BREATH COLLECTION EQUIPMENT FOR CLINICAL APPLICATIONS WITH SIFT-MS INSTRUMENTS

A thesis submitted in partial fulfilment of the requirements for the

Degree

of Master of Mechanical Engineering

in the University of Canterbury

by Ketan Lad

University of Canterbury

2006

Abstract

Real time detection of Volatile Organic Compounds (VOCs) using Selected Ion Flow Tube – Mass Spectrometry (SIFT-MS) provides a unique opportunity for research into breath testing for clinical diagnosis. However, before engaging in research into breath analytes as markers of disease, appropriate breath collection methods are required. Collection of breath for SIFT-MS instruments fall into two categories, direct breath collection into the instrument and the remote breath collection onto a storage medium. This thesis describes the development and validation of both methods of breath collection equipment for SIFT-MS analysis.

Development of the direct breath collection device involved standardising and optimising the way in which breath is sampled by SIFT-MS. Design considerations include ergonomics, patient safety, breathing resistance, materials, and appropriate operating conditions of the device. Results from materials testing showed that all materials emit VOCs and the best approach is to minimise VOC emission by careful material selection. To minimise flow resistance experienced by the patient, the capillary from which the SIFT-MS instrument samples, is placed as close as possible to the users mouth. The optimal operating temperature of the device was found to be 100°C - 120°C, which ensures that water vapour will not condense inside the capillary causing blockage. In order to ensure patient safety the device is adequately insulated using stagnant air which also minimises VOC emission from insulation materials.

Because a SIFT-MS instrument is large and cannot be easily shifted around a hospital, a system of remote sample collection is required. It is also important to separately

collect and analyse breath from the respiratory alveolar region. For this reason the remote breath collection device designed also fractionates collected breath samples into the breath from the upper airways and alveolar breath. The storage medium chosen for the collected breath samples is a gas sampling bag made from Tedlar™. Collection of breath into Tedlar™ bags allows breath to be stored as a whole air sample, the ideal form for analysis with the SIFT-MS technique.

Alveolar breath is fractionated from deadspace gasses by measuring a subject's exhalation and collecting the portion of interest. The breath exhalation is measured by an averaging Pitot tube and pressure transducer. Signal processing and automation of the remote breath collection device is controlled by a Cypress Microsystems PSoC microcontroller. To validate the device isoprene and acetone concentrations in fractionated breath samples were compared with a whole breath sample. Results showed that the alveolar breath fraction had a higher concentration of acetone than the upper airway fraction, indicating that the breath was successfully fractioned. However, isoprene concentrations were lower in both fractions due to hyperventilation of the subject causing a dilution effect of alveolar VOCs. Therefore, a higher sample collection volume is required per exhalation, and regulating subjects' breathing rate will avoid the dilution effect observed in collected breath samples.

Overall, this thesis had designed, developed and validated two forms of breath collection systems for use with SIFT-MS technology.

Acknowledgements

Firstly like to thank my university supervisor, Associate Professor Dr Geoff Chase who has provided me with the opportunity to work on this masters project along with his guidance and support. His ability to review multiple thesis chapters overnight still amazes me.

I would also like to thank Dr Randall Allardyce and Dr Jenny Scotter, my project mentor and manager at Syft Technologies Ltd for all their advice and support throughout the project.

As well as Randall and Jenny I would also like to thank everyone at Syft Technologies for accommodating me in the company. In particular I would like to thank Dr Justin Stevenson, Senti Sentilmohan, Dr Paul Wilson, Katy Ledingham, and Professor Murray McEwan for all their advice and support.

I would like to thank my project partner James Neilson, whom I have worked closely with on this master's project. I have gained many personal attributes from working with James, not only through this project, but in my undergrad years as well.

A big thanks to Rodney Elliot from the Applied Mechanics laboratory in the Mechanical Engineering lab wing. Thanks for putting up with my constant questions and making three revisions for the circuit board for the remote breath collection device. I know you're going to miss having someone asking you questions all day long but I'm sure someone will take my place.

Also, I would like to thank Graeme Harris from the Fluid Mechanics laboratory in the Mechanical Engineering lab wing for use of lots of equipment, advice and help making the averaging Pitot for the remote breath collection device.

Finally I would like to thank Catherine for her support in my times of stress.

