

A Simple Breath Alcohol Indicator

A Laboratory Protocol

May 31, 2007

James K Beard, Ph.D.
Assistant Professor
Rowan University
beard@rowan.edu
jkbeard@comcast.net

Concept by:
Dr. Robi Polikar
polikar@rowan.edu

Table of Contents

1	Introduction.....	1
1.1	About this Lab.....	1
1.2	Background.....	1
2	The Breath Alcohol Indicator	2
2.1	Blood alcohol content	2
2.2	The breath analyzer concept	3
2.3	Calibration and detection	4
3	The Base Concept of an Auto-Ranging Ohmmeter	5
3.1	The digital Ohmmeter concept.....	6
3.1.1	Voltage measurements are used to compute resistances.....	6
3.1.2	Accuracy of digital Ohmmeters	6
3.2	Microcontroller based autoranging	7
3.3	Measuring the ON resistance of the CD4066B.....	10
4	The Sensor and the Circuit.....	11
4.1	Sensor description.....	11
4.2	Sensor capabilities for other gases.....	13
4.3	The sensor circuitry.....	13
4.4	Integrating the sensor circuitry with the microcontroller	15
4.4.1	The array of calibration resistors for autoranging.....	15
4.4.2	The microcontroller circuitry	16
5	Constructing the circuit.....	17
5.1	Mounting the sensor.....	17
5.2	Building the circuit	17
5.3	Calibration.....	17
5.4	Testing the breath alcohol indicator.....	17
6	Laboratory Instructions.....	18
6.1	Equipment list.....	18
6.2	Procedure	18
6.2.1	Assemble the sensor and cable	19
6.2.2	Build the circuit of Figure 9.....	19
6.2.3	Add the microcontroller, pushbuttons, and LEDs	19
6.2.4	Correct and update the microcontroller software.....	19
7	The Report	20
8	Appendix A: Additional Considerations for a Field Breath Alcohol Indicator	21
8.1	Power Supply	21
8.2	LCD Display	21
8.3	Temperature and humidity compensation.....	22
9	Appendix B: Calibration Procedure.....	23
9.1	The calibration process	23
9.2	Formula for the 0.08% alcohol solution for calibration.....	24
10	Appendix C: The Microcontroller program.....	24
10.1	Requirements	24
10.2	Allocation of requirements to hardware	25
10.3	The Turing table.....	25
10.4	The Initialization mode.....	28

10.5	The Normal Operation and Medical Alert modes.....	28
10.5.1	Computations	29
10.5.2	Actions	29
10.5.3	Autoranging	29
10.5.3.1	Boundaries between ranges in an autoranging Ohmmeter	29
10.5.3.2	Building your table of threshold resistances for your microcontroller code	30
10.6	The Medical Alert mode	31
11	Appendix D: Microcontroller Program Listing	31
	References.....	37

List of Figures

Figure 1.	Blood alcohol content for males vs. number of drinks (see text). Subtract 0.01% for each 40 minutes since drinking began.	2
Figure 2.	Blood alcohol vs. number of drinks (see text) for females. Subtract 0.01% for each 40 minutes since drinking began.	3
Figure 3.	Simple Ohmmeter Concept.	7
Figure 4.	Using an Electronic Range Switch in an Ohmmeter	8
Figure 5.	Pin-Out of the CD4066B. Use pins 2, 3, 9 and 10 as inputs.	9
Figure 6.	On Resistance of CD4066B from [6].	10
Figure 7.	Sensor Resistance versus Gas Concentration from [8].....	12
Figure 8.	TGS 2620 Specifications and pin out from [8].	13
Figure 9.	Sensor circuitry for our laboratory. Note that pin-out of sensor is for top view. Use CD4066B pins 2, 3, 9 and 10 as inputs.....	15
Figure 10.	Pin-Out of the PIC Microcontroller 16F876A.....	16
Figure 11.	Sensor variation in temperature and humidity from [8].	23

List of Tables

Table 1.	Alcohol vapor concentration in breath as a function of BAC from [5].	4
Table 2.	Nominal Calibration Resistances	15
Table 3.	Operation times for PIC 16Fxxxx with 10 MHz clock.....	20
Table 4.	Top Level Turing Table for Breath Analyzer Software.	27
Table 5.	Table of Nominal Resistances for Autoranging Thresholds.....	30

Change Log

Date	Author	Changes	Sections Affected
May 30, 2007	J. K. Beard	Original Draft	ALL
June 1, 2007	J. K. Beard	Mode/ Display Update, add table of PIC operation times, minor changes	2.2, Fig. 9, 6.2.4, Appendices C, D

A Simple Breath Alcohol Indicator

1 Introduction

1.1 About this Lab

We will build and test a simple hydrocarbon detector. Three teams will configure their project as a Breath Alcohol Indicator, a field device for fast, simple estimates of blood alcohol levels, whereas the other two teams will configure their project as an alarm for traces of natural gas. You are in the former group.

1.2 Background

In 2004, there were about 17,000 alcohol related traffic fatalities in the U.S., which represented about 40% of about 42,500 all traffic fatalities. This represents one alcohol-related fatality every 31 minutes. There were also 248,000 people were injured in traffic accidents where alcohol was involved – about one every 2 minutes! Perhaps more disturbingly, in 85% of all alcohol-related fatalities, the blood-alcohol concentration (BAC) of those involved in the accident was reported to be above 0.08%, the legal limit in the U.S. About 51% had a BAC over 0.16, a level at which an individual is typically severely intoxicated¹. As high and disturbing these numbers are, there is a bit of good news in this: this numbers represent about 33% drop in alcohol related fatalities (from 26000) in 1982². While the reasons for this drop is many, one could include toughened traffic laws and the now ubiquitous use of breath analyzers that can be used on the spot to test near-objectively test the individuals level of intoxication.

Breath analyzers (note that a Breathalyzer³ is a particular brand name) are now available even for consumers who can purchase these and test their BAC at any time. These devices, like countless other consumer or medical grade biomedical equipments are designed by biomedical engineers who typically have an electrical and computer engineering background. The goal of this laboratory exercise is to give you a glimpse of this exciting and rapidly growing area of biomedical engineering, where engineering solutions are sought for real-world problems in medicine. In this lab, you will build a prototype of such a breath analyzer.

On the other hand, no concept of biological origin – including designing devices for biomedical applications – can be fully appreciated if such an exercise is divorced from the underlying physiological underpinnings. It is therefore important to at least have some understanding of the physiological affects of alcohol on humans, more specifically, on the central nervous system. This physiological effect and the associated impairment that causes ten thousands of alcohol related traffic fatalities. You are therefore asked to read about the central nervous system and the effect of alcohol on this system before coming to lab. A short review will also be provided by Dr. Polikar.

2 The Breath Alcohol Indicator

2.1 Blood alcohol content

An estimate of BAC can be made from the charts⁴ shown below as Figure 1 and Figure 2. As a notional sensor, the Breath Alcohol Indicator as we are building it today can provide a quick check about whether you are in the red zone, or near the red zone, in the event that you need to drive. As a safety accessory, this may be the convincing data that a taxicab or other alternative to driving is appropriate. Detailed examination of [4], published as a web page by a campus police organization, and says repeatedly in the most direct possible terms that driving with any alcohol in your system is ill advised.

APPROXIMATE BLOOD ALCOHOL PERCENTAGE									
Drinks *	BODY WEIGHT IN POUNDS								EFFECT ON PERSON
	100	120	140	160	180	200	220	240	
0	.00	.00	.00	.00	.00	.00	.00	.00	ONLY SAFE DRIVING LIMIT
1	.04	.03	.03	.02	.02	.02	.02	.02	IMPAIRMENT BEGINS.
2	.08	.06	.05	.05	.04	.04	.03	.03	
3	.11	.09	.08	.07	.06	.06	.05	.05	DRIVING SKILLS SIGNIFICANTLY AFFECTED.
4	.15	.12	.11	.09	.08	.08	.07	.06	
5	.19	.16	.13	.12	.11	.09	.09	.08	LEGALLY INTOXICATED. CRIMINAL PENALTIES IN <u>ALL</u> STATES **
6	.23	.19	.16	.14	.13	.11	.10	.09	
7	.26	.22	.19	.16	.15	.13	.12	.11	
8	.30	.25	.21	.19	.17	.15	.14	.13	
9	.34	.28	.24	.21	.19	.17	.15	.14	
10	.38	.31	.27	.23	.21	.19	.17	.16	

Figure 1. Blood alcohol content for males vs. number of drinks (see text). Subtract 0.01% for each 40 minutes since drinking began.

APPROXIMATE BLOOD ALCOHOL PERCENTAGE										
Drinks *	BODY WEIGHT IN POUNDS									EFFECT ON PERSON
	90	100	120	140	160	180	200	220	240	
0	.00	.00	.00	.00	.00	.00	.00	.00	.00	ONLY SAFE DRIVING LIMIT
1	.05	.05	.04	.03	.03	.03	.02	.02	.02	IMPAIRMENT BEGINS.
2	.10	.09	.08	.07	.06	.05	.05	.04	.04	DRIVING SKILLS SIGNIFICANTLY AFFECTED.
3	.15	.14	.11	.11	.09	.08	.07	.06	.06	
4	.20	.18	.15	.13	.11	.10	.09	.08	.08	LEGALLY INTOXICATED. CRIMINAL PENALTIES IN <u>ALL</u> STATES **
5	.25	.23	.19	.16	.14	.13	.11	.10	.09	
6	.30	.27	.23	.19	.17	.15	.14	.12	.11	
7	.35	.32	.27	.23	.20	.18	.16	.14	.13	
8	.40	.36	.30	.26	.23	.20	.18	.17	.15	
9	.45	.41	.34	.29	.26	.23	.20	.19	.17	
10	.51	.45	.38	.32	.28	.25	.23	.21	.19	

Figure 2. Blood alcohol vs. number of drinks (see text) for females. Subtract 0.01% for each 40 minutes since drinking began.

2.2 The breath analyzer concept

The breath analyzer concept is to use a simple hydrocarbon sensor that presents measurements of alcohol vapor concentration as a resistance with a simple microcontroller. The analog to digital converters incorporated into the microcontroller are used to measure the voltages across a resistor in series with the sensor and the voltage across the sensor, and Ohm's law and simple network theory is used to find the sensor resistance. Sensor data is the used to provide a blood alcohol content (BAC) estimate. This estimate is used to light LEDs for four stages:

- Onset of impairment for 0% to 0.04% BAC,
- Warning for 0.04% to 0.08%,
- No Drive for 0.08% to 0.12%,

- Red Alert for 0.12% to 0.16%, and
- Medical Alert -- blinks all four LEDs for BAC over 0.16% BAC.

These four LEDs form a bar graph, with the highest warning blinking with a duty cycle proportional to the BAC level between its level and the next lower level. The bar graph changes with the alcohol vapor concentration at the sensor, except the Medical Alert mode, which "sticks" and must be reset with a pushbutton.

Because the sensor can vary widely in base resistance, and because it provides a very wide range of resistances as alcohol vapor concentration varies, a single voltage divider will not provide sufficient accuracy. Two measures are taken to provide high accuracy in our laboratory project: a calibration mode that provides a data point at the most critical point, 0.08% BAC – the legal limit for driving under the influence in most states – and use of the capability of the microcontroller to switch resistances that are used on the voltage divider. The sensor resistance is measured with an autoranging ohmmeter to maintain best accuracy over the entire range of capability of the sensor.

The breath analyzer presented here is defined as appropriate for a laboratory. For considerations in building a field instrument or product, please refer to Appendix A: Additional Considerations for a Field Breath Alcohol Indicator.

2.3 Calibration and detection

Calibration of the Breath Alcohol Indicator is summarized as follows; see Appendix B: Calibration Procedure for details of calibration. The sensor is stabilized by waiting at least two minutes after power is applied, then exposing the sensor to air containing the critical amount of alcohol vapor designated for calibration, then pressing the Calibration pushbutton and releasing it quickly. This should be done within two or three seconds of exposure to the test alcohol vapor concentration to best replicate actual usage of the sensor.

