

3.1.8 Power

- (R-21) The device will use a single 9V battery
- (R-22) The device will be operable for three days without changing the battery

3.1.9 Cost

- (R-23) The device will cost $30.00

4 Design Alternatives

4.1.1 Housing

All machining of the housing was contracted, and the main issues were development cost and time. After consulting with a lab on campus and outside commercial labs, it became readily apparent that work performed by a commercial lab would cost almost twice that of the NCSU facility. Since the NCSU facility (Precision Machine Shop in Burlington Labs) could guarantee reasonable turnaround times, the choice was quickly resolved.

4.1.1.1 Base Unit

The Project Sponsor (Dr. A. J. Attar) required the housing unit to be composed of a sturdy, yet light-weight plastic. A plastic box of suitable size was purchased at a retail supplier for under $10. Necessary millwork was performed by the Burlington shop on campus.

4.1.1.2 Detection Cell

According to Sponsor requirements, this was constructed out of heavy plastic material. Fabrication was done by Burlington shop.

4.1.2 Power Supply

The field detector unit will need to be battery operated, and had to be able to be in use for at least 24 hours before the batteries required changing.

Specialty batteries could be used for extended life, but the expense per unit and limited availability makes this choice impractical. The device will be designed for use with readily-available alkaline batteries. The battery options include

- One single 9V battery
- Four (4) 1.5V AAA cells

AAA cells have more capacity than the 9V battery (1,250 mA-h versus 650 mA-h). However, the nine volt battery fits into the device.

A voltage regulator will need to be used to keep the DC supply at a constant voltage level. Phillips semiconductor makes low-power voltage regulators in a variety of output voltages. Their SA57004 is a low-noise, low-power 150mA CMOS linear regulator designed for battery applications, with a cost of $0.40 each. Other vendors could be considered depending on availability.

4.1.3 Photosensors

A large variation in products was available. The most interesting was a new product recently released by Texas Advanced Optoelectronics Solutions (TAOS inc.). The product, TCS230, is a programmable color light-to-frequency converter on a single, surface mount CMOS IC. It combines multiple clear, red, green and blue sensors to give full visible light spectral response at high resolution.

The output is a 50% duty cycle square wave with a frequency directly proportional to light intensity. The digital outputs and control interface is designed for direct connection with a microcontroller. The device has low non-linearity error, a stable temperature coefficient and a small operating current with power-down mode.

The bulk rate per-unit cost of the device is $2.67, and free samples and promise of technical support from the company was obtained. This device was chosen.

The alternatives involve using multiple photosensors of differing spectral response, and will necessitate a different design methodology including physical construction of the detection cell as well as switching logic between the different sensors and light sources. By using the TCS230, these complications were avoided.

4.1.4 Light Source

The color film in the dosimeter needed to be illuminated for the photosensor to determine the color intensity. The manufacturer of the photosensor (TAOSinc) stated that a white light source should be used with the sensors.

Lumex makes such a device and several were ordered from Digikey at $3.00 apiece. Price breaks are of course given when buying in bulk. Other vendors had similar priced devices at similar operating characteristics.

4.1.5 Central Processing Unit (CPU)

While cost and size were always a consideration, the CPU needed to have certain capabilities. This included the ability to accurately perform timed input captures, communicate with a serial device, as well as sufficient memory space for program.

The Motorola 908 series microcontroller had these capabilities, was relatively low power operation, and cost only $5.50 per unit in bulk. While other microcontrollers were comparable, there were certain considerations to consider that made Motorola a better choice.

The main consideration is that one design team member (Robert) had previous experience designing projects with the Motorola 908 family processor, and is familiar with the development environment and the Instruction Set Architecture. Also there is support for this development environment here on campus, as this device is used in a popular ECE course. Technical resources include the professor of this course, Dr. John Sutton, and the Senior Design TA, David Hoff.

Choosing the Motorola 68HC908GP32 – a stable microcontroller with high functionality – will significantly reduce development time by avoiding the time spent learning a new development environment.

The main problem with most microcontrollers, including this one, is that they are not very capable of doing complex floating point math operations, such as calculating exponentials. While exponential calculations could be done using numerical methods (Taylor Series), this would be difficult to program, and the computationally intensive – thereby consuming more power.

A means to effectively perform exponential calculations would require a DSP. This would add significant complexity to the design, as well as cost and power consumption

An option to performing exponential calculations is to store exponential values in a lookup table. This would be cheaper, consume less power, and not increase the complexity of the design. This option was chosen.

4.1.6 LCD Output

An 8x2 LCD alpha-numeric display was chosen.

4.1.7 Memory (SRAM)

RAM Memory was needed to store the specific user information such as name, location, etc., as well as concentration and dose (exposure) data collected over a period of time that could be up to three days. This data is intended to be later download into a PC-based application. The microcontroller did not have enough RAM space to adequately store this amount of data, so off-chip SRAM had to be used.

The chip which was used is the Hitachi HM 628512C and is a 4MB 5V SRAM chip. This chip was obtained as a free sample. Other options are detailed.

Hynix

Org.	Part Number	Density	Voltage	Speed	PKG	Power
x8	HY62UF08401C	4M	3.0V	55ns	Small TSOP-I	15mW active 18uW standby

Mitsubishi

Part Number	Memory Organization	Access Time (ns)	Package Type	Supply Voltage (volt)	Power
M5M5V408BFP-85H	512K x 8	85.0(max)	32pin 525mil SOP	2.7(min) 3.6(max)	15mW active 0.9uW standby

Hitachi

Memory capacity(bit)	Organization (bit)	Part No.	Package	Access time (ns)	Supply voltage(V) (min. to max.)	Power
4M	x 8	HM62V8512BLFP-7	SOP(32)	70	2.7 to 3.6	15mW active, 3uW standby
4M	x 8	HM62V8512CLFP-5	SOP(32)	55	2.7 to 3.6	6mW active, 2.4uW standby

4.1.8 Serial Communications Interface

Two options for serial communications with the PC were using a standard RS-232 interface with a DB-9 connector, and using a USB interface.

The RS-232 was known to be the easiest to implement. Although it is slower than the USB, the speed difference was not an issue in this application. What could have been an issue however, is the bulky nature of the DB-9 connections, especially in the portable field unit. A USB connector is much easier for the end user. Also, the laboratory based computer may need the RS-232 port for other applications, and a USB interface will be more convenient for this application as well.

However, since the USB interface is more difficult to implement than the RS-232, we first developed an RS-232 interface to make sure the overall serial Microcontroller-to-PC communication was established.

The device required a serial communications interface adapter. This could have been synchronous or asynchronous. Asynchronous communication was easier to implement and was entirely sufficient for this application. Many vendors manufacture such adapters, often called UART's (Universal Asynchronous Recevier Transmitter) and there are low-power 3VDC versions.

The RS-232 application required an RS-232 Driver to convert the 1's and 0's (5V and 0V) pulses into -12 VDC and +12 VDC signals. The Maxim RS232 chip was chosen.

5 Design Specifications

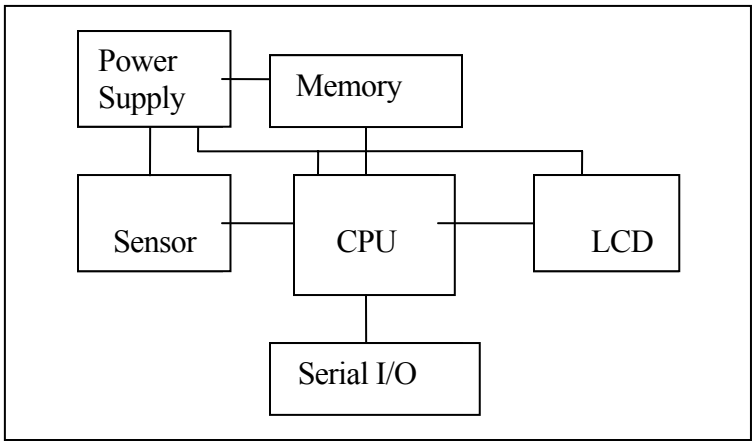
5.1.1 Description

The Colorimetric Chemical Detector (CCD) is a portable and small gas detector which can be used in the field to produce reliable information on one's exposure to gas (dosage), the concentration of the gas in the air, the duration of exposure, and other relevant data.
The detector is composed roughly of a membrane which will change color logarithmically with the presence of a certain gas. A photo sensor will detect this color change, and deliver a voltage signal to an amplifying circuit which will be sent to the CPU for analysis. The CPU will perform the desired operations on this signal and produce a number of different measures, depending on the device's current application.

5.1.1.1 Features

- Motorola 68HC908GP32 (8 bit, 2MHz)
- 4Mb SRAM
- 2 line, 8 character LCD display
- 24 gas selector cards
- Serial port
- Audio-visual alarm settings

5.1.1.2 Block Diagram



CCD Block Diagram

5.1.1.3 Performance

The CCD samples the air (by checking the card's color change) every minute.

5.1.1.4 User Interface

The user will be presented with an alpha-numeric keypad and a two line, ten characters wide LCD.

5.1.1.5 Reliability and Maintainability

The CCD will be rugged enough to withstand a three-foot drop test. The code and computation parameters will be field-upgradeable through the serial port.

5.1.1.6 Safety and Failure Modes

The CCD will not operate in a safety mode.

5.1.1.7 Operational Environment

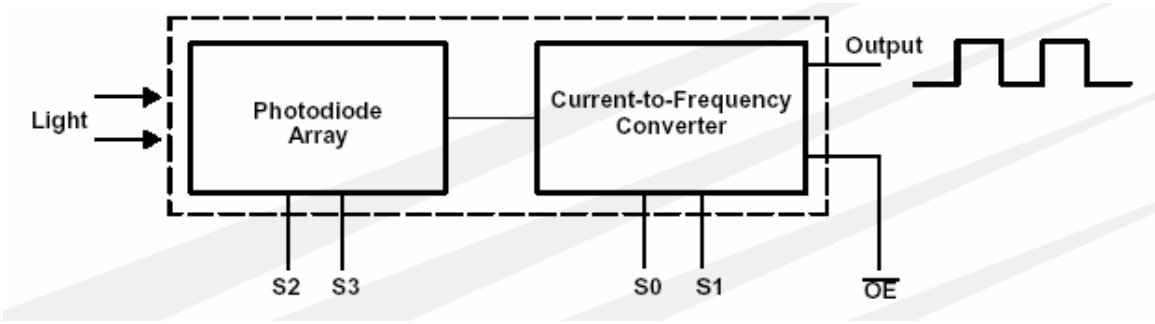
The CCD will operate in normal room temperature.