Contents

Figures

Tables

1

Olfaction as an aid to disease diagnosis

For many years, physicians have known that the sense of smell can provide information about a person's physiological state. However, using one's sense of smell is an inaccurate method of diagnosis as this sense will vary from physician to physician, as well as with age and other factors. Hence, most current methods of diagnosis involve analysis of blood, urine and/or other bodily fluids.

Commonly, physicians analyse blood samples along with descriptions of symptoms to diagnose for illnesses. More specifically, blood resides within the body and is continually flowing throughout it, acting as a medium for providing oxygen and nutrients, and receiving waste products (i.e. carbon dioxide (CO_2)) to expel from the body.

Similarly, certain disease states upset chemical balance within the body, also resulting in measurable changes to blood chemistry. Gas exchange between body tissue and blood cells occurs by chemical reactions between red blood cells and the tissue walls. Figure 1.1 shows gas exchange from body tissue to blood cells.

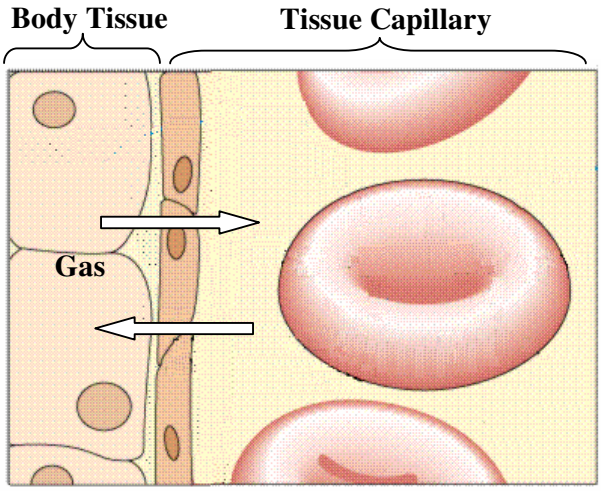


Figure 1.1 - Gas Exchange between body tissue and blood (Purves, 2004)

During the period in the lungs where blood uptakes oxygen, it exchanges waste gasses. Figure 1.2 shows the exchange of gasses between blood and the alveolar tissue membrane. These gasses are then exhaled out of the body. The waste gasses exhaled also provide significant information about the person's physiological state. Therefore, analysis of these gasses could be an ideal non-invasive method of disease diagnosis if compounds that are associated with disease states are detectable.

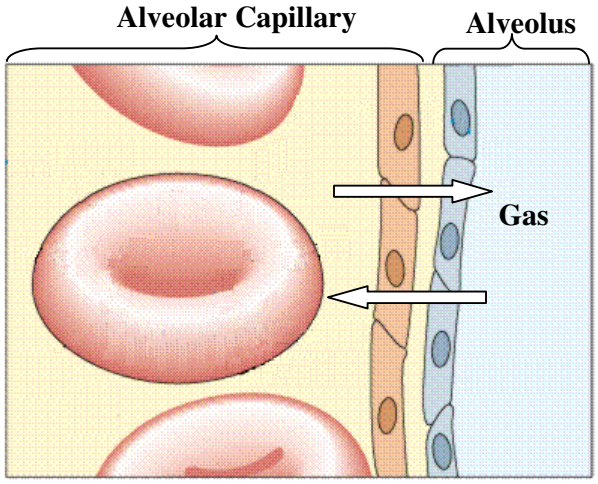


Figure 1.2 - Gas exchange between blood and inspired air (Purves, 2004)

1.1 Human Lung Physiology

The human respiratory system can be broadly divided into two sections, the upper and lower airways. The upper airways consist of the nasal and oral cavities, trachea, and the necessary plumbing, the bronchi, required getting air to the lung tissues (the lower airway), Figure 1.3 shows this basic physiology schematically. It is in the lung tissues of the lower airway that gas exchange takes place. As discussed previously, the primary function of the lungs is rapid gas exchange between circulatory blood and inhaled air.