Table 1. Alcohol vapor concentration in breath as a function of BAC from [5].

In Breath		In Blood		
mg/liter	ppm	mg/100 ml	mg/cc	%
0.05	26	10	0.1	0.01
0.10	52	20	0.2	0.02
0.20	104	40	0.4	0.04
0.25	130	50	0.5	0.05
0.30	156	60	0.6	0.06
0.40	208	80	0.8	0.08
0.50	260	100	1.0	0.10
0.60	312	120	1.2	0.12
0.70	364	140	1.4	0.14
0.80	416	160	1.6	0.16
0.90	468	180	1.8	0.18
1.00	520	200	2.0	0.20

The relationship between BAC and alcohol vapor concentration in breath is given in a Figaro application note⁵. We repeat that information as Table 1 above. Note that 300

ppm, the alcohol vapor concentration used most often as a calibration limit in Figaro documentation corresponds to about 0.115% BAC, but a more modern critical value of 208 ppm, corresponding to a BAC of 0.08% is more appropriate for a Breath Alcohol Indicator. Note that Table 1 the BAC in percent is obtained from alcohol vapor content in breath by dividing by 2600.

For a Breath Alcohol Indicator calibrated this way, the output can be computed using information from Figaro technical notes [5] and data sheets for the TGS 2620 [8]. Table 1 and Figaro information are used in the microcontroller software that computes the concentration in PPM and converts to BAC. The highest BAC measured is retained until Reset is pressed.. The retained BAC is compared to thresholds and these results are then used to light and blink appropriate LEDs:

- The green Ready LED blinks at low BAC and comes on steady at BAC 0.04% for the Onset of Impairment indication,
- The yellow Warning LED begins to blink at 0.04% and comes on steady at 0.08%,
- The red No Driving LED begins to blink at 0.08% and comes on steady at 0.12%,
- The Red Alert LED begins blinking at 0.12%, and comes on steady as BAC nears 0.16%, and
- For a BAC over 0.16% all the LEDs flash; we call this the Medical Alert mode.

Thus some impairment is evident if any change in the green LED is seen, and the breath analyzer shows a BAC that is grounds for conviction in any Zero Tolerance environment such as underage drinking, learner's permit driving license.

If any indication is seen in the yellow LED, the breath analyzer is measuring a BAC over 0.04%.

If any indication is seen in the red No Driving LED, the BAC that the breath analyzer is measuring is grounds for a DUI conviction in most States. Driving is strongly proscribed if any activity is seen in the yellow LED.

If any activity is seen in the Red Alert LED, a BAC over 0.12% is measured by the breath analyzer. This BAC range corresponds to serious intoxication.

If all the LEDs blink, the breath analyzer is measuring over 0.16% BAC, which is twice the legal limit that is grounds for a DUI conviction in most states. A case of alcohol poisoning is in progress. Since BAC can increase for up to an hour after drinking has ceased, this is an indication that a trip to the emergency room should be considered.

3 The Base Concept of an Auto-Ranging Ohmmeter

Our basic concept is to use a microcontroller to configure an ohmmeter, and to measure the resistance of a sensor. The way that this is accomplished is to use a CMOS analog switch, the CD4066B, to select the configuration of the ohmmeter.

3.1 The digital Ohmmeter concept

3.1.1 Voltage measurements are used to compute resistances

Refer to Figure 3 for reference. There, a calibration resistance R_C is placed in series with an unknown resistance R_U . The voltage V_A applied to the two resistors, denoted as node A, and the voltage V_B between the two resistors, denoted as node B, are measured. Digital Ohmmeters measure the current through the calibration resistance using V_A , V_B , R_C and Ohm's law, and the unknown resistance by measuring the voltage across the calibration resistance for this current, again using Ohm's law. The result is

$$(3.1) \quad R_U = \frac{V_B}{V_A - V_B} \cdot R_C = \frac{\frac{V_B}{V_A}}{1 - \frac{V_B}{V_A}} \cdot R_C.$$

3.1.2 Accuracy of digital Ohmmeters

The accuracy of an Ohmmeter can be evaluated by looking at the sensitivity of the measured value to the Ohmmeter parameters. These sensitivities are partial derivatives of R_U with respect to V_A , V_B , (or V_B/V_A), and R_C in (3.1). These sensitivities are

$$(3.2) \quad \left\{ \begin{array}{l} \frac{\partial R_U}{\partial V_A} = -\frac{V_B}{(V_A - V_B)^2} \cdot R_C = -\frac{R_U}{V_A - V_B} \\ \frac{\partial R_U}{\partial V_B} = \frac{V_A}{(V_A - V_B)^2} \cdot R_C = \frac{V_A}{V_B \cdot (V_A - V_B)} \cdot R_U \\ \frac{\partial R_U}{\partial R_C} = \frac{V_B}{V_A - V_B} = \frac{R_U}{R_C}. \end{array} \right.$$

We see from (3.2) that the accuracy of the calibration resistor maps directly into the accuracy of the resistance measurement as expected.

The accuracy of a digital Ohmmeter is shown by (3.2) to be best when V_B is about half of V_A , and that the resistance reading will be as accurate as the calibration resistor and about two bits less accurate than the voltage measurement V_B . Since voltages can be measured to accuracies of 10 to 16 bits easily and inexpensively, a digital Ohmmeter can be as accurate as the smallest of the resistance of the calibration resistor R_C or two bits less than the ADC word length.

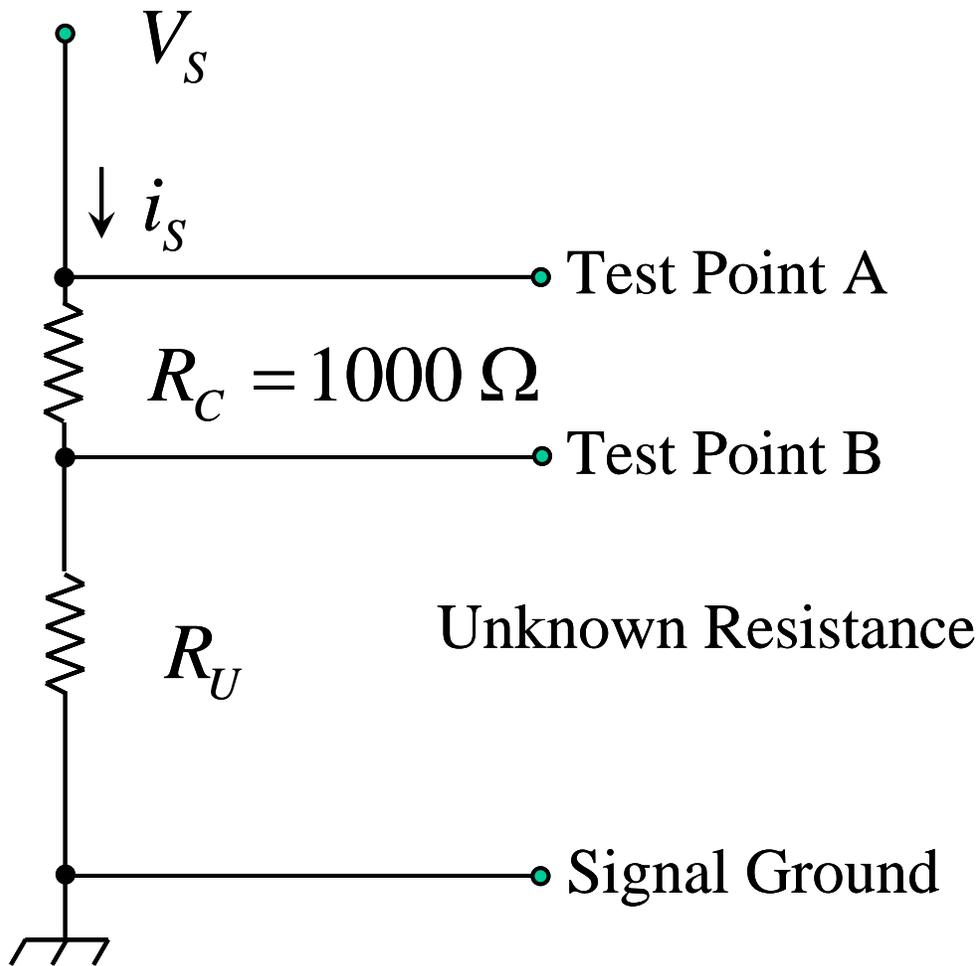


Figure 3. Simple Ohmmeter Concept.

3.2 Microcontroller based autoranging

With an inexpensive PIC microcontroller, we can measure V_A and V_B to 10 bits of accuracy with the on-board analog to digital converters (ADCs). The PIC can also accept pushbutton or single-bit digital inputs, and can drive a three-wire RS-232 or Controller Area Network (CAN) bus or other two-way digital communication. The PIC can also drive LCD displays very easily, and software support to do this is included with the PICC compilers available at Rowan.

We have seen that we need to adjust the calibration resistance R_C to keep the voltage V_B about half of the source voltage V_A . We do this electronically by using the binary outputs of the microcontroller to operate a CMOS switch. The basic circuit is shown in Figure 4. The switch signal comes directly from an output pin of the microcontroller. Several calibration resistors are connected to the test resistor, and only one electronic switch is ON at any time.

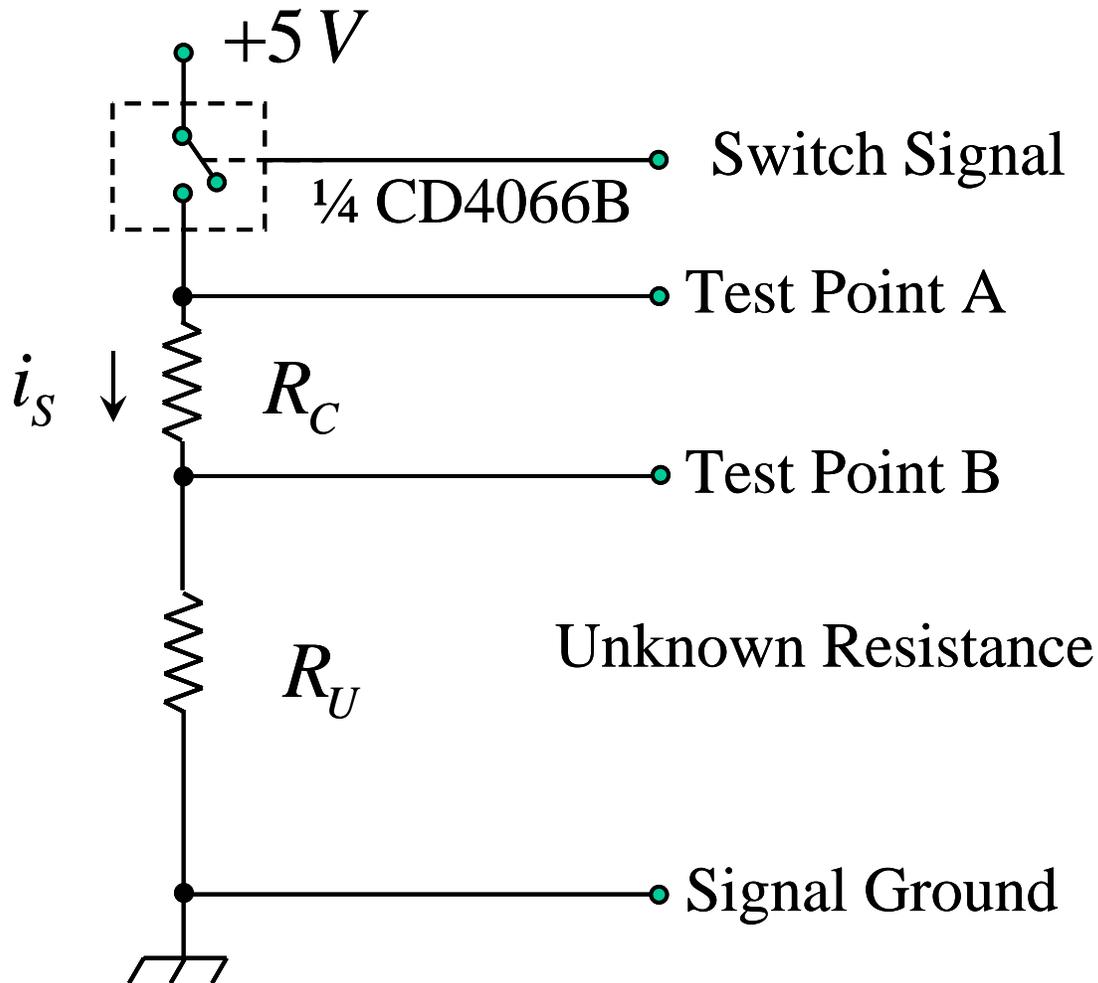


Figure 4. Using an Electronic Range Switch in an Ohmmeter

The ON resistance of the CD4066B from the T.I. data sheet⁶ is shown in Figure 6. Note that the lowest ON resistance is obtained when the signal voltage is at the positive supply voltage, which indicates that the configuration shown in Figure 4 will give the lowest ON resistance for a given device. The data sheet[6] states that the maximum ON resistance is about twice the typical ON resistance as given, so from Figure 6 we see that $500\ \Omega$ is an upper bound for ON resistance when we use the configuration of Figure 4. We maintain accuracy by sensing V_A when the calibration resistance is less than about $25\ k\Omega$.