5.1.2 COMPONENTS

5.1.2.1 Sensor

The sensor used is a TCS230 programmable color light-to-frequency converter. It combines configurable silicon photodiodes and a current-to-frequency converter on single monolithic CMOS integrated circuit. The output is a square wave (50% duty cycle) with frequency directly proportional to light intensity (irradiance). The full-scale output frequency can be scaled by one of three preset values via two control input pins. Digital inputs and digital output allow direct interface to a micro controller or other logic circuitry. Output enable (OE) places the output in the high-impedance state for multiple-unit sharing of a micro controller input line.

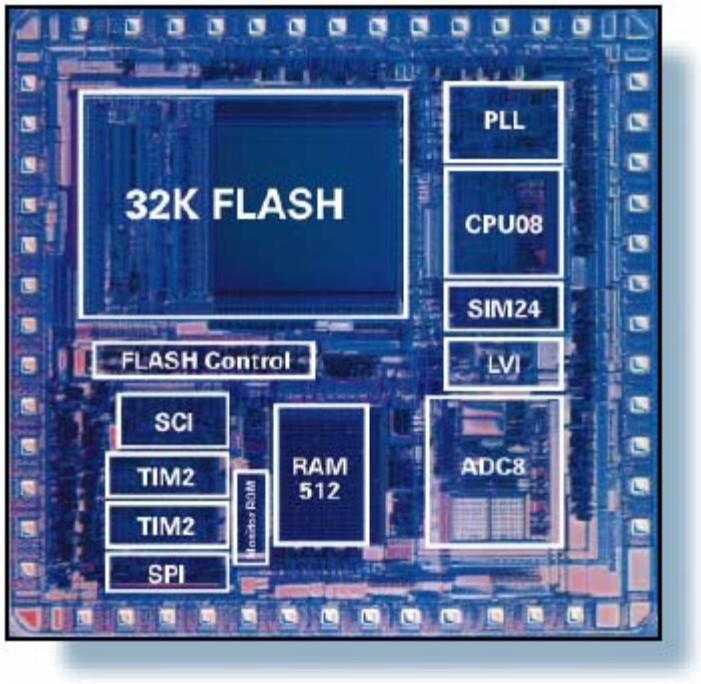
The light-to-frequency converter reads an 8 x 8 array of photodiodes. Sixteen photodiodes have blue filters, 16 photodiodes have green filters, 16 photodiodes have red filters, and 16 photodiodes are clear with no filters. The four types (colors) of photodiodes are inter-digitated to minimize the effect of non-uniformity of incident irradiance. All 16 photodiodes of the same color are connected in parallel and which type of photodiode the device uses during operation is pin-selectable. Photodiodes are 120 mm x 120 mm in size and are on 144-mm centers.



Sensor Block Diagram

5.1.2.2 CPU

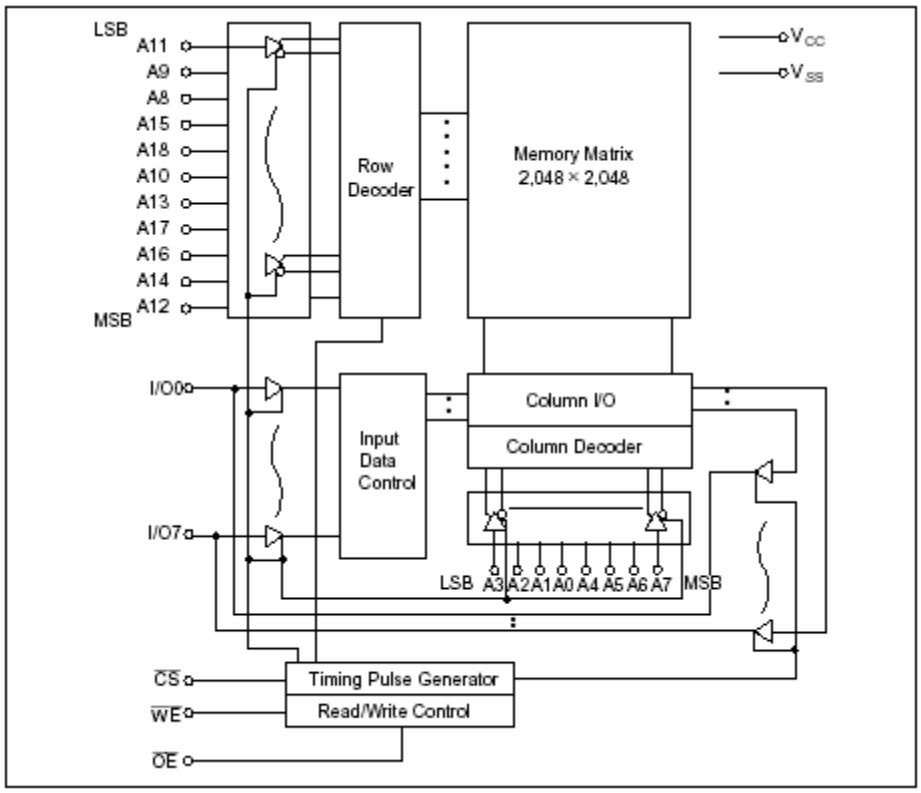
The Motorola 68HC908GP32 is used in the CCD. It provides a highly integrated 8-bit FLASH micro controller (MCU) solution. The 8HC908GP32 builds on the success of the 68HC05 family by offering a code compatible migration path to higher performance FLASH MCUs.



CPU Block Diagram

5.1.2.3 Memory

The Hitachi HM 628512C is a 4-M bit static RAM organized 512-kword x 8-bit. It realizes higher density, higher performance and low power consumption by employing CMOS process technology (6- transistor memory cell). The device is packaged in a 525-mil SOP. The HM 628512C is suitable for battery backup system.



Memory Block Diagram

5.1.2.4 Power Supply

The CCD will use one nine volt battery.

5.1.2.5 LCD, Speaker, Serial Connector

The CCD will use a standard 2x8 character display LCD, speaker, and serial connector, obtainable from a local computer parts store.

5.1.2.6 LED

The CCD will use two LEDs to illuminate the reflective color card. The type of LED used in the CCD is 5mm ultra white water clear.

5.1.2.7 Enclosure

The enclosure has a pop-out end panel, plus compartment for batteries. 2-3/4x4-5/8x1".

5.1.2.8 Sensor Cell Block

The sensor cell block has two card viewing apertures for the sensors to fit into in order to measure color change. Please refer to appendix I for a detailed drawing.

6 Design Description

6.1 Printed Circuit Board

The Printed Circuit Board was designed in Capture CIS as a schematic and the resulting netlist was exported to ORCAD Layout. Please refer to the Appendix G to see a copy of the schematics. All of the footprint creation and routing was done in this software. The layout files are included in the PCB directory on the CDROM.

6.1.1 ORCAD Layout: Component Footprints

Footprints that did not immediately map to ORCAD's database had to be constructed or assigned for the following components. These footprints are included in the footprint library folder in the PCB directory of the CDROM.

6.1.1.1 Voltage Regulator, Potentiometer

The voltage regulator and potentiometers required three drill-holes. There was no available footprint that fit these components so the standard eight header footprint, WALCON.100/VH/TM2OES/W.325/14, was edited so that only the first three pins remained. This footprint was titled VREG.

6.1.1.2 LCD

The LCD used the WALCON.100/VH/TM2OES/W.325/14 footprint.

6.1.1.3 LEDs, Speaker, and Buttons

The LEDs, speaker, and buttons were all given the same RAD/.150X.100/LS.100/.031 footprint.

6.1.1.4 Serial, Power Connections

The serial and power connections used the testpoint TP footprint in order to maximize final wiring options when the board was placed inside the housing.

6.1.1.5 Memory Chip

The memory chip's footprint was the most difficult to create. It was based off of the SOG.050/32/WG.420/L.500 footprint. The distance between the two sets of pads had to be widened to a distance of 11.3mm.

6.1.2 ORCAD Layout: Routing

6.1.2.1 Trace Width/Spacing/Length

Because the board was two-sided and extremely crowded, the PCB manufacturer's minimum trace width and spacing of 10mils was used. Trace length was not an issue because there were no high frequency components or special need to have very high signal quality.

6.1.2.2 Double Sided

In order to procure fast turnaround time and to minimize costs, a double sided PCB setup was used.

6.1.2.3 Drill Sizing

Four drill sizes were used and are summarized below in the table.

Drill Size (mils)	*Quantity*
280	212
310	18
350	12
360	5
420	14

6.1.3 ORCAD Layout: Component Positioning

Certain components had critically important positioning requirements due to placement in box or signal crosstalk.

6.1.3.1 Sensors

The sensors had the most strict place requirements, due to the fact that they needed to align with the sensor-card viewing apertures built into the housing.

6.1.3.2 Capacitors

Capacitors placed on the supply voltage for each component were not placed optimally because of space issues. The power lines on the latches ran directly underneath their respective chips. This did not affect the operation of any component.

6.1.3.3 Crystal Timing Circuit

The crystal timer circuit was placed as close as possible to the GP32 in order to minimize interference in the clock frequency. The trace length between the crystal and the GP32 is 1.34".

6.1.4 PCB Lamination

The laminate used was FR-4 with a thickness of 0.062". It meets a broad range of thermal and electrical requirements. It has processing characteristics consistent with industry FR-4's and uses high quality woven E-glass, copper foils and resins.

6.2 Components

The following is a discussion on the different components used in the design and their operation, connectivity to the rest of the system, and other salient issues.

6.2.1 Photo-detector

6.2.1.1 Operation

(From the tcs230 datasheet)
The TCS230 programmable color light-to-frequency converter combines configurable silicon photodiodes and a current-to-frequency converter on single monolithic CMOS integrated circuit. The output is a square wave (50% duty cycle) with frequency directly proportional to light intensity (irradiance). The full-scale output frequency can be scaled by one of three preset values via two control input pins. Digital inputs and digital output allow direct interface to a microcontroller or other logic circuitry. Output enable (OE) places the output in the high-impedance state for multiple-unit sharing of a microcontroller input line.

The light-to-frequency converter reads an 8 x 8 array of photodiodes. Sixteen photodiodes have blue filters, 16 photodiodes have green filters, 16 photodiodes have red filters, and 16 photodiodes are clear with no filters. The four types (colors) of photodiodes are interdigitated to minimize the effect of non-uniformity of incident irradiance. All 16 photodiodes of the same color are connected in parallel and which type of photodiode the device uses during operation is pin-selectable. Photodiodes are 120 mm x 120 mm in size and are on 144-mm centers.