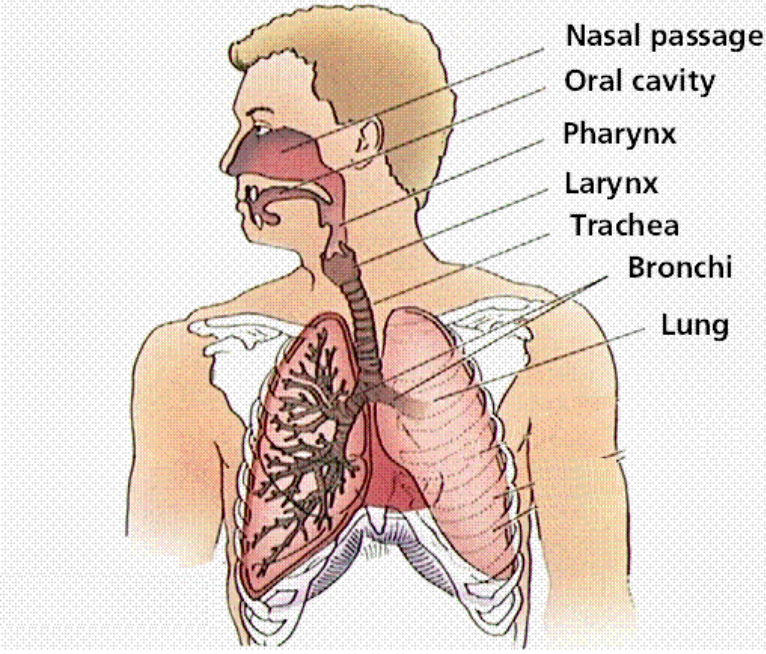


Figure 1.3 - Human Respiratory System (Purves, 2004)

The human lung tissues can be broken down into smaller, individual components. After many airway branches, inhaled air reaches the lower lung tissue: the bronchioles and the alveoli. The alveolus consists of alveolar ducts, sacs, acini and individual alveoli. Gas exchange occurs in the alveolus where a laminar flow of blood and

inhaled air are separated by a thin tissue layer. Blood flow occurs in a network of capillaries that encase each alveolus. The source of this blood flow is the pulmonary artery and the arteriole network extending from it to the lung tissues. This structure is shown schematically in Figure 1.4.

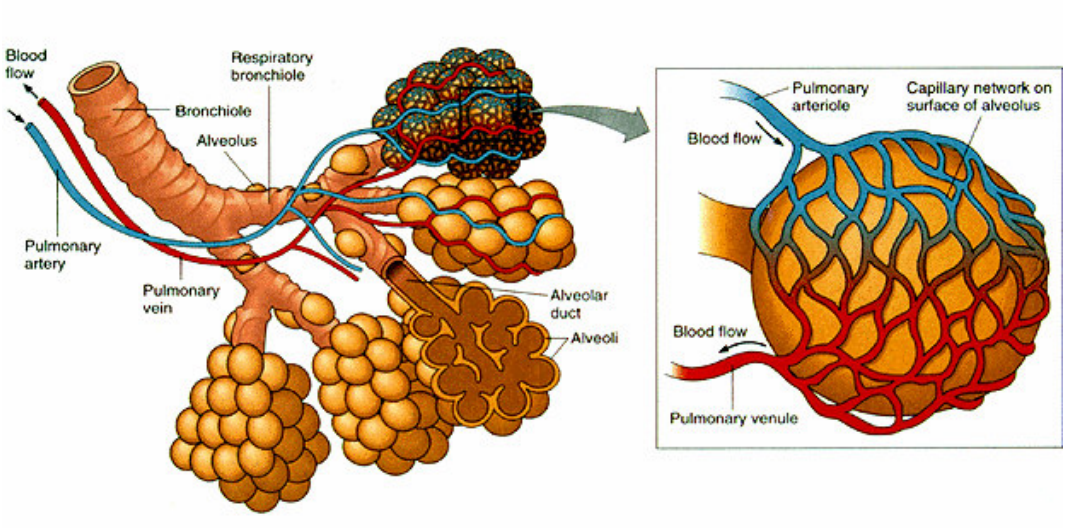


Figure 1.4 - Respiratory Tubes and Alveoli (Tamarkin, 2006)

The human body requires its entire blood volume (approximately 5 litres) to be re-oxygenated approximately every minute. Therefore, the gas exchange process needs to be fast, and the lungs must have many alveoli to supply a high surface area for gas exchange. To meet this demand humans have around 300 million alveoli. Because there is a high turnover of blood through the lungs, humans exhale breath containing these waste gas molecules and compounds into the atmosphere. The portion of exhaled air, containing gasses that have exchanged with blood, is referred to as alveolar breath. Exhaled compounds within the alveolar breath have been the focus of respiratory researchers for many years in trying to detect and diagnose respiratory, and other diseases (Hamilton, 1998; Baumbach *et al.*, 2005).