The data sheet[6] states that minor startup issues with coupling of the supply voltage with the output can be avoided by using pins 2, 3, 9 and 10 as the inputs to the switches. The pin-out of the CD4066B is shown in Figure 5.

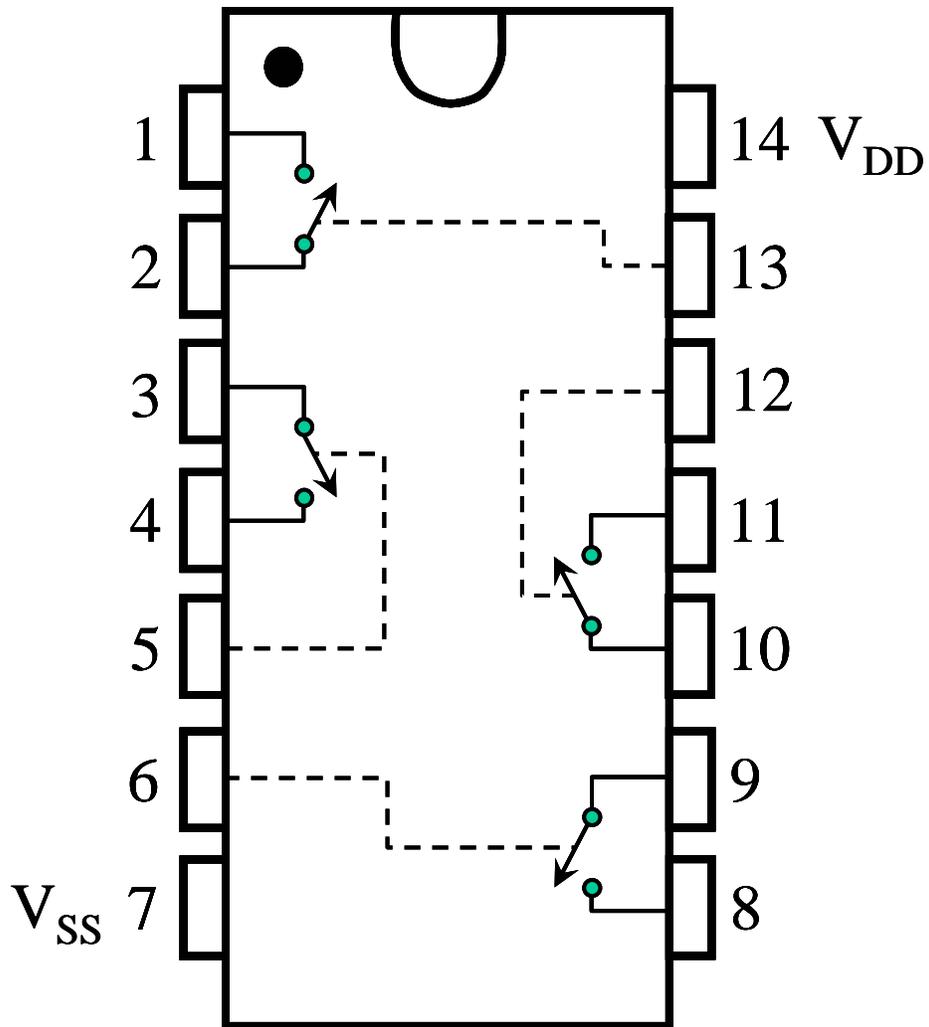


Figure 5. Pin-Out of the CD4066B. Use pins 2, 3, 9 and 10 as inputs.

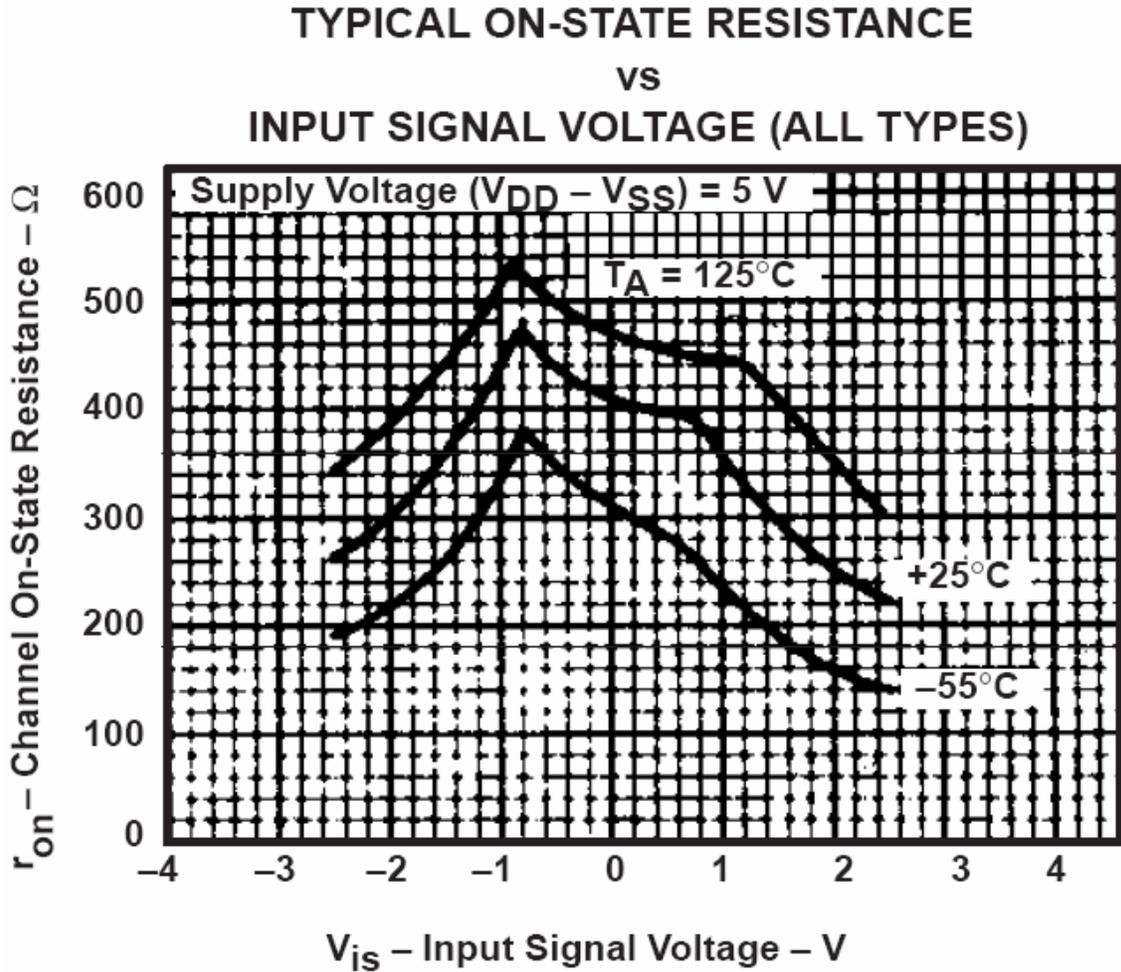


Figure 6. On Resistance of CD4066B from [6].

3.3 Measuring the ON resistance of the CD4066B

Build the circuit of Figure 4 and use a resistance R_C of $1000\ \Omega$, and a short circuit for the resistance under test R_U . Measure the output of the power supply to get an exact number for V_S , using your bench DVM and record the voltage to as high an accuracy as possible. Use the same voltmeter to measure V_A . Since we are using a short circuit for R_U than V_B will be zero. Use the bench multimeter to measure the exact resistance of R_C . Then, the ON resistance of the switch under test is

$$(3.3) \quad R_{ON} = \left(\frac{V_S}{V_A} - 1 \right) \cdot R_C.$$

Record this value; you will be using it in the microcontroller software. The value should be about $220\ \Omega$. If it exceeds $500\ \Omega$ then put the CD4066B aside and ask for another.

4 The Sensor and the Circuit

4.1 Sensor description

The Figaro⁷ TGS 2620 is a low-cost sensor for hydrocarbon vapors. Figaro mentions⁸ alcohol sensors, organic vapor detectors and alarms, and solvent detectors for factories, dry cleaners, and semiconductor industries.

This sensor is a thick film metal oxide semiconductor⁹ such as tin oxide, SnO₂, with a heating element. When the metal oxide is heated in air, donor electrons in the crystal surfaces transfer to adsorbed oxygen, generating a coating of negatively charged oxygen ions. Capacitive charge-balancing effects generate a positively charged space charge layer in the semiconductor, which pinches off the conductive layer and reduces current flow. When a gas is present that can react with the layer of oxygen ions, this decreases the number of oxygen ions which in turn reduces the space charge and increases current flow. The sensor must be warmed up and allowed to stabilize to achieve the desired accuracy, but if this is done, very reliable results are possible for long periods of usage. Warm-up time for the TGS 2620 is given as two minutes, once initial on-time exceeds a stabilization time of about an hour; initial stabilization for first-time use is given in Figaro documentation as two days ([9], Section 2-7 page 7).

The equation for sensor resistance as a function of the gas is

$$(4.1) \quad R_S = R_{CAL} \cdot \left[\frac{C}{C_{CAL}} \right]^{-\alpha}$$

where

- R_S = Resistance of the sensor
- R_{CAL} = Resistance at calibration point
- C = Gas concentration
- C_{CAL} = Gas concentration at calibration point
- α = Slope of the R_S curve on a log-log plot.

The measured resistance of the sensor versus gas concentration[8] is shown below in Figure 7. The gas concentration is found by solving (4.1) for C,

$$(4.2) \quad C = \left[\frac{R_S}{R_{CAL}} \right]^{-\frac{1}{\alpha}} \cdot C_{CAL}$$

The specifications of the TGS 2620, taken from [8], are shown below as Figure 8.