6.2.1.2 Settings

S0	S1	OUTPUT FREQUENCY SCALING (f_o)		S2	S3	PHOTODIODE TYPE
L	L	Power down		L	L	Red
L	H	2%		L	H	Blue
H	L	20%		H	L	Clear (no filter)
H	H	100%		H	H	Green

Sensor Settings

6.2.1.3 Connectivity

The output of the sensor was connected to port D4. Switches S0, S1 were connected high. Switches S2 and S3 were connected to Ports D1 and D2.

6.2.1.4 Empirical Evaluation

The sensor was empirically evaluated using a project box. The sensor was mounted on a small surface mount board and placed on the housing, at the card viewing aperture. A card was placed inside the device and the setup was placed inside a relatively light proof project box. The output frequency was then measured and graphed. Please refer to Appendix J for the data.

6.2.1.4.1 Surface-mount Boards

The sensor chip had to be soldered onto small surface mount boards. This required the use of a microscope and sophisticated laboratory equipment.

6.2.1.4.2 Differences in Placement

There were significant differences that occurred when the sensor position was varied. By raising the sensor only 1/8" from the opening (maintaining light proofing), the output frequency was attenuated to a level that was too low to work with (220kHz, rather than 250kHz)

6.2.1.4.3 Color Cards

Color cards were constructed with approximately the same specifications as the true gas cards in order to test the response of the sensor chip. The only difference between the construction of the gas cards and the test color cards was that the gas cards have a small piece of laminate over the colored area, whereas the test cards did not. Although this did create a difference between the gas and color cards, the difference was not critical, because all of the testing was done with the same type of card, thereby normalizing the discrepancy.

6.2.1.4.4 Design Considerations

According to device specifications, the trace length for the output signal should be less than one foot in length. This was not an issue, given the size of the PCB.

6.2.2 LCD

6.2.2.1 Operation

The LCD works similar to a latch. Data is placed on the data lines and is then strobed into the LCD using the enable line.

6.2.2.2 Settings

In order to save pins on the GP32, the LCD was connected in four bit mode. In this setup, data and commands are clocked into the LCD four bits at a time, rather than all eight bits at once. 4-bit mode or full mode options programmed into the LCD by software.

6.2.2.3 Connectivity

The LCD's data lines were connected to ports C1-C4. The LCD's data bus was 8 bits wide, but only 4 bits of that bus were used. The other connections can be seen on the connection spreadsheet included in the Appendix K.

6.2.3 CPU: MC68HC908GP32

The Motorola MC68HC908GP32 processor is a versatile 8 bit micro-processor that has found many applications throughout the embedded industry. It programs very quickly and has a large 32K ROM

.

6.2.3.1 Operation

The GP32 is essentially plug and play in terms of hardware. Because of its robust construction and design, there were no signal quality or configuration issues in our design.

6.2.3.2 Settings

There were no special settings needed in hardware for the GP32.

6.2.3.3 Connectivity

The full connection list for the GP32 can be found in Appendix K.

6.2.3.4 Crystal Circuit

The crystal circuit used a 32.768kHz crystal. The GP32 PLL took this and converted it to an 8MHz internal frequency.

6.2.3.5 In-Circuit Programming

The programming for the GP32 was originally done on a wire-wrapped board that was constructed at the beginning of the semester. This board developed a short, however, and so a professional programming POD was used to program the GP32.

6.2.4 Memory

6.2.4.1 Operation

The Hitachi HM 628512C memory chip was used in this design. It employs SRAM technology which minimizes power consumption by eliminating the need for a strobe or clock.

6.2.4.2 Settings

There were no special settings needed for the memory chip.

6.2.4.3 Connectivity

The memory chip's address lines were connected to ports B0-B7 and C0-C7. The data lines were connected to two data latches in parallel. This was done to conserve pins on the processor. For detailed information, please review section 1.2.6.

6.2.5 Latches

6.2.5.1 Operation

(From the datasheet)
The DM74LS374 8 bit latch was used. These 8-bit registers feature totem-pole 3-STATE outputs designed specifically for driving highly-capacitive or relatively low-impedance loads. The high-impedance state and increased high-logic level drive provide these registers with the capability of being connected directly to and driving the bus lines in a bus-organized system without need for interface or pull-up components. They were used to interface the memory to the GP32.

The eight latches of the DM54/74LS373 are transparent D-type latches meaning that while the enable (G) is high the Q outputs will follow the data (D) inputs. When the enable is taken low the output will be latched at the level of the data that was set up. The eight flip-flops of the DM54/74LS374 are edge-triggered D-type flip flops. On the positive transition of the clock, the Q outputs will be set to the logic states that were set up at the D inputs. A buffered output control input can be used to place the eight outputs in either a normal logic state (high or low logic levels) or a high-impedance state. In the high-impedance state the outputs neither

load nor drive the bus lines significantly. The output control does not affect the internal operation of the latches or flip-flops. That is, the old data can be retained or new data can be entered even while the outputs are off.

6.2.5.2 Settings

There were no special settings needed to operate these components.

6.2.5.3 Connectivity

Both latches had 0.1uF capacitors placed on their power supplies. Please refer to section 1.2.6 for a description of the latch/memory connectivity.

6.2.6 Memory & Latch Combination

6.2.6.1 Description of Connectivity and Operation

The latches were used as a buffer between the GP32 and the memory chip in order to conserve pins on the GP32. By using the latches as temporary storage for data, the address bus was able to be shared with the data bus.

Ports B0-B7 and C0-C6 were used in the memory/latch setup. Port B was shared by three devices, the two latches and the memory chip. One of the latches served as the latch which held a byte of data to be written. That latch will be called latch_in. The second latch held a byte of data that was read from memory. That latch will be called latch_out. The output of latch_in was connected to the data bus of the memory chip. The input of latch_out was also connected to the data bus of the memory chip.

In order to write to memory, the byte to be written was placed on Port B. The latch_in was then enabled and the data was clocked into the chip. After the byte was safely docked inside latch_in, the 15 bit address to which the byte was to be written in memory was placed on ports B and C, then the memory chip CS_N (chip select) was set low, enabling the memory chip. The WE_N (write enable) line was set low, enabling writes, and the output of latch_in was turned on, placing the byte directly into the correct location in the memory chip. The memory chip was then deselected, and the latch_in output turned off.

In order to read from memory, the address to be accessed was placed on ports B and C. The memory chip was then selected, and the OE_N (read enable) was set low, enabling reads from memory. At this point, the byte to be read was sitting on the output of the memory and on the input to latch_out. The latch_out chip was then turned on and the byte was clocked into the latch. To get the byte out of latch_out, port B was changed from an output to an input and the byte read from the port. For more details on this, see the section 6.5.14.

6.2.7 RS232

6.2.7.1 Operation

The Maxim RS232 chip was used to convert a normal signal from the GP32 to a +/- 15V signal for use with a serial cable.

6.2.7.2 Settings

There were no special settings on this chip.

6.2.7.3 Connectivity

The Maxim 232 required four 0.1uF capacitors connected to it in addition to a 0.1uF capacitor on the power supply.

6.2.8 LEDs

6.2.8.1 Operation

The LED used was a clear white LED with a 60 degree viewing angle.

6.2.8.2 Settings

There were no settings for the LED.

6.2.8.3 Connectivity

A 300 ohm potentiometer was used to limit the current. The device was calibrated with 17mA through the LED. This corresponded to 294 ohms on the POT, with a five volt power supply.

6.2.9 Speaker

6.2.9.1 Operation

The speaker was a thin ceramic piezo-electric speaker. It provided the loudest signal when a 3.7kHz wave was placed on its inputs.

6.2.9.2 Settings

There were no special settings for this device.

6.2.9.3 Connectivity

The speaker was connected to Port D7, which is a special output of the GP32 and can be programmed to output a specified frequency.

6.2.10 Buttons

6.2.10.1 Operation

There were three buttons on the device. A reset, and two buttons connected to ports A0 and A1, which double as keyboard inputs on the GP32.

6.2.10.2 Settings

There were no special settings for the buttons.

6.2.10.3 Connectivity

The buttons were all connected to 1k pullup resistors. They were not debounced in hardware because a software delay was used to filter out jitter.

6.2.11 Serial Connection

6.2.11.1 Operation

The serial connection was only three test points for Transmit, Receive, and Ground.

6.2.11.2 Settings

There were no settings needed for the serial connection.

6.2.11.3 Connectivity

The transmit point was connected to pin 14 (T1OUT) of the RS232. The receive point was connected to pin 13 (R1IN) of the RS232.

6.2.12 Power Supply

6.2.12.1 Operation

The power supply was a nine volt battery. It was connected to two test points on the PCB, PWR and GND. The PWR point was then connected to the input of a 7405 voltage regulator and the output of that regulator was connected to VDD for the board.

6.2.12.2 Settings

There were no settings for the power supply.

6.2.12.3 Connectivity

Three decoupling capacitors were used to filter out low and high frequency noise. They were 10pF, 0.1uF, and 10uF. Viewed on a scope, the power and ground lines were both very clean.

6.3 Software

6.3.1 Memory Map

To see the memory map of the GP32, please refer to Appendix E.

6.3.2 Initializations

Before the Motorola MC68HC908GP32 Microcontroller (henceforth referred to as the '908') can use certain functions or interface with other components, specific initialization procedures must be engaged.

Many of these initializations modify internal operation of the Motorola 908, and therefore should be done immediately upon startup or reset. Other initializations are to set up external devices. These devices have their own particular specifications and requirements, and are being addressed by the particular initialization sequence.

6.3.3 Configuration

The CONFIG registers are accessible only once during normal operation, and so must be set correctly at program startup. Two functions of the CONFIG registers concern us: The Watchdog Timer and the Serial Communication Baud Rate Clock Source.

The Watchdog Timer must be disabled to prevent unwanted resets occurring because "nothing" happens within a certain time period. While a Watchdog Timer is

beneficial for routines that are prone to crash or lock up, our code is bold and courageous.

The SCI Baud Rate must be changed to select the internal bus frequency as SCI baud clock source, rather than an external clock.

```
        lda     #$01            ; Kill the dog
        ora     CONFIG1
        sta     CONFIG1
        lda     #$01
        ora     CONFIG2         ; set SCI for internal clock
        sta     CONFIG2
```

Since these two changes are so easily accomplished, they are actually incorporated in the Timer1 initialization routine for compactness.

6.3.4 PLL

The Motorola 908 must have some clock source from which the internal bus clock is driven or derived. A high-frequency crystal oscillator/resonator can generate a constant crystal frequency clock, or the internal Phase-Locked Loop submodule can generate a Voltage-controlled oscillator frequency clock using a 32 kHz crystal. The selected frequency clock then becomes the base for the internal bus clock.

This INIT routine engages the Phase Lock Loop Registers that control the clock generation and selects frequency clock for the microcontroller.

The PLL must first be disabled before changing the registers. The PWBC Register is set to allow automatic bandwidth control and the PLL Registers are set so that an 8 MHz internal bus clock is derived from a 32.768 kHz crystal. This is accomplished specifically by setting the PLL Prescaler to 1, the Exponent to 2, the PLL Multiplier to `0h03D1`, the VCO range select to `0hD0`, and the PLL Reference Divider to 1.

Once the frequency is set, the PLL is enabled, and a wait routine makes sure the PLL stabilizes before continuing. After stabilization occurs, the clock source select switch is changed from external oscillator to the internal VCO.

```
      bclr   5,PCTL           ; Disable PLL
      mov    #$02,PCTL        ; (Prescaler=1), (VPR) = 2
      mov    #$80,PBWC        ; Automatic bandwidth control
      mov    #$03,PMSH        ; Upper byte of $3D1 = PLL
                              ; multiplier (N)
      mov    #$D1,PMSL        ; Lower byte of $3D1 = PLL
                              ; multiplier
      mov    #$D0,PMRS        ; VCO range select (L) = $D0
      mov    #$01,PMDS        ; PLL reference divider (R) = 1
      bset   5,PCTL           ; Enable PLL
wait_here:
      brclr  6,PBWC,wait_here ; wait until PLL stabilizes
      bset    4,PCTL          ; switch clock source to PLL
```

It is important to make the PLL initialization as soon as possible, as the bus frequency directly impacts every operation of the microcontroller. For instance, if a "delay" is called (see functions, section 2.4, below) before the PLL is initialized, the pause will actually be in effect 250 times longer than it is supposed to. (32 kHz : 8 MHz = 1:250)

6.3.5 Timer1

Of the two timers available, Timer 1 is given the responsibility of timing the input captures from the photo-sensor output as well as timing the delays between each pulsed "look" at the card.

Because the photo-sensor will "look" at the card 8 separate times in 1 second, the Timer1 needs to overflow and cause an interrupt once every 0.125 seconds. For our 0.125 us bus clock, this is done by setting the Tim1 Prescaler Divisor to 16 and the Timer1 Modulus to `0hF424` (decimal 62,500).

The Timer1 Status and Control Register is set so that Timer1 Channel 0 (PortD pin4) is set for a rising-edge input capture. The interrupt enable is left off for the time being, but will be enabled whenever the microcontroller is ready to actively "look" at the card via the photo-sensor. The Timer1 Overflow Interrupt enable is likewise left off for the time being, since this will be turned on and off within the program as necessary.

```
mov    #$34,T1SC      ; stop, reset, /16
clr    T1CH0H         ; clear TIM1 Ch0
clr    T1CH0L
mov    #$F4,T1MODH    ; set TIM interrupt at xF424 "tics"
mov    #$24,T1MODL
mov    #$04,T1SC0     ; input capture, rising edge
```

6.3.6 Timer2

Timer 2 is used to generate an output Pulse Width Modulated signal of a certain frequency to be used as the signal to the alarm speaker. The initialization sets the timer to generate a PWM signal of 50% duty cycle and 4 kHz frequency. This value could be changed to generate other pitches or changing pitches if desired.

```
mov    #$30,T2SC       ;Divide by 1 (no prescale)
mov    #$07,T2MODH     ;
mov    #$D0,T2MODL     ;$07D0 @ 0.125us = 4 kHz
clr         T2CH1H
mov    #$80,T2CH1L     ;$0080 = 50% duty cycle
mov    #$16,T2SC1      ;PWM output, toggle, no interrupt
```

An interrupt is not necessary because when the timer is enabled it toggles the Tim2 Channel 0 (PortD pin6). When the timer2 is not enabled the Tim2Channel0 pin is

held low. Therefore, when an alarm is to be sounded all that needs to be done is to enable the Timer2.

6.3.7 Keyboard

Since there are only two buttons for user input at this time, there are only two Keyboard ports that need to be initialized. This routine specifies the Keyboard Ports to be used (A0 and A1) and enables both of them to cause interrupts when engaged. Additionally the interrupts are set to occur during both high to low transitions on the port pin as well as logic low.

6.3.8 Serial Communication Interface

The SCI is internal to the Motorola 908, and allows serial communication with another microcontroller, PC, or other device capable of serial transmission and/or receive. The initialization routine is used to determine the baud rate and enables different functions.

For our purposes, all we need is to enable transmit, set the baud rate to 9600 baud, and specify one stop bit and no parity bit. This is accomplished by setting the SCC1 and SCBR registers accordingly. If receive was needed to collect incoming data from another device, SCC2 would be set to enable the receive function.

6.3.9 LCD

The LCD is an external device that must be programmed so that it can accept data from the Motorola 908 and be able to interpret it accordingly. All initializations done in this routine cause logic signals on Port Pins tied to the LCD to initialize the LCD correctly, according to its particular specifications. Nothing in this initialization affects internal operation of the Motorola 908.

The LCD is wired to the 908 in "4-bit" configuration, which means that each byte of data has to be transmitted one nibble at a time, first the high-order, then the low.

- Write $03 to the LCD three times to enter initialization process
- Write $02 to the LCD to indicate 4-bit transmission mode
- Write $28 to the LCD for two-line display
- Write $0C to the LCD to turn display on and cursor off
- Write $06 to the LCD for memory direction (left to right) and display (no shift)

The LCD is now ready to accept nibblewise data and print the characters to its display in the format required by the project specifications.

6.3.10 Registers

This is where the Port Pins are declared to be input or outputs, and whether or not the input pins have an internal pullup resistor engaged.

```
mov   #$FF,DDRB     ; set ports B, C as output
mov   #$FF,DDRC     ;
mov   #$FC,DDRA     ; set ports A0, A1, E1 as inputs
mov   #$01,DDRE
mov   #$EF,DDRD     ; set D4 as input, rest output
```

Also, user registers (RAM) are initialized to proper values before beginning the program. This routine does not affect internal operation of the 908.

6.3.11 Memory

The external memory must be ready to accept data writes from the Motorola 908 during the program, and to dump the data contents back to the 908 when requested. This initialization ensures the memory and latches are turned off and/or disabled upon program start.

Memory		Latch In		Latch Out	
Mem	908	Latch	908	Latch	908
CS	D1	OE	A5	OE	A3
OE	D2	E	A4	E	A2
WE	A6	D0-D7	B0-B7	Q0-Q7	B0-B7

- Memory and Latch Connectivity

```
bset  1,PTD              ; mem chip select - off
bset  2,PTD              ; mem output enable - off
bset  6,PTA              ; mem write enable - off
bset  3,PTA              ; disable output of latchIn
bclr  2,PTA              ; disable writing to latchIn
bset  5,PTA              ; disable output of LatchOut
bclr  4,PTA              ; disable writing to latchOut
clr   memory_add
mov   #$20,memory_add+1  ; start memory address 20 bytes
                         ; forward.
```

6.3.12 IRQ

The IRQ (interrupt request) is an external interrupt that is initiated by bringing the external IRQ pin to a logic low. The IRQ has one of the highest priorities of all

interrupts, and this can be used to advantage. The IRQ must be enabled for this function to work

```
clr  INTSCR     clear Interrupt Status and Control Register
```

6.4 Interrupts

Many of the functions of the chemical detector are interrupt-driven. Normal operation continues until a specific event occurs, triggering an interrupt which is then evaluated and acted upon. Certain interrupts are disabled during operation to prevent unintended interrupts from interfering with the operation

6.4.1 Keyboard

Keyboard interrupts are enabled during times when the program requires input from the user (such as gas selection) and during times when input from the user is a possibility (during wait periods, user can initialize data dump sequence).

Upon keyboard interrupt (KBI enabled, and a key is pressed), program jumps to service routine. In this routine, the register "`keybd_flag`" is set according to which button or buttons were pressed. Certain tricks are employed to ignore "jumpy" buttons (i.e. buttons are debounced) to give smooth button operation during selection routines.

Enter Conditions: "keybd_flag" register is unknown (could be clear or have data)
Exit Condition: "keybd_flag" = 00000001 if button A0 is pressed (enter)
00000010 if button A1 is pressed (select)
00000111 if both buttons are pressed

Upon returning from interrupt, the program will act on value of "keybd_flag" as appropriate

6.4.2 Timer1 channel 0

This interrupt is triggered by a rising edge input on the T1CH0 pin, and monitors the output of the photosensor. This interrupt is only enabled during the time when the microcontroller is actually "looking" at the card. This means it is enabled for .0125 seconds, 8 times in sequence, to "count" the number of pulses from the output of the photosensor in one second. The total value of these counts is the value of the photosensor's frequency output.

When the interrupt is enabled, a rising edge on the T1CH0 pin causes an interrupt, and the program jumps to the service routine. This service routine increments the 16-bit H:X register by one, and returns.

One problem here is that interrupt calls automatically store the H:X register to the stack and restore the values upon an "rti" (return from interrupt). This makes it very difficult to use the 16 bit H:X registers to count the values.

Other methods of 16-bit addition consume more cycles (time), and it is imperative that this service routine complete quickly, as higher frequencies can try to trigger another interrupt before the first has been addressed.

This problem is avoided by not using the traditional "rti" instruction to return from the interrupt, which would render the fast H:X counter worthless. Instead, the stack pointer is moved back to the return address, skipping the other pushed values, and an "rts" is called which only modifies the Program Counter. The interrupt enable mask is manually cleared.

Code for Timer Channel 0 Interrupt Service Routine

```
sei                 ; set interrupt mask          2 cycles
aix     #$01        ; increment H:X register      2 cycles
lda     T1SC0       ;                             3 cycles
and     #$7F        ; manually clear the          2 cycles
sta     T1SC0       ;    T1CH0 Interrupt Flag     3 cycles
ais     #$03        ; adjust stack pointer        2 cycles
cli                 ; clear interrupt mask        2 cycles
rts                 ; PC <- return address        4 cycles
```

Overall, 11 cycles of program time are saved by this method, allowing higher frequencies to be measured

6.4.3 Timer1 Overflow

Timer 1 was initialized to overflow the counter every 0.125 seconds. When the Timer1 counter overflows, an interrupt occurs. Because there are different responses to a timer overflow, his interrupt merely sets a bit in the "timer_flag" register.

This overflow is used for two different functions:

During the "look at card" portion of the program, the H:X register to accumulate pulse counts until the timer overflows after exactly 125 ms (1/8 second). Once the overflow occurs, a flag is set, and control is returned to the main program. The main program then accumulate H:X counter into a larger (24-bit) total pulse count register set, and continues to do so over eight separate 1/8 second intervals.