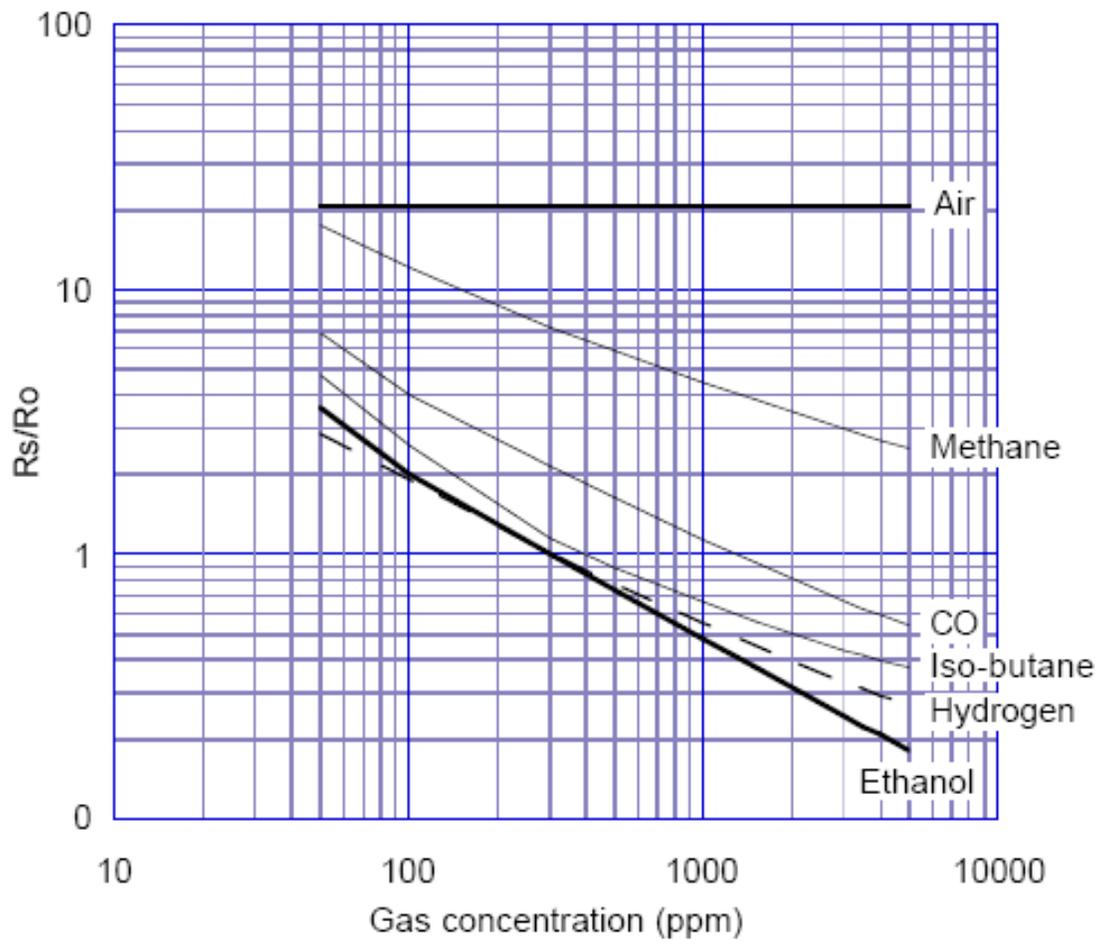


Figure 7. Sensor Resistance versus Gas Concentration from [8]

Specifications:

Model number		TGS 2620	
Sensing element type		D1	
Standard package		TO-5 metal can	
Target gases		Alcohol, Solvent vapors	
Typical detection range		50 ~ 5,000 ppm	
Standard circuit conditions	Heater Voltage	V_H	$5.0 \pm 0.2V$ DC/AC
	Circuit voltage	V_C	$5.0 \pm 0.2V$ DC/AC $P_s \leq 15mW$
	Load resistance	R_L	Variable $0.45k\Omega$ min.
Electrical characteristics under standard test conditions	Heater resistance	R_H	83Ω at room temp. (typical)
	Heater current	I_H	$42 \pm 4mA$
	Heater power consumption	P_H	approx. 210mW
	Sensor resistance	R_s	$1 \sim 5 k\Omega$ in 300ppm ethanol
	Sensitivity (change ratio of R_s)		$0.3 \sim 0.5$ $\frac{R_s(300ppm)}{R_s(50ppm)}$
Standard test conditions	Test gas conditions	Ethanol vapor in air at $20 \pm 2^\circ C$, $65 \pm 5\% RH$	
	Circuit conditions	$V_C = 5.0 \pm 0.01V$ DC $V_H = 5.0 \pm 0.05V$ DC	
	Conditioning period before test	7 days	

Structure and Dimensions:

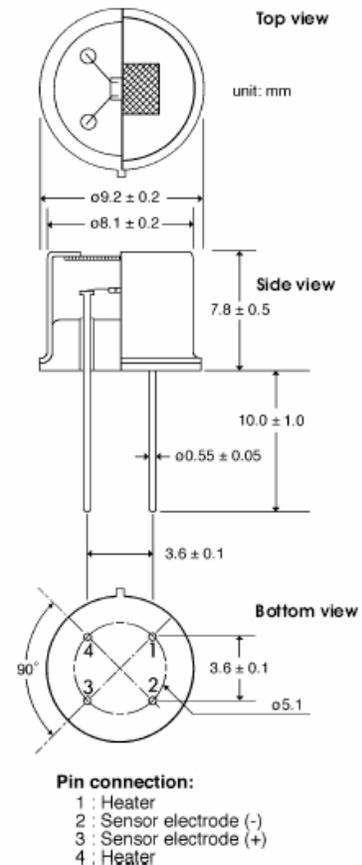


Figure 8. TGS 2620 Specifications and pin out from [8].

4.2 Sensor capabilities for other gases

The TGS 2520 is capable of sensing any gas that can be catalyzed to react with oxygen ions at a moderate temperature, including carbon monoxide, although it is optimized to sense hydrocarbon vapors. Since the sensor characteristics are functions of oxygen availability, relative humidity, and ambient temperature, highest accuracy is obtained using standard air conditions and temperature compensation. For our use, errors on the order of 10% are acceptable so this degree of control and compensation are not needed.

4.3 The sensor circuitry

This data sheet and technical information for the Figaro TGS 2620 sensor show designs using a voltage divider, with the sensor resistance at the voltage source. This provides an output which increases with gas concentration in a circuit of minimum complexity. The other resistor also serves as a power limiting resistor, ensuring that the power in the sensor itself never exceeds its maximum rating of 15 milliwatts. The equation for the minimum value of the Ohmmeter calibration resistance R_C to limit power to a sensor resistance

$$(4.3) \quad R_C \geq \frac{V_C^2}{4 \cdot P_{MAX}}.$$

For a supply voltage V_C of 5 Volts and a maximum load power of 15 milliwatts, the minimum load resistance is

$$(4.4) \quad R_L \geq 416.7 \, \Omega \text{ (5 Volts, 15 milliwatts)}.$$

so we will use a minimum calibration resistance of $500 \, \Omega$ with the ON resistance of the CD4066B as an added safety factor. The sensor circuit is shown below in Figure 9.

For every switch that is used on the CD4066B, the input pin coming from above or from the left in Figure 9, is one of pins 2, 3, 9 or 10 as we stipulated in Section 3.2.

The data sheet information shown in Figure 8 shows that both the base coefficient R_{CAL} and the sensitivity coefficient α in (4.1) vary between sensors, particularly the base coefficient. Thus each sensor must be calibrated before use, and the software in the microcontroller must provide a way to do this. The value of α also varies significantly as shown by the related value β that is given in the data sheet [8],

$$(4.5) \quad \beta = \left[\frac{300}{50} \right]^{-\alpha}$$

The inverse of (4.5) is

$$(4.6) \quad \alpha = \frac{-\ln(\beta)}{\ln\left(\frac{300}{50}\right)} = \frac{-\ln(\beta)}{1.792}.$$

The range of values of α is 0.39 to 0.67, a ratio of about 7:4. For notional use, this variation is acceptable if the sensor is calibrated at the most critical value of C for the application, C_{CAL} , and use of the sensor is most accurate in a range of C centered on this value. From this rationale, and also observation of apparent values of α near 0.633 in Figure 7, we will use a notional value of 0.633 for α in this laboratory. Note that (4.1) breaks down for zero C so that sensor resistance in air is not useful in calibration. The data sheet gives a range of $1 \, k\Omega$ to $5 \, k\Omega$ for sensor resistance in 300 ppm of alcohol vapor; this has a median value of $2236 \, \Omega$, which corresponds to a sensor resistance of about $2800 \, \Omega$ at 208 ppm. This defines the calibration requirements for our sensors:

- A notional value of 0.633 for α and a default value of $2800 \, \Omega$ for R_{CAL} ,
- Ignore sensor resistance in clear air, and
- Calibrate the sensor for the most critical gas concentration for the intended use, e.g. 208 ppm, which corresponds to a BAC of 0.08%, the legal limit for DUI citations in most states.

The microcontroller software will initialize with a nominal calibration value for the parameter R_{CAL} in (4.1) of $2800 \, \Omega$. A pushbutton input to the microcontroller will measure the sensor resistance at that time and overstore this value.

4.4 Integrating the sensor circuitry with the microcontroller

4.4.1 The array of calibration resistors for autoranging

Use the resistor values in Table 2. Measure the resistances with your bench multimeter and record the values to as high an accuracy as you can; you will be using these values in the microcontroller software, and they are critical to the accuracy of the breath analyzer.

Table 2. Nominal Calibration Resistances.

Index	Nominal Calibration Resistance
0	500 Ω
1	1000 Ω
2	2000 Ω
3	5000 Ω
4	10 k Ω
5	20 k Ω
6	50 k Ω
7	100 k Ω

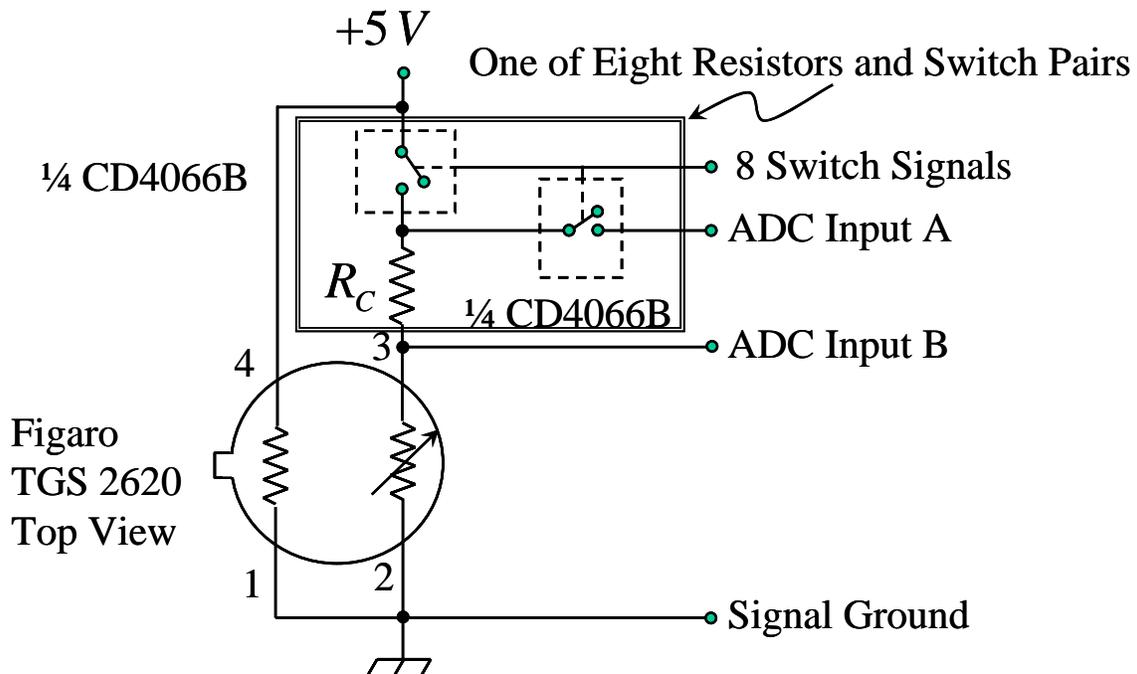


Figure 9. Sensor circuitry for our laboratory. Note that pin-out of sensor is for top view. Use CD4066B pins 2, 3, 9 and 10 as inputs.

4.4.2 The microcontroller circuitry

The microcontroller portion of the project is a PIC project board that hosts a PIC 16F876A microcontroller or equivalent. The PIC microcontroller has several buses denoted by letters A, B, and C for the 16F876A; larger microcontrollers have more buses. The pin-out of the PIC 16F876A is shown below as Figure 10.