The other function occurs after the sensor frequency output has been measured. During the remainder of the program, the Timer continues to overflow (and cause interrupts) every 1/8 second. These overflows are counted so that total elapsed time can be measured. This allows the "look at card" routine to be repeated at regular intervals. Currently this is set so that the card is looked at once every 5 seconds, although this amount can be changed to any value desired.

```
        brset   7,timer_flag,countdown_10sec
                                    ;which timer function?

        bset    0,timer_flag        ; set flag to indicate
                                    ; overflow
        bset    5,T1SC              ; stop timer
        lda     T1SC
        and     #$27
        sta     T1SC                ; clear Tim1 Overflow
                                    ; Interrupt
        cli
        rti                         ; return

countdown_5sec:         ; if the 5-second counter is needed,
                        ; code executes here to count overflows
```

6.4.4 IRQ

Currently this Interrupt is not being used. Initial plans were to attach a button to the IRQ pin which would cause an IRQ interrupt and initiate some function, such as data dump to serial port. This wound up not being needed as the two Keyboard Interrupt buttons were sufficient for functionality.

This option is always available for future expansion and could incorporate any type of external exception desired to interrupt normal operation of the device, other than reset, which is already provided for.

6.4.5 Dummy Interrupts

The dummy interrupt service routine is provided for all unused interrupts in the event that a spurious and unintentional interrupt is requested. This service routine immediately returns control back to the routine from which it was called.

6.5 Main Routine

Upon a power-on or external reset, the device immediately starts executing at the memory location provided by the reset vector. In this program the starting address (main_init) is $D000.

The first thing that occurs upon starting main_init is to run through the entire initializations. These are the initializations described in the preceding section, and are accessed by several successive "jump subroutines". Once this is finished, the main program begins and each of the following subroutines are entered in turn.

6.5.1 prompt_gas

Enter Conditions: none
Exit Conditions: "gas_number" register contains number (0–17) of selected gas.
"gas_text" contains address of text for the gas selected.

This routine prompts the user to find out which gas is to be detected. Obviously nothing can be started until this is specified, as each gas has different constants associated with its calculations.

LCD prints the name of the first gas, and the user can either press "select" or "enter". If select is pressed, the next gas name is printed. Pressing select will continue to cycle through all 18 gas names until the last name is reached where it rotates back to the first name and repeats. Once enter is pressed the number of the gas is stored in "gas_number" and the subroutine is exited.

6.5.2 Calibrate

Enter conditions: none
Exit conditions: "reference_value" holds a value that is the difference between the measured unexposed color and the base calibration color (white)

Once a gas card is selected, the microprocessor must calibrate according to the color of the unexposed card. The user is prompted to press the "enter" button when ready. The card must be inserted prior to pressing enter. The calibrate routine looks at the card 4 separate times to establish the base unexposed color level. If this value varies at all, calibration fails. If the color measured is too dark, it is assumed the card is bad, and calibration fails.

If the user wishes to skip the failure checks, they can press both buttons (select and enter) simultaneously, and the calibrate reference will be forced as the first value read with no error checking.

6.5.3 look_at_card

Enter conditions: none
Exit conditions: 24-bit value of frequency stored in the following registers:

```
pulse_xtra : pulse_count : pulse_count+1
```

This function enables the timer1 overflow interrupt in conjunction with the Timer1 Channel0 interrupt to count total number of pulses output by the photosensor.

The LED is first turned on and allowed to "warm up" for 250ms prior to measuring. The output is then measured eight times each at 125 ms (1/8) second, and each count is added to the 24-bit register sequence noted above.

Once the sequence is complete, the LED is turned off, and the Channel 0 interrupt is disabled and control is returned to the main loop.

6.5.4 start_pulse_timer

Enter conditions: none
Exit conditions: timer_flag, pulse_timer, total_time are all modified.

This routine sets the "timer_flag" so that subsequent Timer1 overflows will count the number of overflows that occur. "pulse_timer" is set to the value of 30, and this value will be decremented each time the Timer1 Overflow interrupt occurs.

Since 1.25 seconds were used in the "look at card" routine, 30 occurrences of timer overflows give a total time of 5 seconds. (1.25 + 30 * 0.125 = 5.00).

"total_time" increments by one, counting the total number of 5 second intervals.

6.5.5 avg_pulses

Enter conditions: pulse_xtra, pulse_count, pulse_count+1
Exit conditions: pulse_count, pulse_count+1

This routine divides the total number of pulses counted in one second by four and rounds to the nearest whole number. This new 16-bit value is stored in `pulse_count:pulse_count+1` and is effectively the average number of pulses counted in a 250 ms. period. The resulting 16-bit value is easier to work with when converting to frequency because the DIVide function can only divide a 16 bit value by an 8 bit value.

6.5.6 pulse_to_freq

Enter conditions: pulse_count, pulse_count+1
Exit conditions: frequency

This routine converts the number of pulses found in a 250 ms period to a value equal to the frequency of the output in kilohertz. By dividing the 16-bit value in pulse_count by 250, we have effectively divided the total pulses in 1second by 1000. Since DIVide can only accommodate an 8-bit divisor, this works nicely because 250 just barely makes the 8-bit divisor requirement.

6.5.7 store_history

Enter conditions: frequency
Exit conditions: history stack is modified.

This function stores the most recent frequency value measured on the photosensor (found in frequency) onto the history stack. The history stack is a 9-location FIFO stack that rotates "downwards".

Location $0230 always contains the current frequency value.

Each time a new value is pushed onto location $0230, every other value is shifted down one place.

The oldest value, at location $0238, is then dropped and lost.

$0230	0th (current)
$0231	1st (recent)
$0238	8th (oldest)

6.5.8 smooth_points

Enter conditions: samp0-samp7
Exit conditions: approx0-approx8 are updated

This routine fits a quadratic curve to a series of 8 data points stored in samp0-samp7. This is done by accessing a table stored in smooth_coefficients (Appendix C). Each entry in the table is 32 bits and the MSB is a sign bit. If the MSB = 1, the coefficient value is supposed to be negative. Nine approximated values are stored in approx0-approx8. The ninth (extrapolated) value should be compared to samp8 and the difference compared to the standard deviation.

6.5.9 compute_std

Enter conditions: samp8, approx0-approx8
Exit conditions: detect is updated

This function computes the standard deviation for samp0-samp7 and determins if samp8 exceeds the twice the standard deviation. If so, it flags the detect variable with the appropriate value.

6.5.10 look_detect

Enter conditions: "detect"
Exit conditions: none

This routine looks at the "detect" register that was modified by the compute_std routine prior. If there was no change in frequency measured, detect will hold the value 0h00. The detect register will have the value 0hFF if there was a notable

decrease in the frequency of the photosensor, which corresponds to an increase in color intensity of the card and corresponding change in exposure.

If such a change is detected, the alarm is enabled by enabling timer2 which has previously been configured to deliver a PWM signal of 4 kHz. Furthermore, the associated "alarm light" (LED) will be turned on.

These alarms will automatically be turned off prior to re-entering the "look at card" routine. Therefore, if there is continual concentration of gas present, the alarm will sound for 3.75 seconds, turn off for 1.25 second, then repeat until there is no more gas concentration present.

Additionally, "detect" can have a positive value, 0h01, if there is a decrease in the accumulated exposure on the card. This would only happen legitimately if the card being used was reversible, that is to say it measures concentration and not exposure. Otherwise, if the measured frequency has increased, this would indicate the photosensors determined that color of the card has lightened since the last 'look' which would indicate some sort of measuring error either in the photosensors, or more likely caused by a change in LED intensity, alteration of the card, or change in positioning of the sensor.

The alarm is not programmed to sound on a positive value of "detect" as this is either a function of a reversible dosimeter card being removed from presence of gas, or an anomaly of the detection process.

6.5.11 calculate_dose

Enter conditions: gas_number, frequency
Exit conditions: dose_value(16-bit), old_dose_value(16-bit)

This is the heart of the program. This routine takes the measured frequency of the card, which is linearly related to color intensity, and corresponds it to the physical value of dose (actually exposure) as measured in PPM-Hours.

First, the gas_number is used to index ROM memory where the gas constants are stored. There are three 16-byte values stored in memory for each gas: A, B, C. The governing equation for calculating dose according to these constants is:

$$\log_2 (\text{Dose}) = A + B*(\text{frequency}) + C*(\text{frequency})^2$$

• Equation to Calculate Dose

For some gases, there is no 2^{nd} order term, so C is stored as zero and calculated to simplify the overall algorithm. Additional complication is that some of the gas constants are positive and some are negative. In general, the integer math routines

used to multiply or divide large numbers are for unsigned values. Therefore, to avoid unnecessary difficulty, we specify whether A, B, and C are positive or negative by bit flags in a control byte associated with each set of gas constants. These signs are then applied after multiplication to each term as appropriate.

The right hand side of the above equation is evaluated completely as shown. This value becomes the power to which 2 is raised to calculate the total dose. Since an 8-bit microcontroller complains greatly when asked to evaluate exponentials, this is accomplished by use of a lookup table (Appendix A).

The gas constants were actually calculated using powers of 2, so we apply this algorithm in reverse to the final result of the RHS of the equation. Now we can easily extract out the meaningful bits of the power value, and the associated lookup table is constrained to a manageable 256 locations. These values are meaningful within the range of possible card readings, and have sufficient resolution for purposes of detecting measurable changes without having to store ridiculous amounts of information in ROM.

The Dose value is a sixteen bit value that is actually the Dose amount expressed in thousandths ppm-hours (i.e., ppm-hours / 10^3). Once this binary value is converted into decimal digits, it is a simple matter of inserting a decimal point so the display reads correctly.

For more detail on how these algorithms are used, see the Section 6.5.12, below.

6.5.12 calculate_concentration

Enter conditions: dose_value, old_dose_value
Exit conditions: conc_value

The relationship between Dose and Concentration is as follows:

$$Dose = \int Concentration\, dt \qquad\qquad Concentration = \frac{d}{dt} Dose$$

To find the current concentration, then, is a simple matter of finding the difference between the current dose and the old dose values, and dividing it by the time. since the time difference, 5 seconds, is actually 1/720 hour, we multiply the difference by 720 to get the concentration.

Since the Dose values are actually in thousandths of ppm-hours, this concentration is in thousandths pmm's. Such precision is unnecessary and larger concentrations will not fit into a 16-bit location. Therefore this amount is divided by 100 before being stored into its final location, "conc_value".

Since conc_value is the concentration in tenths of ppm's, we can apply the same principle as dose_value, where the binary value is converted to decimal digits for display, and a decimal point is inserted in the proper position on the display.

6.5.13 output_results

This subroutine prints the results of the previous calculations. Dose value is displayed on the top line and is expressed in ppm-hours with three decimal places of precision. Concentration is displayed on the bottom line and is expressed in ppm, with one decimal place precision.

Function calls are used to convert the binary/hex values to representative decimal digits. Function calls are also used to write all data to the LCD, so it is only a matter of passing the character to be printed to another subroutine.

Once data is written to the LCD, it retains it as long as power is on, or until new data is written, or a "clear LCD command" is written to the LCD.

6.5.14 write_memory

The external memory is a 512 K byte-addressable memory with 19 address lines. Currently we are only using 15 address lines so only 32K bytes are available, but this is sufficient for our purposes.

Once the dose and concentration values have been determined, the program write them to the external memory if there is has been a change. The exception to this rule is that the first 16 measurements will be recorded regardless of change.

Whenever memory is written, four pieces of information are stored

1) The 16-bit "total_time" value. elapsed time in increments of 5 seconds.

2) The 16-bit "dose_value" information, in thousandths of ppm-hours

3) The 16-bit "conc_value" information, in tenths of ppms

4) The 16-bit zero-extended "detect" byte.

Each time a change occurs, 8 bytes are written. This will allow for over 4000 changes to be written at current addressing capabilities. Adding address lines will extend this amount. When the data is recovered, it will be easy to reconstruct the dose and exposure history by referencing each point to its relative event time.

8 bytes of data are written one byte at a time in the following manner

1) "memory_add" holds 16-bit address, "byte_write" holds data to be written
2) The byte to be written is sent out the Port B
3) Latch In is enabled so that this data is written into the Latch

4) Latch In is disabled so further changes on Port B affect the latch
5) 15-bit address is written to Port C[6:0] : Port B[7:0]
6) Memory chip is enabled for this address
7) Latch In, (holding the data to be written) is Output Enabled
8) Data in Latch In is seen at the memory's 8-bit Data bus
9) Memory is Write Enabled, writing data to memory
10) Latch In and Memory are disabled.
11) 16-bit "memory_add" register is incremented.
12) Repeat for each of the remaining bytes.

6.5.15 download_history

Once all of the above routines have completed there is a waiting period to determine what happens next. If the user wishes to download the accumulated data to a PC, they will press both "select" and "enter" buttons simultaneously to enter the "download_history" routine.

If the user does not request download history, the pulse timer will continue to decrement until it is time to get another "look" at the card. As mentioned earlier, this is currently set at 5 seconds between pulses, but this can be easily adjusted to any amount desired.

When download history is requested, a "stop byte" is written to the current memory address to indicate the end of the current memory. Then the LCD will display a prompt for the user to "press enter to begin". This allows the user time to check the connection of the DB-9 cable from the detector unit to the serial port on a PC or laptop. When this is confirmed, enter is pressed and the download begins.

Data is read from the memory beginning at address 0h0000 and continues until the "stop byte" is reached. Once one byte is read from memory it is transmitted by the Serial Communication Interface to the Maxim 232 and then along the DB-9 to the connected serial device.

With memory_add = 0000, begin reading memory:

1) Disable LatchOut, keeping output at High-Z
2) Output 15-bit memory_add to PortC[6:0] : PortB[7:0]
3) Enable Memory read this address and Output its data.
4) Enable LatchOut to 'write' data found on memory data bus.
5) Disable Memory, keep LatchOut's output disabled
6) Switch PortB to become an input
7) Enable LatchOut's output so that it will be sent to PortB
8) Collect data at PortB, save to "byte_read"
9) Check SCI Status Register until it is no longer "busy"
10) Write data in "byte_read" to SCI Data register.
11) Check if "byte_read" is the "stop_byte", if so, routine ends
12) Increment "memory_add", repeat cycle at step (1)

Data is sent to the PC in the following format:

"Gas Name = X", "total time = N"
time1, dose1, conc1, detect_byte1
time2, dose2, conc2, detect_byte2
time3, dose3, conc3, detect_byte3
…
…
timeN-1, doseN-1, concN-1, detect_byteN-1
timeN, doseN, concN, detect_byteN
"stop"

6.6 Algorithms

6.6.1 Exponential Extraction and Lookup

Each dosimeter card has an associated formula by which its color is related to the accumulated exposure that has developed. These formulas have been previously described by the card's developer, Dr. A. J. Attar of Raleigh, NC, who developed the "ChemSense" cards.

Generally speaking, the intensity of the color is logarithmically proportional to the exposure, as measured in ppm-hours. A typical formula for a gas card is

$$\log(\text{Dose}) = A + B * R1 + C * R1^2$$

Two calibration cards (white and grey) are associated with all of the gas cards, and corresponds the relationship between different color levels and dosage gains. The calibration cards are designed to give a linear gain from 0 (white) and 221 (grey).

When these calibration cards are measured by the light-to-frequency converter, they are found to have a linear gain from 240 kHz (white) to 125 kHz (grey). Therefore the R1 value in the gas card datasheets supplied by ChemSense must be translated to Frequency, by the ratio such that

$$R1 = 240 - 115 * \text{Frequency} / 221$$

Replacing all instances of R1 in the ChemSense formulas gives the Log(Dose) as a function of frequency. To simplify matters, the formulas are changed to base2 so that the log is base 2 rather than base 10. This is accomplished by dividing the right hand side by the by the base10 log of (2). Now the general formula is

$$\log_2(\text{Dose}) = (1/\log(2))*(A+B*(240-\text{Freq}*115/221) + C*(240-\text{Freq}*115/221)^2)$$

Reevaluating this formula for the specific gas card gives the following:

$$\log_2(\text{Dose}) = A' + B' * \text{Freq} + C' * \text{Freq}^2$$

Where $A' = [A + (B * 240) + (C * 57600)] / \log(2)$
$B' = [(B * 115 / 221) + (C * 55200 / 221)] / \log(2)$
$C' = (C * 13225 / 48841) / \log(2)$

These constants (A', B', and C') are still of very small magnitude, often to the order of 10^{-2} to 10^{-6}, therefore we must make them sufficiently large so that they can be calculated in integer math routines without losing accuracy. To solve this issue we multiply the entire Right Hand Side of the Equation by 2^{16}, 2^{18}, or 2^{20}, depending upon the magnitude order of the smallest values.

If, for instance, we multiply by 2^{18}, the new formula becomes:

$$\log_2(\text{Dose}) / 2^{18} = (A' + B' * \text{Freq} + C' * \text{Freq}^2) * 2^{18}$$

and the new gas constants, A'', B'' and C'' are simply equal to the previous single-prime versions multiplied by 2^{18}. These new gas constants are calculated for each gas card, and are placed in the gas constants lookup table (Appendix B). The multiplier (2^{16}, 2^{18}, or 2^{20}) is chosen to give the largest possible numbers while keeping each of them within a 32-bit range.

Once this table has been set up correctly, all that is needed to accurately determine dose is the base (unexposed) frequency reading and the current (exposed) frequency reading from the dosimeter card. This of course assumes that the LED/photosensor combination remains electrically (led luminosity) and mechanically (sensor placement) identical to the setup that determined the calibration card readings of 240kHz (white) and 125 kHz (grey).

Determining Dose is done as follows:

Given a specific gas card, the appropriate constants A, B, and C, are read from the lookup table. Also in the lookup table is information indicating the sign of the constants (positive or negative) and what multiplier was used (2^{16}, 2^{18}, or 2^{20}) when determining the constants.

The frequency of the exposed card is increased by a "reference_value" which is the difference between the gas card's unexposed reading and the calibration base of 240. This adjusted frequency reading effectively puts the Dose v. Time curve at the origin for time=0.

The adjusted frequency and the gas constants are used to evaluate the formula,

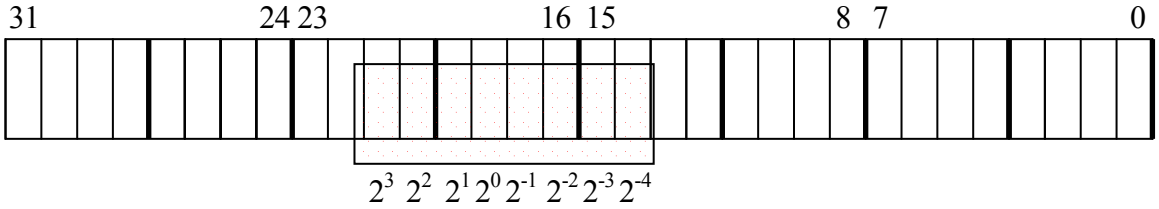
$$A + B * \text{Freq} + C * \text{Freq}^2$$

If for instance the original multiplier was 2^{18}, this value is then equal to the log base 2 of the dose, divided by 2^{18}. Alternately, we can say this value, divided by 2^{18}, is the power which 2 must be raised in order to find the Dose.

In other words, if the original multiplier was 2^{18}, then

$$Dose = 2^{\frac{A + B * Freq + C * Freq2}{2^{18}}}$$

By extracting the meaningful bits, we can use a lookup table to find the value of the dose (Appendix A). The equation, $A + B * Freq + C * Freq^2$, has been found and the result is a 32-bit value. Rather than divide by 2^{18}, it is simpler to extract bits 14 to 21 where bits 18-21 are the whole number of the exponent, and bits 14-17 are the fractional part of the exponent.



This 8-bit extraction gives a single byte that represents an exponent value from

+ 7.9375 to - 8.000

and all points in between with a resolution of 0.0625

This byte is now the actual exponent, within the range just described, and is used as an index to access a table of values which contain the values of 2 raised to that exponent. The values that are stored in the exponent lookup table are technically binary values, but they are equal to 2 raised to that power, then multiplied by 1000. This answer is the thousandths of ppm-hours described earlier, and so are easily converted to decimal characters and overlaid onto a preset decimal point.

Example:

say the bits extracted from the above result were 00101100. This represents the value 2.75, since the binary point is located between the two nibbles. The hex representation of these bits is 0h2C. Since the lookup table contains 16-bit values, the exponential table is indexed with an offset of 0h58.

We know that 2 raised to the 2.75 is (rounded to three decimal places) 6.727. The 16-bit value found on the lookup table is 0h1A47. Hex 1A47 is 6727, which is the answer we desire, multiplied by 1000.

6.6.2 Polynomial Regression/Standard Deviation

The algorithm to compute the standard deviation is straightforward. Eight previous sensor measurements are stored in samp0-samp7. The current measurement is stored in samp8.

Polynomial regression is used to fit a curve to the previous eight samples and to extrapolate a predicted approximation for the current sample. The fitted or approximated points are stored in approx0-approx8. The extrapolated prediction (approx8) is then compared to the current measurement (samp8) and if the difference is greater than two times the standard deviation computed for the previous eight values, the alarm is set.