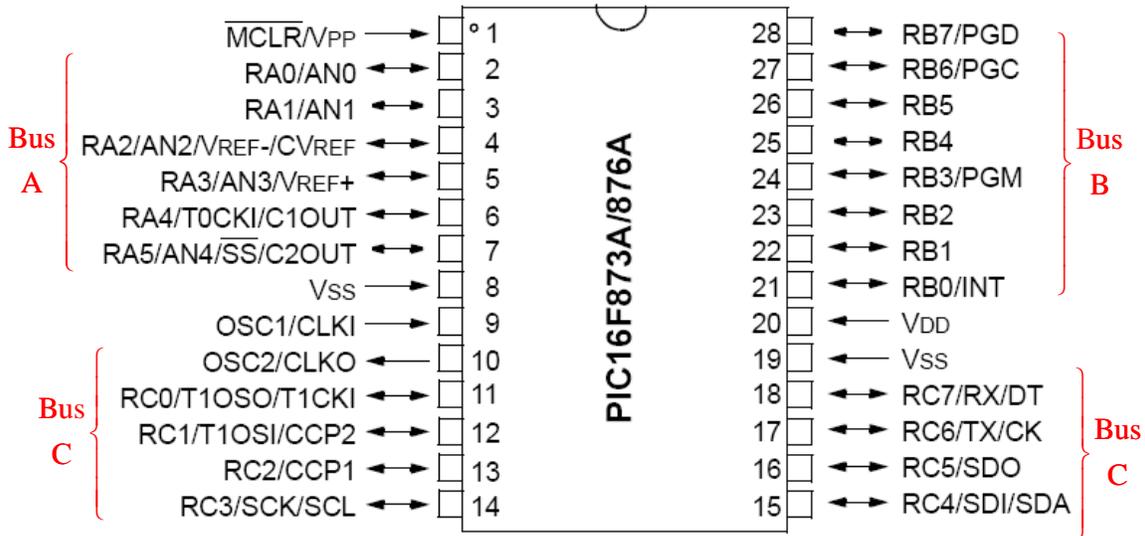


Figure 10. Pin-Out of the PIC Microcontroller 16F876A.

Your project board provides for connection to the buses of the microcontroller, provides power and clock signals to the microcontroller, and manipulates the master clear pin through the programming interface, so you don't need to be concerned about anything except the bus signals for this laboratory. For a portable stand-alone breath analyzer, the master clear pin must be connected to the 5 Volt power for the microprocessor to run, and either a clock crystal or resonator of 10 MHz must be connected between pins 8 and 9, or a clock signal of 10 MHz must be provided to pin 9.

Each bus has eight data lines, except the A bus which has five. The A bus can be used for analog inputs and has three 10-bit ADCs incorporated into the microcontroller.

Two of the A bus inputs are used as ADC Input A and ADC Input B in Figure 9. Bus B is used for output and drives the Switch Signals in Figure 9. The C bus is used for pushbutton inputs and LED outputs.

For the LED outputs, use an LM324 as a quad comparator and LED driver. Each LED will have a 470 Ω current-limiting resistor connected to the 5 Volt power supply and to the LED anode. The LED cathode will be connected to an op-amp output. A voltage divider that provides a 2.5 Volt reference signal that is applied to all the non-inverting op-amp inputs. The appropriate pins of the PIC microcontroller for specific C bus bits will be applied to the inverting inputs, so that a logical 1 on that bit will light the LED.

For more detail see Appendix C: The Microcontroller program and Appendix D: Microcontroller Program Listing.

5 Constructing the circuit

5.1 Mounting the sensor

In building the circuit, mount the sensor in a transistor socket. The leads on this sensor are Kevlar and are difficult to solder. Use a cable of about three feet or 1 meter length to allow manipulation of the sensor without moving the breadboard.

Be VERY CAREFUL in keeping the pin-out of the sensor correct and avoid any short circuits between the wires. *We have found by experience that a start-up current surge in the sensor will destroy it instantly if the sensor is connected cold directly across 5 Volts without a limiting resistor.* This sensor is very expensive and we don't have spares, so heed this warning. This warning applies to pins 2 and 3; note that the heater is on pins 1 and 4 and the heater is connected directly between 5 Volts and ground. This makes it especially important to watch out for solder bridges where the cable is connected to the transistor socket.

5.2 Building the circuit

Build up the rest of the circuit on a wireless breadboard. Begin by building the sensor circuit shown in Figure 9, and make sure that the signal is present and that the test point shows that the microcontroller is autoranging. Use the laboratory power supply on the 5 Volt setting.

5.3 Calibration

When powering up the circuit for the first time, make let the circuit settle for at least ten minutes in clear air. Ignore any LEDs that may be lit or blinking, except that on start-up the Ready LED should blink once a second for two minutes and then remain on.

When the ten minute initial sensor stabilization period is up, notify the instructor that your experiment is ready for calibration. The instructor will have ready a test apparatus made up of a solution containing a small amount of Listerine Antiseptic™ mouthwash in four ounces of bottled water, kept in a half-pint water bottle. The test apparatus is this test solution and a soda straw.

The calibration process requires that bubbles blown through the solution through a soda straw. The sensor is placed near the mouth of the water bottle to expose it to vapors from the test solution. This provides only a few seconds before the alcohol vapor concentration begins to decline. During this period, press the Calibrate button. The Ready LED will turn off for 1 second when calibration has been accomplished. Your breath analyzer is now ready for use.

5.4 Testing the breath alcohol indicator

The first test is to simply move the sensor near the mouth of the top of the mouthwash bottle and observe the warning lights. This establishes the capability of the breath alcohol indicator to detect alcohol vapor.

A second test is to have a team member sniff the mouthwash, breath normally for a few seconds, and then gently exhale through the mouth over the sensor. Any indication by

the breath alcohol indicator demonstrates how quickly alcohol can enter the bloodstream through vapors carried in the air.

Other tests will be discussed in the laboratory.

6 Laboratory Instructions

6.1 Equipment list

The equipment that you will need for this laboratory includes but is not necessarily limited to the following:

- A Figaro TGS 2620 hydrocarbon vapor sensor, a four-pin TO-5 transistor socket, and 1 meter of four-conductor ribbon cable.
- A lab computer with the Microchip MPLAB integrated development environment (IDE) installed.
- A license for the CCS PIC-C compiler for the PIC microcontroller used in your project, such as the PIC 16F876A. This compiler must be integrated with the MPLAB IDE and will be accessed through the IDE.
- A PIC project board that includes an in-circuit debugger (ICD) or other interface that allows the Microchip MPLAB IDE to accept and compile your code, debug it in-place on the project board, and compile it and program your microcontroller.
- Four CD4066B quad bilateral analog switches.
- One LM324 op-amp.
- Four LEDs.
- Assorted resistors, including those listed in Table 2 and a 470 Ohm pull-up for each LED, and 10 k Ω pull-up resistors for the pushbuttons..
- A bench power supply with a regulated 5 Volt output.
- A bench digital multimeter that accurately measures resistances and voltages.
- An oscilloscope to use with your multimeter for checking and testing your circuits as you build them.

6.2 Procedure

Before beginning to build the circuit, read this entire document and discuss it with your teammates. When at least one team member is comfortable with every part of the breath analyzer concept, analog hardware, digital hardware, and microcontroller software, you are ready to begin.

6.2.1 Assemble the sensor and cable

With the Figaro TGS 2620 sensor not plugged into the transistor socket, solder the four pins of the socket to the four conductors of your 1 meter cable. Be sure and keep the pin-out of the sensor marked so that you can complete the next step without a miswire.

6.2.2 Build the circuit of Figure 9

Be VERY CAREFUL in keeping the pin-out of the sensor correct and avoid any short circuits between the wires. *We have found by experience that a start-up current surge in the sensor will destroy it instantly if the sensor is connected cold directly across 5 Volts without a limiting resistor.* This sensor is very expensive and we don't have spares, so heed this warning. This warning applies to pins 2 and 3; note that the heater is on pins 1 and 4 and the heater is connected directly between 5 Volts and ground. This makes it especially important to watch out for solder bridges where the cable is connected to the transistor socket. Check the cable and pin-out with an Ohmmeter before applying power.

Use the pin-out in Figure 5 for the CD4066B, and use pins 2, 3, 9 and 10 as the input sides of the switches. Test the circuit as described in 5.2. When you are satisfied with the operation of this circuit, add the rest of the calibration resistors and analog switches to complete the circuit shown in Figure 9.

6.2.3 Add the microcontroller, pushbuttons, and LEDs

Connect the microcontroller bus B pins to the appropriate bilateral analog switches so that pins B0 through B7 control the calibration resistances from the lowest resistance to the highest in sequential order.

Connect points A and B to the microcontroller pins A0 and A1.

Connect the pushbuttons for Calibrate and Reset as momentary (closed when pushed) contacts to ground, with a 10 k Ω pull-up to 5 Volts.

Connect the LM324 to drive the four LEDs from the C bus pins C4, C5, C6 and C7. If your design uses more than four LEDs you will need a second LM324. Use the op-amps in the LM324 as directed in 4.4.2.

6.2.4 Correct and update the microcontroller software

Begin with the PIC-C program in Appendix C: The Microcontroller program and Appendix D: Microcontroller Program Listing. Use the in-circuit debugger (ICD) capability of the MPLAB IDE to verify correct operation of the program in autoranging, correctly measuring resistance, and responding properly to the pushbuttons. Use the calibration process described in 5.3 and detailed in Appendix B: Calibration Procedure. Using the PIC MPLAB ICD, obtain and record the value of R_{CAL} stored in the microcontroller.

The operation times for the PIC 16F876A are shown in Table 3¹⁰. Note that floating point operations in the microcontroller code, though fast enough for our purposes here, are slow to step through using the ICD in the debugger mode. Note that the exponentiation to the power of $1/\alpha$ requires a logarithm and an exponential; this 8.5 milliseconds is a lot of instructions to step through. The best way to cross lines of code

with floating point operations more complex than a comparison or multiplication is to set a breakpoint on the line of code following the expression and let the program run to the breakpoint.

Table 3. Operation times for PIC 16Fxxxx with 10 MHz clock.

Operation	With int	With long	With int32	With float
Add	1.2	2.8	6	222.6
Subtract	1.2	2.8	6	227.8
Multiply	22.2	94.4	264	356.6
Divide	46.4	141.6	478.4	661.8
exp(*)	NA	NA	NA	3394.6
ln(*)	NA	NA	NA	4035.4
sin(*)	NA	NA	NA	4369.0

7 The Report

Prepare a report due one week after the date of the laboratory. The guidelines for the report are on the course web site at

http://rowan.jkbeard.com/Electronics_I_ECE_Materials/Report_Guidelines.htm.

The calibration and testing of your Breath Alcohol Indicator shall be treated as a separate experiment that follows the building and testing of the electrical circuit.

In your report, describe the circuit of Figure 9 and the functions of each element of the circuit. Describe how you determined whether it worked properly. Describe the operation of the three op-amps that perform as comparators and LED drivers.

Document the software for the microcontroller. In particular, describe the calibration process. Your software documentation should include, but is not necessarily limited to, these elements:

- The requirements – what are the necessary functions of the breath analyzer? Describe what the microcontroller needs to do. Detailed microcontroller software requirements are given in Appendix C: The Microcontroller program and you may refer to this documentation without repeating it; updates, changes, additions, and breath analyzer requirements not given in Appendix C: The Microcontroller program should be given in your report.
- Allocate the functions necessary to provide the functionality to meet the requirements. For example, you need two ADC channels, eight discrete outputs to drive the CD4066B CMOS switches, at least two pushbutton inputs, and three LED outputs. Allocate these to the A, B and C buses. If these disagree with what is given in Appendix C: The Microcontroller program and Appendix D: Microcontroller Program Listing, modify the I/O mapping to meet your

requirements. Note that if you desire to use interrupts to log pushbutton inputs, this can be done using bus B of the PIC 16F876A but not bus C because interrupts on changes are available only for bus B on this microcontroller^{11,12}, so you will need to remap the I/O to use interrupts to service pushbuttons..