If we use y to represent samp, and Y to represent approx, the curve can be fitted using a standard four term polynomial regression algorithm.

$$Y := x \rightarrow b0 + b1\,x + b2\,x^2 + b3\,x^3$$

where x = time and

$$b0 := \frac{59}{66}y_0 + \frac{8}{33}y_1 - \frac{2}{33}y_2 - \frac{4}{33}y_3 - \frac{1}{22}y_4 + \frac{2}{33}y_5 + \frac{1}{11}y_6 - \frac{2}{33}y_7$$

$$b1 := \frac{31}{126}y_1 + \frac{145}{252}y_2 + \frac{17}{42}y_3 + \frac{1}{84}y_4 - \frac{41}{126}y_5 - \frac{83}{252}y_6 + \frac{5}{18}y_7 - \frac{31}{36}y_0$$

$$b2 := -\frac{39}{308}y_1 - \frac{47}{231}y_2 - \frac{101}{924}y_3 + \frac{23}{462}y_4 + \frac{155}{924}y_5 + \frac{32}{231}y_6 - \frac{19}{132}y_7 + \frac{5}{22}y_0$$

$$b3 := \frac{5}{396}y_1 + \frac{7}{396}y_2 + \frac{1}{132}y_3 - \frac{1}{132}y_4 - \frac{7}{396}y_5 - \frac{5}{396}y_6 + \frac{7}{396}y_7 - \frac{7}{396}y_0$$

Substituting in the above equations for b0 - b3, we can solve for Y(0)-Y(8) in terms of y0 – y8.

$$Y0 := 0.8939393939\,y_0 + 0.2424242424\,y_1 - 0.06060606061\,y_2 - 0.1212121212\,y_3 - 0.04545454545\,y_4 + 0.06060606061\,y_5 + 0.09090909091\,y_6 - 0.06060606061\,y_7$$

The rest of the equations are located in the Appendix D.

The constants in the equation for Y were stored in the standard deviation table of values as 32 bit values. They were all scaled by a factor of 2^17, in order to get six decimal places of accuracy. If the constant was negative in the equation, there was a leading sign bit to alert the calculating function to this fact. That sign bit is intended to be masked before the entry is used in computation.

Because it is very difficult to compute to take the square root of a number in the GP32, a slightly different standard deviation algorithm was used than is normally. The end result is the same, however.

The formula for the standard of deviation is

$S = \sqrt{S^2}$

where

$S^2 = \sum(|samp - approx|)^2 / N$

and N = # of samples used to compute the standard deviation (in this case N = 8).

S is then used with the current value and current approximation as follows:

```
If( |approx8 - samp8| > 2*S) then alarm
Else no_alarm
```

Because it is prohibitively difficult to take a square root, the evaluation in the if statement is squared and expanded on both sides. This is complicated, however, by the absolute value term. This can lead to a loss of information in cases when samp8 > approx8. Because of this, a condition flag was set whenever samp8 > approx8.

The resulting pseudo-code is

```
If( samp8 > approx8 ) then sign = -1
Else sign = +1
Value = samp8^2 + approx8^2 + sign*2*samp8*approx8
If( value > 4*S^2) then alarm
Else no_alarm
```

The S^2 term is multiplied by four because in the original test, 2*S was being evaluated. To square the term, then, results in 4*S^2.

6.6.3 Housing / Mechanical

The housing consists of two components. A shell, bought from Radio Shack (RS# 270-1801) and a custom made piece which provides a slot for the card to fit into and viewing apertures with holes for an illuminating LED and a viewing sensor. The Radio Shack enclosure must also be modified somewhat according to the plans included in the Appendix I.

7 Construction Details

This device should only be constructed by professionals in a properly equipped lab, due to the fine pitch leads on the majority of the components.

The first thing that should be done is an assembly of the code using the WinIDE software, included on the CDROM. Program the GP32 using a programming POD.

After gathering all the components, solder them onto the PCB. It is very important that the LED for the sensor is not soldered prior to fitting it. This can be accomplished by sliding it into the vias and placing the PCB onto the card viewing apertures. Fit the LED into the LED hole cut into the aperture and then fit the board down onto the aperture in such a way that the sensor is placed in the center of the opening. Then solder the leads of the LED. Connect a battery to the device. Adjust the LCD contrast potentiometer. Solder jumper wires between the appropriate vias and their corresponding buttons and the speaker/alarm LED.

After all the components have been soldered onto the PCB, the device must be calibrated. Place the white-white calibration card into the device, close it, and run the over-ride calibration feature by pressing button1-button2 when it says it is ready to calibrate.

The reading should be F0. If it is not, open the device and adjust the LED potentiometer until the over-ride calibration displays F0. After the device is calibrated, it is ready for use.

8 Construction Costs

Microcontroller:	$5.50
Photo-Sensor:	$2.67 x 2
White LEDs:	$3.00 x 2
SRAM:	$4.75
Discrete Comp.:	$5.00
Housing:	$7.99
Housing Mods:	custom cost
PCB:	custom cost
Total:	**$34.58 + custom costs**

9 Conclusions

The following improvements should be made to the device

- Track down and eliminate the bug in the memory read/write code
- Re-route and cut a hole in the PCB to allow the LED to jut out of the card viewing aperture through the PCB. This will allow the sensor to be mounted directly on the PCB and also sit flush in the aperture.
- Add mounting holes to the PCB layout
- Re-assemble the code with the alarm initialization routine enabled
- Conduct extensive testing with various gasses to fine-tune the coefficients in the dose calculation table
- Develop and integrate the circuitry which will allow the device to be re-programmed in the field.

User Instructions

1. Connect a nine-volt battery to the device.

2. The LCD prints the name of the first gas, and the user can either press “select” or “enter”. If select is pressed, the next gas name is printed. Pressing select will continue to cycle through all 18 gas names until the last name is reached where it rotates back to the first name and repeats.

3. Once a gas card is selected, the microprocessor must calibrate according to the color of the unexposed card. The user is prompted to press the “enter” button when ready. The card must be inserted prior to pressing enter. The calibrate routine looks at the card 4 separate times to establish the base unexposed color level. If this value varies at all, calibration fails. If the color measured is too dark, it is assumed the card is bad, and calibration fails.

4. If the user wishes to skip the failure checks, they can press both buttons (select and enter) simultaneously, and the calibrate reference will be forced as the first value read with no error checking.

5. At this point the device is operating in normal mode and will continue to do so until the user wishes to transfer the exposure history to a computer

6. If the user wishes to download the accumulated data to a PC, they can press both buttons simultaneously to enter the "download history" routine.

7. The LCD will then display a prompt for the user to "press enter to begin". This allows the user time to check the connection of the DB-9 cable from the detector unit to the serial port on a PC or laptop. The user should start the data visualization program or HyperTerminal. When this is confirmed, enter is pressed and the download begins.

8. Data is sent to the PC in the following format:

"Gas Name = X", "total time = N"
time1, dose1, conc1, detect_byte1
time2, dose2, conc2, detect_byte2
time3, dose3, conc3, detect_byte3
…
…
timeN-1, doseN-1, concN-1,
timeN, doseN, concN, detect_byteN
"stop"

Appendix A

Exponent Table

```
    org exp_table_p

index_pos:
    fdb
1000T,1044T,1091T,1139T,1189T,1242T,1297T,1354T,1414T,1477T,1542T,1610T,
1682T,1756T,1834T,1915T,2000T,2089T
    fdb
2181T,2278T,2378T,2484T,2594T,2709T,2828T,2954T,3084T,3221T,3364T,3513T,
3668T,3830T,4000T,4177T
    fdb
4362T,4555T,4757T,4967T,5187T,5417T,5657T,5907T,6169T,6442T,6727T,7025T,
7336T,7661T,8000T,8354T
    fdb
8724T,9110T,9514T,9935T,10375T,10834T,11314T,11815T,12338T,12884T,13454T,
14050T,14672T,15322T,16000T
    fdb
16708T,17448T,18221T,19027T,19870T,20749T,21668T,22627T,23629T,24675T,
25768T,26909T,28100T,29344T
    fdb
30643T,32000T,33417T,34896T,36441T,38055T,39739T,41499T,43336T,45255T,
47258T,49351T,51536T,53817T
    fdb 56200T,58688T,61287T,64000T


    org exp_table_n

index_negs:
    fdb
4T,4T,4T,4T,5T,5T,5T,5T,6T,6T,6T,6T,7T,7T,7T,7T,8T,8T,9T,9T,9T,10T,10T,
11T,11T,12T,12T
    fdb
13T,13T,14T,14T,15T,16T,16T,17T,18T,19T,19T,20T,21T,22T,23T,24T,25T,26T,
27T,29T,30T,31T,33T
    fdb
34T,36T,37T,39T,41T,42T,44T,46T,48T,50T,53T,55T,57T,60T,63T,65T,68T,71T,
74T,78T,81T,85T,88T
    fdb
92T,96T,101T,105T,110T,115T,120T,125T,131T,136T,142T,149T,155T,162T,169T,
177T,185T,193T,201T
    fdb
210T,220T,229T,239T,250T,261T,273T,285T,297T,310T,324T,339T,354T,369T,
386T,403T,420T,439T,459T
    fdb
479T,500T,522T,545T,569T,595T,621T,648T,677T,707T,738T,771T,805T,841T,
878T,917T,958T
```

Appendix B

Gas Constants Table

```
    org GasValues

gas_constants:

       fdb $0200,$0FF0,$000E,$123F,$0000,$0C72,$0000,$0000   ; (0)  Ammonia (L)

       fdb $0100,$3FC0,$0016,$641C,$0000,$5807,$0000,$007B   ; (1)  Anilene (L)   2^18

       fdb $0300,$0FF0,$000C,$6885,$0000,$096E,$0000,$0001   ; (2)  Carbon Monoxide (H)

       fdb $0500,$0FF0,$000B,$E7CE,$0000,$38B3,$0000,$0030   ; (3)  Chlorine

       fdb $0200,$0FF0,$0035,$E6E6,$0000,$3D58,$0000,$0000   ; (4)  Formeldahyde (C)

       fdb $0200,$0FF0,$001E,$EF74,$0000,$2428,$0000,$0000   ; (5)  Glutereldahyde

       fdb $0200,$FF00,$004F,$980E,$0000,$883F,$0000,$000B   ; (6)  Hydrogen Sulfide (H) 2^20

       fdb $0200,$0FF0,$0005,$5DFA,$0000,$0A79,$0000,$0000   ; (7)  Hydrogen Sulfide (L)

       fdb $0500,$0FF0,$0003,$671D,$0000,$2B88,$0000,$0027   ; (8)  Isocyanides

       fdb $0200,$0FF0,$0037,$EA13,$0000,$503F,$0000,$0019   ; (9)  Methanol

       fdb $0500,$0FF0,$0078,$975F,$0001,$58BD,$0000,$00E4   ; (10) MMH

       fdb $0500,$0FF0,$0065,$A9B2,$0001,$1A7C,$0000,$00BD   ; (11) Mustard

       fdb $0200,$0FF0,$000C,$C3BE,$0000,$11F5,$0000,$0000   ; (12) Nitrogen Dioxide (H)

       fdb $0300,$0FF0,$0004,$D506,$0000,$04C2,$0000,$0005   ; (13) Nitrogen Dioxide (L)

       fdb $0500,$0FF0,$0041,$C85B,$0000,$CC3D,$0000,$0089   ; (14) Ozone

       fdb $0300,$0FF0,$000A,$14A5,$0000,$090D,$0000,$0001   ; (15) Propyline Oxide

       fdb $0500,$0FF0,$0010,$3896,$0000,$3E5C,$0000,$0032   ; (16) Sulfur Dioxide (H)

       fdb $0500,$0FF0,$0007,$87E4,$0000,$22C0,$0000,$0021   ; (17) Sulfur Dioxide (L)
```