- Build a Turing table for the code in the operating loop that implements the Normal Operation and Medical Alert modes and make sure that you have an entry for every possible input. The simplest form of a Turing table is a two-column table; the left column of your table will include all possible inputs, and the right column will be the action that you want the microcontroller to perform, if any. Another form is inputs labeling rows and microcontroller states labeling columns, so that you have a separate Turing table for each state.
- Construct a simple flow chart of your software before you begin. The microcontroller software has an environment block, which has the `<include>` statements and the `#define` statements, the initialization block which is the code that is executed before the main loop begins, and a main loop where all the required functions are performed once the processor starts up. Do a separate flowchart for the loop that implements the Normal Operation and Medical Alert modes.
- A listing of your software must be included as an Appendix of your report. Refer to the listing in your software description. This may be simpler if the lines are numbered in your listing. Be sure and comment your software clearly and copiously so that it is easy to read with the report. If you use the software in this Protocol or start from it and modify it, note the modifications, and log them in the comments at the beginning of the source code file.

8 Appendix A: Additional Considerations for a Field Breath Alcohol Indicator

8.1 Power Supply

This laboratory is conducted using the bench power supply 5 Volt output. A portable power supply using AC input, batteries, or both will allow this project to be used as a portable breath analyzer.

8.2 LCD Display

A free 8-bit bus can be used to drive an LCD display. This display can be used to output BAC, to provide gas concentration in PPM, or provide text messages to the user. The project as given here uses all three of the buses available on the PIC 16F876A, so another microcontroller in a 40 pin package is needed to provide the I/O to drive an LCD display.

The CCS PIC C compiler comes with a large set of examples and drivers, including LCD.C, which is in the drivers folder. Code from that file can be incorporated into your microcontroller program. This will make available to you these functions:

- `lcd_init()`: Must be called before any other function.

- `lcd_putc(c)`: Will display `c` on the next position of the LCD.
- `lcd_gotoxy(x,y)`: Set write position on LCD (upper left is 1,1)
- `lcd_getc(x,y)`: Returns character at position `x,y` on LCD

The `lcd_putc(c)` function accepts these control sequences:

- `\f` Clear display
- `\n` Go to start of second line
- `\b` Move back one position

These functions use either the B or D bus. The breath analyzer uses the B bus and the 16F876A does not have a D bus, but 40-pin devices such as the PIC 18F4520 will support all the functions of the breath analyzer plus operation of an LCD display on the D bus.

With a two-line LCD display, the breath analyzer can display the BAC on one line and the mode on the second line. A typical LCD display has two lines of 40 characters each, which is enough room to display both BAC and the gas concentration in ppm, and the results of applying thresholds to the BAC levels measured.

8.3 Temperature and humidity compensation

The variation in the sensor output as functions of temperature and humidity is shown below in Figure 11, which is taken from [8].

Temperature compensation can be added through incorporation of a thermistor in the voltage divider, as discussed in [9]. In a microcontroller solution, a temperature sensor such as the National LM35 series can provide a separate analog input for temperature compensation. A humidity sensor such as the Humirel HS 1100 requires additional circuitry and significant complication, since this and most other low cost humidity sensors are capacitive, and measurement of humidity is essentially translated into a capacitance measurement problem, but this is an added opportunity if a microcontroller based Breath Alcohol Indicator is used.

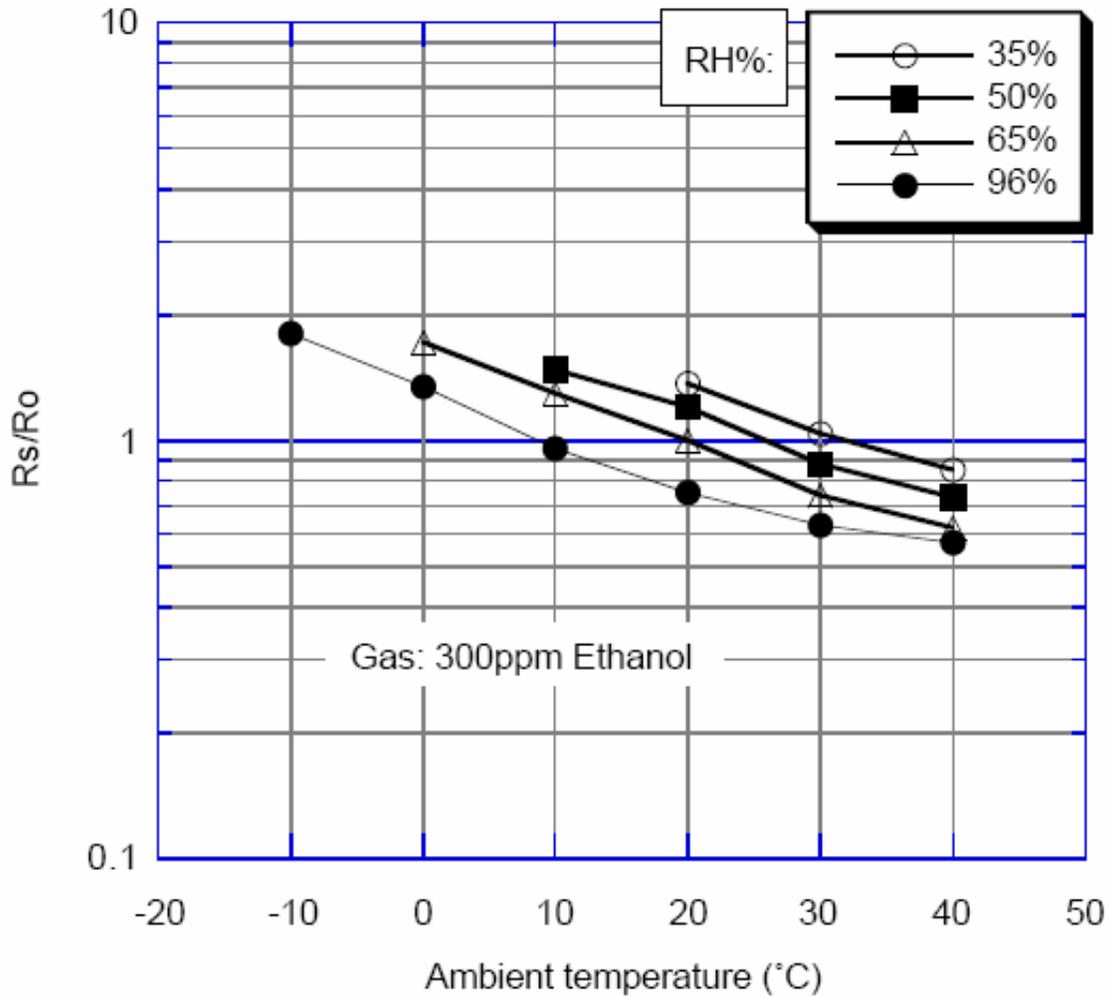


Figure 11. Sensor variation in temperature and humidity from [8].

9 Appendix B: Calibration Procedure

9.1 The calibration process

The Figaro documentation recommends a procedure modeled after commercial breath alcohol analyzer calibrators [5]. Figaro recommends that bubbles blown through a solution of 0.1% alcohol maintained at a temperature of 34 C (93.2 F) about 3 C cooler than nominal body temperature of 37 C, will produce air containing 300 ppm alcohol vapor. Our calibration procedure is to construct a solution containing .08% alcohol and fill a narrow-necked pint water bottle about half full with it. The bottle is then shaken, and the air in the top half of the bottle is a reasonable simulation of human breath from a person with 0.08% BAC, which is 208 ppm, our calibration point. Air is blown into the water through a straw, and the air in the top half of the bottle is displaced out the neck of the bottle and made available to the TGS 2620 sensor for calibrating our breath analyzer.

9.2 Formula for the 0.08% alcohol solution for calibration

The calibration solution of 0.08% alcohol by volume is made by diluting 1 cc of any convenient product containing alcohol. For example, original formula Listerine mouthwash is labeled as containing 26.9% alcohol. The volume of alcohol in 1 cc of this product has $p \cdot V$ or 0.279 times 1 cc, or 0.269 cc of alcohol. The total volume after dilution is

$$(9.1) \quad \frac{V_{Alcohol}}{V_{TOTAL}} = 0.08\%$$

Thus we have

$$(9.2) \quad V_{TOTAL} = \frac{p \cdot (1 \text{ cc})}{BAC_{CAL}} = \frac{p}{0.0008} \text{ cc} = 1250 \cdot p \text{ cc} = 473.75 \text{ cc}$$

for this particular mouthwash. If other products containing alcohol are used to mix the calibration solution, use (9.2) to determine the proper dilution to obtain 0.08% alcohol by volume.

10 Appendix C: The Microcontroller program

10.1 Requirements

The code requirements are elements of a list of things that the software must do. We can begin with the obvious or top-level and develop derived requirements, which are more detailed requirements that follow from the top-level requirements. The top-level requirements are

- The program must use conditional compilation to support testing of the software and circuits without an alcohol atmosphere.
- The microcontroller must support inputs from the analog signals at points A and B in Figure 9.
- The microcontroller must accept push-button inputs for operation of the breath analyzer.
- The microcontroller must provide eight outputs to operate the 16 electronic switches in the CD4066B quad analog switches as shown in Figure 9.

We want the breath analyzer to have states of operation, and to perform certain functions, and the microcontroller must implement these operations through the I/O we have specified. These states and operations are

- The microcontroller will wait for two minutes while the sensor warms up and stabilizes. During this time, there will be a Ready LED that will blink once a second. When it is done, the ready light will stop blinking and stay on. During the waiting period, the calibration resistance used with the sensor will be the highest available resistance.

- During operation the microcontroller will monitor the resistance of the sensor. The microcontroller will use the voltage measurements at points A and B and compute the resistance of the sensor according to (3.1). The concentration of alcohol vapor will be determined from the resistance according to (4.2). The BAC will be estimated from the concentration of alcohol vapor in ppm by dividing by 2600 as stated in 2.2 and evidenced in Table 1 [5].
- The BAC estimate will be applied to a set of thresholds and a Warning LED will come on at BAC 0.04%, a No Driving LED will come on at 0.08%, and a Red Alert LED will come on at 0.12%. Only one LED will come on at a time, except for a BAC over 0.16% when all the LEDs flash until a reset button is pressed; this special mode is the Medical Alert mode..
- The calibration resistance will be changed as necessary to maintain the highest possible accuracy of measurement of sensor resistance. This is the autoranging function.

This brings us to definition of what operations we need to control with pushbuttons. We will have pushbuttons that tell the microcontroller to perform these actions:

- One pushbutton will cause the microcontroller to read the sensor and use the measurement to calibrate the sensor, and replace the default nominal calibration. This calibration will be non-volatile – that is, it will still be in effect if the breath analyzer is turned off, then turned back on at a later time. Pushing the calibration button again will repeat the calibration process, even if the calibration has been performed before.
- If the breath analyzer is in the Red Alert mode, a pushbutton will take it out of the Medical Alert mode and put it back in the normal mode.

10.2 Allocation of requirements to hardware

The A bus will be used for measurements of the voltages at points A and B in Figure 9. Pins A0 will be used for point A and pin A1 will be used for point B.

The B bus will be used to drive the CD4066B quad analog switches. Pin B0 will be used to turn on the lowest calibration resistance value, pin B1 will be used to turn on the next higher calibration resistance value, etc.

The C bus will be used for pushbutton inputs and for LED driver outputs. Pin C0 will be used for calibration. Pushbuttons will be active low, so they will be implemented as momentary contact to ground with a 10 k Ω pull-up to 5 Volts. Pin C1 will be used for the pushbutton that resets the Red Alert mode. The pins C4, C5, C6 and C7 will be used to drive the Ready, Warning, No Driving, and Red Alert LEDs, respectively.

10.3 The Turing table

We have three modes defined now: Initialization, Normal Operation, and Medical Alert. We have two pushbuttons defined, so now we are able to provide a top level Turing table of modes, inputs, and actions.

Note that the Turing table implements some functional features:

- The breath analyzer will not respond to the pushbuttons while in initialization. This is because the sensor data is not yet stabilized. The only exit to the Initialization mode is an elapsed time of two minutes from start-up of the microcontroller.
- The Calibrate button will work in either the Normal Operation mode or the Medical Alert mode. This allows calibration when the sensor parameters are far enough from the nominal default parameters so that the microcontroller goes into the Medical Alert mode with a calibration atmosphere.
- The reset button will not take the microcontroller out of the Medical Alert mode if a BAC of 0.16% is still being detected.