Appendix C

Smoothing Constants Table

```
    org SmoothTable

smooth_coefficients:

       fdb

$0001,$C9B2,$0000,$7C1F,$8000,$1F07,$8000,$3E0F,$8000,$1745,$0000,$1F07,$0000,$2E8B,$8000,

$1F07

       fdb
$0000,$7C1F,$0000,$BFB9,$0000,$A873,$0000,$5D17,$0000,$046E,$8000,$3ABC,$8000,$39A0,$0000,

$2E8B

       fdb
$8000,$1F07,$0000,$A873,$0000,$D5E3,$0000,$9F95,$0000,$3BD8,$8000,$1F07,$8000,$3ABC,$0000,

$1F07

       fdb
$8000,$3E0F,$0000,$5D17,$0000,$9F95,$0000,$A0B1,$0000,$77B0,$0000,$3BD8,$0000,$046E,$8000,

$1745

       fdb
$8000,$1745,$0000,$046E,$0000,$3BD8,$0000,$77B0,$0000,$A0B1,$0000,$9F95,$0000,$5D17,$8000,

$3E0F

       fdb
$0000,$1F07,$8000,$3ABC,$8000,$1F07,$0000,$3BD8,$0000,$9F95,$0000,$D5E3,$0000,$A873,$8000,

$1F07

       fdb
$0000,$2E8B,$8000,$39A0,$8000,$3ABC,$0000,$046E,$0000,$5D17,$0000,$A873,$0000,$BFB9,$0000,

$7C1F

       fdb
$8000,$1F07,$0000,$2E8B,$0000,$1F07,$8000,$1745,$8000,$3E0F,$8000,$1F07,$0000,$7C1F,$0001,

$C9B2

       fdb
$8001,$0000,$0001,$2492,$0001,$2492,$0000,$0000,$8001,$4924,$8001,$B6DB,$8000,$4924,$0004,

$0000
```

Appendix D

Polynomial Regression Functions Y(y)

$$Y0 := 0.8939393939\ y_0 + 0.2424242424\ y_1 - 0.06060606061\ y_2 - 0.1212121212\ y_3 - 0.04545454545\ y_4 + 0.06060606061\ y_5 + 0.09090909091\ y_6 - 0.06060606061\ y_7$$

$$Y1 := 0.2424242424\ y_0 - 0.1147186147\ y_5 - 0.1125541126\ y_6 + 0.09090909091\ y_7 + 0.3744588745\ y_1 + 0.3290043290\ y_2 + 0.1818181818\ y_3 + 0.008658008658\ y_4$$

$$Y2 := -0.06060606061\ y_0 - 0.06060606061\ y_5 - 0.1147186147\ y_6 + 0.06060606061\ y_7 + 0.3290043290\ y_1 + 0.4177489177\ y_2 + 0.3116883117\ y_3 + 0.1168831169\ y_4$$

$$Y3 := -0.1212121212\ y_0 + 0.1168831169\ y_5 + 0.008658008658\ y_6 - 0.04545454545\ y_7 + 0.1818181818\ y_1 + 0.3116883117\ y_2 + 0.3138528139\ y_3 + 0.2337662338\ y_4$$

$$Y4 := -0.04545454545\ y_0 + 0.3116883117\ y_5 + 0.1818181818\ y_6 - 0.1212121212\ y_7 + 0.008658008658\ y_1 + 0.1168831169\ y_2 + 0.2337662338\ y_3 + 0.3138528139\ y_4$$

$$Y5 := 0.06060606061\ y_0 + 0.4177489177\ y_5 + 0.3290043290\ y_6 - 0.06060606061\ y_7 - 0.1147186147\ y_1 - 0.06060606061\ y_2 + 0.1168831169\ y_3 + 0.3116883117\ y_4$$

$$Y6 := 0.09090909091\ y_0 + 0.3290043290\ y_5 + 0.3744588745\ y_6 + 0.2424242424\ y_7 - 0.1125541126\ y_1 - 0.1147186147\ y_2 + 0.008658008658\ y_3 + 0.1818181818\ y_4$$

$$Y7 := -0.06060606061\ y_0 - 0.06060606061\ y_5 + 0.2424242424\ y_6 + 0.8939393939\ y_7 + 0.09090909091\ y_1 + 0.06060606061\ y_2 - 0.04545454545\ y_3 - 0.1212121212\ y_4$$

$$Y8 := -0.5000000000\ y_0 - 0.8571428571\ y_5 - 0.1428571429\ y_6 + 2.\ y_7 + 0.5714285714\ y_1 + 0.5714285714\ y_2 - 0.6428571429\ y_4$$

Appendix E

Memory Map

0000	System Registers
0040	User RAM
0100	Runtime Stack
0230	History Stack
0240	Reserved
8000	Exponent Lookups
8200	Text
8800	Integer Math Routines
9000	Gas Constants
9200	Data Smoothing
D000	Program ROM
FFDC	Interrupt Vectors

Appendix F

Bill of Materials

Item	Quantity	Reference	Part
1	11	C1,C8,C9,C10,C11,C12,C16,C17,C18,C19,C20	0.1uF
2	1	C2	0.01uF
3	1	C3	0.033uF
4	2	C4,C5	20pF
5	1	C13	10pF
6	1	C14	10uF
7	1	C15	.01uF
8	3	D1,D2,D3	LED
9	1	J1	8 HEADER
10	1	LS1	SPEAKER
11	2	Q2,Q1	2N2481
12	1	R1	10M
13	4	R2,R4,R5,R13	10k
14	1	R3	330k
15	5	R6,R8,R10,R11,R12	1k
16	1	R7	330
17	1	R9	POT
18	4	SW1,SW2,SW3,SW4	SW PUSHBUTTON
19	1	TP1	PWR
20	2	TP2,TP4	GND
21	1	TP3	DB9_IN
22	1	TP5	DB9_OUT
23	1	U1	MC68HC908GP32
24	2	U3,U2	TCS230
25	1	U4	HM628512C
26	2	U5,U6	DM74LS374WM
27	1	U7	MAX232A
28	1	U8	VREG
29	1	Y1	CRYSTAL

Appendix G

Schematics

Appendix H

Printed Circuit Board Layout

Appendix I

Sensor Cell Mechanical Diagram

Appendix J

Sensor Data

RED CARD

USING MAX CALIBRATION SETTING		
CLEAR SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	ED	237
5.00	DE	222
8.00	C8	200
15.00	AA	170
30.00	7C	124
RED SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	3F	63
5.00	3F	63
8.00	3D	61
15.00	3B	59
30.00	37	55
GREEN SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	3F	63
5.00	3A	58
8.00	33	51
15.00	28	40
30.00	1A	26
BLUE SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	52	82
5.00	4B	75
8.00	41	65
15.00	33	51
30.00	1F	31

GREEN CARD

USING MAX CALIBRATION SETTING		
CLEAR SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	EE	238
5.00	E7	231
8.00	D6	214
15.00	9E	158
30.00	6C	108
RED SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	3D	61
5.00	3B	59
8.00	36	54
15.00	26	38
30.00	18	24
GREEN SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	41	65
5.00	40	64
8.00	3C	60
15.00	31	49
30.00	25	37
BLUE SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	53	83
5.00	50	80
8.00	4A	74
15.00	34	52
30.00	23	35

BLUE CARD

USING MAX CALIBRATION SETTING		
CLEAR SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	E2	226
5.00	D6	214
8.00	C5	197
15.00	9E	158
30.00	69	105
RED SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	3A	58
5.00	36	54
8.00	31	49
15.00	26	38
30.00	17	23
GREEN SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	3C	60
5.00	38	56
8.00	32	50
15.00	26	38
30.00	17	23
BLUE SENSOR		
COLOR (%)	HEX (kHz)	DEC (kHz)
2.00	51	81
5.00	4E	78
8.00	4A	74
15.00	40	64
30.00	31	49

Appendix K

Pin Connection Spreadsheet

GP32 Pin	First Device Pin	Shared Pin	Shared Pin
A0	KBD0		
A1	KBD1		
A2	LATCH_IN_E		
A3	LATCH_IN_OE		
A4	LATCH_OUT_E		
A5	LATCH_OUT_OE		
A6	MEM_WE		
A7	LCD_RW		
B0	MEM_AD0	LATCH_IN_0	LATCH_OUT_0
B1	MEM_AD1	LATCH_IN_1	LATCH_OUT_1
B2	MEM_AD2	LATCH_IN_2	LATCH_OUT_2
B3	MEM_AD3	LATCH_IN_3	LATCH_OUT_3
B4	MEM_AD4	LATCH_IN_4	LATCH_OUT_4
B5	MEM_AD5	LATCH_IN_5	LATCH_OUT_5
B6	MEM_AD6	LATCH_IN_6	LATCH_OUT_6
B7	MEM_AD7	LATCH_IN_7	LATCH_OUT_7
C0	MEM_AD8	LCD_D4	
C1	MEM_AD9	LCD_D5	
C2	MEM_AD10	LCD_D6	
C3	MEM_AD11	LCD_D7	
C4	MEM_AD12	LCD_D8	
C5	MEM_AD13		
C6	MEM_AD14		
D0	SENSOR/LED_CTRL		
D1	SENSOR_SW2	MEM_CS	
D2	SENSOR_SW3	MEM_OE	
D3	LCD_E		
D4	SENSOR1_INPUT		
D5	ALARM_1		
D6	SENSOR2_INPUT		
D7	ALARM_2		
E0	XMIT-T		
E1	XMIT-R		
IRQ_N	BUTTON_SELECT		

Appendix L

Assembly Code