Table 4. Top Level Turing Table for Breath Analyzer Software.

States		Inputs			Action
Init-ializa-tion	Oper-ating	Reset	Cal-ibrate	BAC	
Y	X	X	X	X	Two minute wait for sensor warm-up; blink Ready LED, 50% duty
N	Y	X	Y	Should be 0.08% BAC calibration value	Calibrate sensor resistance: current measured value corresponds to 0.08% BAC
N	Y	Y	N	ANY	Reset measured BAC to current measured value
N	Y	N	N	ANY	Set measured BAC to highest of current or past measured values
N	Y	N	N	$0 \leq \text{BAC} < .04\%$	Blink green Ready LED once per second with duty $(\text{BAC}/0.04)$
N	Y	N	N	$.04\% \leq \text{BAC} < .08\%$	Ready LED on steady. Blink yellow Warning LED once per second with duty $(\text{BAC}-.04)/.04$
N	Y	N	N	$.08\% \leq \text{BAC} < .12\%$	Ready and Warning LEDs on steady. Blink red No Drive LED once per second with duty $(\text{BAC}-.08)/.04$
N	Y	N	N	$.12\% \leq \text{BAC} < .16\%$	Ready, Warning, and No Drive LEDs on steady. Blink red Red Alert LED once per second with duty $(\text{BAC}-.12)/.04$
N	Y	N	N	$\text{BAC} \geq 0.16\%$	Blink all LEDs once per two seconds with duty 50%

10.4 The Initialization mode

The initialization mode is the beginning of execution of the microcontroller when the breath analyzer is turned on. The operations include, but are not necessarily limited to, the following:

- Initialization of the B bus and setting the highest available calibration resistance R_{C7} as the ohmmeter configuration. This means that B0-B6 are set to zero and B7 is set to 1.
- Delay for two minutes to allow the sensor to warm up and stabilize. During this wait, the Ready LED is to blink once per second. It will be ON for 500 milliseconds, then OFF for 500 milliseconds for 120 counts.
- Initialization of all flags and variables that must be defined before entering the Normal Operation mode.

At the conclusion of the two minute warm-up wait, the Initialization mode exits to the Normal Operation mode.

10.5 The Normal Operation and Medical Alert modes

This mode is executed in a loop, and the Medical Alert mode is implemented as a submode of this mode and implemented in the same loop. A status flag is used to keep track of whether the breath analyzer is in the Medical Alert mode or not. A separate Turing table can be constructed for this mode.

The operations in the Normal Operation mode include, but are not necessarily limited to, the following:

- Operate the ohmmeter to measure the sensor resistance according to (3.1).
- If the sensor is open or shorted, the Ready LED will blink quickly. If the sensor is shorted, the duty of the Ready LED will be 33%; if the sensor is open the duty will be 67%. This will continue until the breath analyzer is turned off.
- Use the sensor resistance to compute alcohol vapor concentration in ppm according to (4.2).
- Conversion of the alcohol vapor concentration in ppm to BAC in percent is done by dividing the alcohol vapor concentration by 2600 according to the instructions of 2.2 and as inferred from Table 1 and [5].
- Check the Calibration button and update R_{CAL} with the measured resistance and C_{CAL} with the agreed-upon concentration, 208 ppm for calibration with a simulated BAC of 0.08%. When the calibration operation is completed, the green Ready LED is to be turned off for 1 second to acknowledge the calibration to the operator.
- Check the BAC against thresholds of 0.04%, 0.08%, 0.12% and 0.16% and turn on the appropriate LEDs in accordance with Table 4.

- Check the Reset button and, if pushed, take the current BAC as the largest past measured value.

10.5.1 Computations

The computations are to compute the sensor resistance according to (3.1), the alcohol vapor concentration in ppm according to (4.2), and the BAC by dividing the alcohol vapor concentration in ppm by 2600 as described in 2.2 [5].

10.5.2 Actions

The actions result from an open or shorted sensor, inputs, a time-out of the initialization and sensor warm-up time, and thresholds of BAC. Actions include turning on or blinking LEDs and autoranging the ohmmeter configuration as the sensor resistance changes.

10.5.3 Autoranging

10.5.3.1 Boundaries between ranges in an autoranging Ohmmeter

Autoranging is a function of the microcontroller, and is implemented in the Normal Operation and Red Alert modes because the sensor resistance is being monitored in both modes.

The purpose of autoranging is to maintain the highest possible accuracy of the resistance measurement function. From (3.2) we infer that the resistance measurement is most accurate when the calibration resistance R_C is near the measured sensor resistance R_U , and from the sensitivity of the measured resistance R_U to the sensor voltage V_B , a good measure of error is seen to be

$$(10.1) \quad J = \frac{V_B}{V_A} \cdot \left(1 - \frac{V_B}{V_A} \right).$$

From Ohm's law we know that

$$(10.2) \quad \frac{V_B}{V_A} = \frac{R_U}{R_C + R_U}$$

so that our accuracy measure is

$$(10.3) \quad J = \frac{R_C \cdot R_U}{(R_C + R_U)^2}.$$

We want to move from one range to another, that is to change the calibration resistance R_C , when this cost function is the same for two adjacent values of R_C . We set the cost function equal for two values of R_C to find the value of R_U that defines the demarcation of the regions where one is better than the other,

$$(10.4) \quad \frac{R_{C1} \cdot R_{U12}}{(R_{C1} + R_{U12})^2} = \frac{R_{C2} \cdot R_{U12}}{(R_{C2} + R_{U12})^2}.$$

When rearranged,

$$(10.5) \quad R_{C1} \cdot (R_{C2} + R_{U12})^2 = R_{C2} \cdot (R_{C1} + R_{U12})^2$$

this equation can be expanded,

$$(10.6) \quad R_{C1} \cdot R_{C2}^2 + 2 \cdot R_{C1} \cdot R_{C2} \cdot R_{U12} + R_{C1} \cdot R_{U12}^2 = R_{C1}^2 \cdot R_{C2} + 2 \cdot R_{C1} \cdot R_{C2} \cdot R_{U12} + R_{C2} \cdot R_{U12}^2$$

the term with a factor of two cancels so we can take the terms in R_{U12}^2 on the left hand side,

$$(10.7) \quad (R_{C1} - R_{C2}) \cdot R_{U12}^2 = (R_{C1} - R_{C2}) \cdot R_{C1} \cdot R_{C2}$$

which now gives us the satisfyingly simple result

$$(10.8) \quad R_{U12} = \sqrt{R_{C1} \cdot R_{C2}} .$$

10.5.3.2 Building your table of threshold resistances for your microcontroller code

We are now prepared to make a table of values of R_U which we can threshold with the measured value to determine when to switch ranges. An example is Table 5, but you must use the measured values of your calibration resistances that you recorded as directed in 4.4.1, not the nominal values from Table 2 that we copied into our example here. The first column is the index of the calibration resistance value. In the second column, use your measured resistances of the calibration resistors. The third column is the demarcation resistance from (10.8). So, if a measured resistance is less than 937 Ω then we know that a calibration resistance of 500 Ω will give us a more accurate measurement than a calibration resistance of 1000 Ω . This table can be recomputed very easily with a spreadsheet program. **You will use the last two columns of your recomputed table in your microcontroller software.**

Table 5. Table of Nominal Resistances for Autoranging Thresholds.

N	R_C , Ohms	Crossover, Ohms	R_{Low} , Ohms	R_{High} , Ohms
0	500			
1	1000	937	893	984
2	2000	1646	1567	1728
3	5000	3404	3242	3574
4	10000	7304	6956	7669
5	20000	14375	13691	15094
6	50000	31866	30349	33459
7	100000	70944	67566	74491

When a measured resistance is very close to a demarcation value, we can avoid endlessly switching ranges by allowing a band of about 10% where the range is not switched. The fourth column is found by dividing the crossover resistances by 1.05 and the fifth column

is found by multiplying the crossover resistances by 1.05. Our logic, as implemented, will be to require that a measured resistance equal or exceed 984 Ω before the calibration resistance is switched from 500 Ω to 1000 Ω , and the measured resistance must be less than or equal to 893 Ω to have the calibration resistance switched from 1000 Ω to 500 Ω .

10.6 The Medical Alert mode

The Medical Alert mode is used when a BAC of 0.16% or higher is sensed, indicating that a serious case of alcohol poisoning is imminent or in progress. In this mode, all three BAC indicator LEDs blink once every two seconds until both these conditions are met:

- The sensor is clear of atmosphere that is indicative of 0.16% or higher BAC and
- The reset button is pushed.

In order to permit calibration when a new has higher than average sensitivity, the Calibrate button will override the Medical Alert mode.

11 Appendix D: Microcontroller Program Listing

```

////////////////////////////////////
////                               breath_analyzer.c                               ////
////                               ////
//// Copyright 2007 by James K Beard and Rowan University.                       ////
//// All rights reserved.                                                       ////
////                               ////
//// This program is the microcontroller code for the breath                    ////
//// alcohol analyzer designed as a Rowan laboratory by Dr. James               ////
//// K Beard. The concept of a breath analyzer for an ECE                       ////
//// laboratory was originally suggested by Dr. Robi Polikar.                   ////
//// See lab protocol by Dr. Beard for software requirements.                   ////
////                               ////
//// I/O Pins:                                                                    ////
//// Pin  No.  Function                                                            ////
//// A0   2    INPUT  Top of voltage divider for sensor circuit                 ////
//// A1   3    INPUT  Sensor voltage                                             ////
////                               ////
//// B0-B7 21-28 OUTPUT Control signals for CD4066B CMOS switches              ////
////                               ////
//// C0   11   INPUT  Calibrate push button                                     ////
//// C1   12   INPUT  Reset push button                                         ////
////                               ////
//// C4   15   OUTPUT LED signal, Ready                                         ////
//// C5   16   OUTPUT LED signal, Warning                                       ////
//// C6   17   OUTPUT LED signal, No Drive Alert                               ////
//// C7   18   OUTPUT LED signal, Red Alert                                     ////
////                               ////
//// Reference: "A Simple Breath Alcohol Indicator," by Dr. Beard               ////
////                               ////
//// May 30, 2007: First draft of code.                                         ////
//// May 31, 2007: Put Ready light into BAC "bar chart"                         ////
////                               Simplify bar chart code, eliminate function call ////
////                               Latch to highest BAC detected, until Reset     ////
////                               Added an open sensor check to avoid div by 0   ////
////                               Used EEPROM to save calibration flag and data   ////
////////////////////////////////////

#include <16F876A.h> //This must match the processor in use
#define ICD=TRUE //Necessary for ICD use when Compile for ICD not checked
#define fuses HS,NOWDT,NOPROTECT,NOLVP

```

```

#define ADC=10 //10 bits right justified in a 16 bit word
#define delay(clock=1000000) //Put your clock rate here; 10 MHz == 1000000
#include <math.h> //for pow(x,y) function
//#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) //RS-232 not used

//Program testing definitions
//Testing builds compute ADC outputs from testBAC
//Testing builds ignore pushbuttons
//Uncomment the two following lines to test software with testBAC
//#define TESTING //If defined, special test code is used, not
measurements
//#define testBAC 0.078 //Voltage measurements computed from this pseudo-BAC

//I/O CONVENTIONS
//#define ADC_A PIN_A0 //Input of voltage divider, across cal resistance + sensor
//#define ADC_B PIN_A1 //Sensor voltage
#define CAL_BUTTON PIN_C0
#define RESET_BUTTON PIN_C1

#define READY_LED PIN_C4
#define WARNING_LED PIN_C5
#define NO_DRIVE_LED PIN_C6
#define RED_ALERT_LED PIN_C7

//Blood alcohol concentration thresholds
#define BAC_Warning 0.04
#define BAC_NoDriving 0.08
#define BAC_RedAlert 0.12
#define BAC_MedicalAlert 0.16

//PROGRAM CONSTANTS
#define alpha 0.633 //Exponent of sensor sensitivity
#define alphaRecip 1./0.633 //Reciprocal of alpha
#define r_sens_at_cal0 2800. //Default sensor resistance at calibration is 2800 Ohms
#define BAC2C 2600. //Multiply BAC by 2600 to get gas concentration in ppm
#define c_cal BAC2C*BAC_NoDriving //Calibration gas concentration in ppm
#define r_SensorShorted 100. //Sensor resistance less than this is considered shorted
#define r_SensorOpen 500000. //Sensor resistance greater than this is considered open
#define CMOS_ON_R 220. //Replace with measured ON resistance of CD4066B

//CONVERSION CONSTANTS
#define i_to_volts (5.0/1024.)//Convert 10 bit right-justified ADC output to volts
#define BAC2COUNT 25000. //Converts a BAC of 0.04 to 1000

////////////////////////////////////
//Sensor fault mode; you don't get up from this one
void sensor_fault(short SHORTED)
{
while(TRUE)
{
output_bit(READY_LED,1);
(SHORTED) ? delay_ms(100) : delay_ms(200);
output_bit(READY_LED,0);
(SHORTED) ? delay_ms(200) : delay_ms(100);
}
}
////////////////////////////////////
//Main microcontroller program
void main()
{
int irange,i; //Autoranging index, loop index

//The table of Ohmmeter calibration resistors here is nominal -- use measured values; see
4.4.1
float const rcal[8]={500., 1000., 2000., 5000., 10000., 20000., 50000., 100000.};

//The table of resistance measurement limits is nominal; see 10.5.3 and recompute Table 4
float const rlow[8]={0., 893., 1567., 3242., 6956., 13691., 30349., 67566.};
float const rhigh[8]={984., 1728., 3574., 7669., 15094., 33459., 74491., 1.e30};
float r_sens_at_cal; //Sensor resistance at calibration
float v_A, v_B; //Voltage divider voltages

```

```

float r_meas;                //The measured sensor resistance

//The bit "calibrated" must be set to zero by clearing memory before programming
int calibrated;              //FALSE -- flag; set to TRUE once calibration occurs

float concentration;        //Measured alcohol vapor concentration in ppm
float BACm;                 //Measured BAC
float blood_alcohol=0.;     //Highest BAC since last reset
long int blink_ms;         //Blinking LED on time, milliseconds

//Set up the I/O ports
setup_adc_ports( ALL_ANALOG ); //Inputs are A0 A1 A2 A3 A5, ref is 5 V
SET_TRIS_B(0b00000000); //All B bus pins are outputs
SET_TRIS_C(0b00000011); //C0 and C1 are inputs; all others are outputs or unused

setup_adc( ADC_CLOCK_INTERNAL ); //Internal clock, or box crystal if there

output_B(0b10000000);      //Set the B bus pins for highest cal resistance

irange=7;                  //Index of active voltage divider

output_bit(READY_LED,0);   //Turn off all the LEDs
output_bit(WARNING_LED,0);
output_bit(NO_DRIVE_LED,0);
output_bit(RED_ALERT_LED,0);

//Read the initialization flag and calibration resistance from EEPROM
calibrated=~read_eeprom(0); //Erased EEPROM is all 0xFF
for (i=0; i<4; i++)
{
    *(&r_sens_at_cal+i) = read_eeprom(i+1);
}

//Now we are in the initialization mode.
// Blink the ready light once per second for 120 seconds.

#ifdef TESTING              //Skip the 2-minute wait for software testing
for(i=0;i<120;i++)        //This is the sensor warm-up delay
{
    output_bit(READY_LED,1); //Turn on the Ready light for 1/2 second
    delay_ms(500);
    output_bit(READY_LED,0);
    delay_ms(500);
}
#endif

output_bit(READY_LED,1);   //Turn on the Ready light before the first measurement

//END OF INITIALIZATION MODE

//ACTION LOOP; NORMAL OPERATION AND MEDICAL ALERT MODES
while (TRUE)
{
    //////////////////////////////////////
    //Compute sensor resistance, with autoranging
    while (TRUE)           //Autoranging loop
    {

#ifdef TESTING            //Use conditional compilation to insert test BAC
        r_meas=r_sens_at_cal0*pwr(c_cal/(testBAC*BAC2C),alpha);
        v_A=5.*(rcal[irange]+r_meas)/(CMOS_ON_R+rcal[irange]+r_meas);
        v_B=5.*r_meas/(CMOS_ON_R+rcal[irange]+r_meas);
#else
        //Read voltage divider input voltage
        set_adc_channel(0); //Voltage divider is port A0
        delay_us(10);      //A small delay is required before an ADC read
        v_A=i_to_volts*((float) Read_ADC());

        //Read sensor voltage
        set_adc_channel(1); //Sensor voltage is port A1

```

```

    delay_us(10);                //A small delay is required before a read
    v_B=i_to_volts*((float) Read_ADC());
#endif

//Check for v_B==V_A and declare a sensor fault if so
if(v_B == v_A) sensor_fault(FALSE);    //Open sensor

    r_meas=(v_B/(v_A-v_B))*rcal[irange];    //Compute sensor resistance IAW (3.1)

//Autoranging logic
if(r_meas <= rlow[irange])            //Measurement low for resistor; decrease
{
    if(irange <= 0) sensor_fault(TRUE);    //Underrange ==> shorted sensor!!!

    output_bit(PIN_B0+irange,0);        //Change the switching on the B bus
    irange--;
    output_bit(PIN_B0+irange,1);
    continue;
}
else if(r_meas >= rhigh[irange])        //Measurement high for resistor; increase
{
    if(irange >= 7) sensor_fault(FALSE);    //Overrange ==> open sensor!!!

    output_bit(PIN_B0+irange,0);        //Change the switching on the B bus
    irange++;
    output_bit(PIN_B0+irange,1);
    continue;
}
else                                    //Measurement OK, so exit autoranging loop
    break;
}
//End of resistance measurement with autoranging
////////////////////////////////////

//Check for shorted or open sensor
if(r_meas < r_SensorShorted) sensor_fault(TRUE);
if(r_meas > r_SensorOpen) sensor_fault(FALSE);

//Check for calibration button
//Pull-up, pushed==0, so invert the logic here by adding the NOT operator "!"
#ifndef TESTING
    if(!input_state(CAL_BUTTON))        //Calibrate?
#else
    if(FALSE)
#endif
{
    calibrated=TRUE;
    r_sens_at_cal=r_meas;

//Write calibration data to EEPROM
write_eeprom(0,~calibrated);
for (i=0; i<4; i++)
{
    write_eeprom(i+1,&r_sens_at_cal+i);
}

    output_bit(READY_LED,0);    //Turn off Ready LED for 1 second to ack calibration
    delay_ms(1000);
    output_bit(READY_LED,1);
    continue;                    //Now go back and start over
}

//Compute alcohol vapor concentration IAW equation (4.2)
concentration = (calibrated) ?        //Conditional expression
    c_cal*pow(r_sens_at_cal/r_meas,alphaRecip) :    //Calibrated
    c_cal*pow(r_sens_at_cal0/r_meas,alphaRecip);    //Not calibrated

BACm=concentration/BAC2C;                //Measured BAC in percent IAW 2.2

```

```

    if(BACm > blood_alcohol) blood_alcohol=BACm;          //Take the largest BAC seen

#ifndef TESTING                                          //Omit button checks when testing
    if(!input_state(RESET_BUTTON))
    {
        blood_alcohol=BACm;                             //Reset the blood alcohol to what is seen
    now
    }
#endif

//Implement the normal operation rows of the Turing table, Table 3 in 10.3

//Implement the Medical Alert mode -- all on 1 second, all off 1 second
if(blood_alcohol >= BAC_MedicalAlert)                //Highest priority is Medical Alert
{
    output_bit(Ready_LED,1);
    output_bit(WARNING_LED,1);
    output_bit(NO_DRIVE_LED,1);
    output_bit(RED_ALERT_LED,1);
    delay_ms(1000);
    output_bit(Ready_LED,0);
    output_bit(WARNING_LED,0);
    output_bit(NO_DRIVE_LED,0);
    output_bit(RED_ALERT_LED,0);
    delay_ms(1000);
}
else if(blood_alcohol < BAC_Warning)                  //BAC below BAC_Warning?
{
    //Display:  -*- 0 0 0
    blink_ms=(long)((blood_alcohol)*BAC2COUNT);      //LED ON time, ms

    output_bit(Ready_LED,1);
    output_bit(WARNING_LED,0);
    output_bit(NO_DRIVE_LED,0);
    output_bit(RED_ALERT_LED,0);

    delay_ms(blink_ms);                               //Leave the blinking one on
    output_bit(Ready_LED,0); //Turn it off
    delay_ms(1000-blink_ms); //Leave it off until the 1 second is over
}
else if(blood_alcohol < BAC_NoDriving)                //BAC above BAC_Warning, below
BAC_NoDriving?
{
    //Display:  # -*- 0 0
    blink_ms=(long)((blood_alcohol-BAC_Warning)*BAC2COUNT); //LED ON time, ms

    output_bit(Ready_LED,1);
    output_bit(WARNING_LED,1);
    output_bit(NO_DRIVE_LED,0);
    output_bit(RED_ALERT_LED,0);

    delay_ms(blink_ms);                               //Leave the blinking one on
    output_bit(WARNING_LED,0); //Turn it off
    delay_ms(1000-blink_ms); //Leave it off until the 1 second is over
}
else if(blood_alcohol < BAC_RedAlert)                //BAC above BAC_NoDriving?
{
    //Display:  # # -*- 0
    blink_ms=(long)((blood_alcohol-BAC_NoDriving)*BAC2COUNT); //LED ON time, ms

    output_bit(Ready_LED,1);
    output_bit(WARNING_LED,1);
    output_bit(NO_DRIVE_LED,1);
    output_bit(RED_ALERT_LED,0);

    delay_ms(blink_ms);                               //Leave the blinking one on
    output_bit(NO_DRIVE_LED,0); //Turn it off
    delay_ms(1000-blink_ms); //Leave it off until the 1 second is over
}
else
{
    //Display:  # # # -*-
    blink_ms=(long)((blood_alcohol-BAC_RedAlert)*BAC2COUNT); //LED ON time, ms

    output_bit(Ready_LED,1);

```

```
output_bit(WARNING_LED,1);
output_bit(NO_DRIVE_LED,1);
output_bit(RED_ALERT_LED,1);

delay_ms(blink_ms);           //Leave the blinking one on
output_bit(Red_Alert_LED,0);  //Turn it off
delay_ms(1000-blink_ms);     //Leave it off until the 1 second is over
}

}

//Repeat action loop forever
//End of program
```

References

-
- ¹ <http://www.alcoholalert.com/drunk-driving-statistics.html>
 - ² <http://pubs.niaaa.nih.gov/publications/arh27-1/63-78.htm>
 - ³ <http://www.breathalyzer.net>
 - ⁴ University of Oklahoma Police Department, published on web page <http://www.ou.edu/oupd/bac.htm>
 - ⁵ *Application Notes For Breath Alcohol Testers Using the Figaro TGS 2620*, Version 1.0 Revision 11 March, 2003.
 - ⁶ Texas Instruments Data Sheet for the CD4066B, Document Reference SCHS051D – November 1998 – Revised September 2003.
 - ⁷ Figaro Sensors, web site <http://www.figarosensor.com/> .
 - ⁸ Figaro Product Information from TGS 2620 data sheet dated October 2000.
 - ⁹ *Technical Information on Usage of TGS Sensors for Toxic and Explosive Gas Leak Detectors*, Figaro Engineering, Inc. document dated November 2004.
 - ¹⁰ *PIC MCU C Compiler Reference Manual*, Version 4 (2007), hard copy provided with CCS PIC-C for the PIC microcontrollers with 14-bit instruction word such as the 16F876A, page 195. PDF version available at http://www.ccsinfo.com/downloads/ccs_c_manual.pdf
 - ¹¹ *PIC16F87XA Data Sheet*, Microchip Technology, Inc. document DS39582B, available from <http://www.microchip.com> .
 - ¹² Header file for the PIC 16F876A, Custom Computer Services, Inc., included with their PCM C compiler for PIC microcontrollers that have a 14 bit instruction size. See the CCS web site <http://www.ccsinfo.com> .