1.2 Compounds Commonly Found in Breath

Numerous compounds are found in the breath of healthy individuals at parts per billion (ppb) or parts per million (ppm) levels. These compounds are created through metabolic processes including energy production, cholesterol synthesis, glucose metabolism and lipid peroxidation of fatty acids (Phillips, 1999). Table 1.1 lists some common compounds found in breath and their reported concentration ranges from various studies (Phillips *et al.*, 1994; Risby, 2005).

Compound & Levels	Concentration Range	Metabolic Processes
Ammonia	ppb - ppm	Digestion
Acetone	ppb - ppm	Glucose Metabolism
Isoprene	ppb - ppm	Cholesterol Synthesis
Ethanol	ppb	Gut Bacteria
Pentane	ppb	Lipid Peroxidation

Table 1.1 - Typical concentrations of breath analytes

Sources of compounds found in breath

Compounds like ammonia, isoprene and acetone are associated with metabolism and occur in varying concentrations throughout the day. For example, studies have shown that acetone levels in breath are significantly higher during fasting (Smith & Spanel, 2005). Similarly, high protein meals, such as red meat, increase ammonia concentrations after digestion (Smith & Spanel, 2005). Figure 1.5 shows possible sources some of compounds found in alveolar breath.

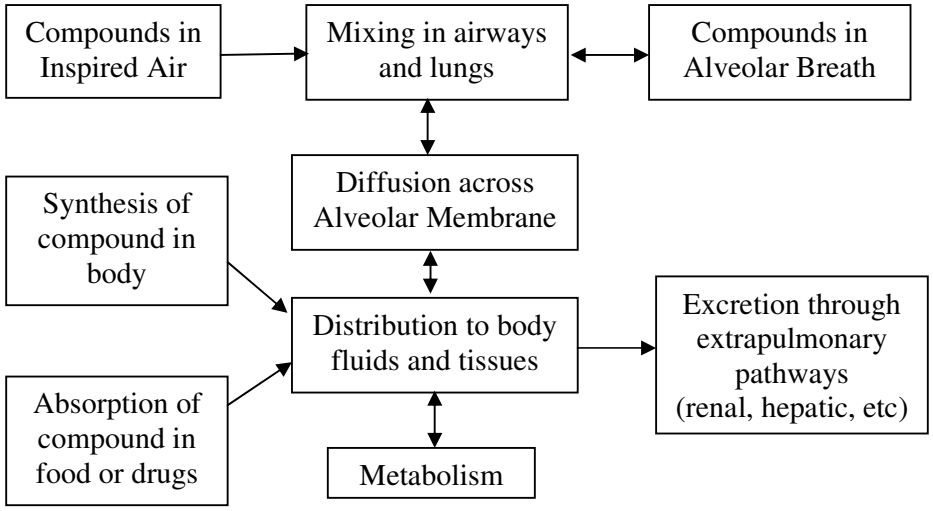


Figure 1.5 - Sources of Compounds found in Alveolar Breath (Phillips *et al*., 1994)

These commonly occurring compounds found in breath, or changes in their levels could potentially be used to diagnose alterations in metabolic processes due to illness or other factors. For example, Pentane is the product of oxygen free radical (OFR) meditated lipid peroxidation of n-6 polyunsaturated fatty acids, and has potential for the diagnosis of disorders such as breast cancer, heart transplant rejection, acute myocardial infarction, schizophrenia and rheumatoid arthritis (Phillips, 1999).

1.3 Breath Sampling

Sample delivery

Before collecting breath samples, an understanding of respiratory physics and physiology is required. Breathing normally, in a state of rest, air inspired and expired in one breath, is typically around 0.5 litres (500 ml), commonly referred to as the tidal volume. However, the total lung capacity is approximately 6 litres. If a person were to

take a deep breath and exhale as much as they could, their total exhalation would be approximately 4.8 litres, a value called the vital capacity. The remaining 1.2 litres, which remains in the lungs, is called the residual volume. Figure 1.6 provides a more complete description of the physics of breathing.

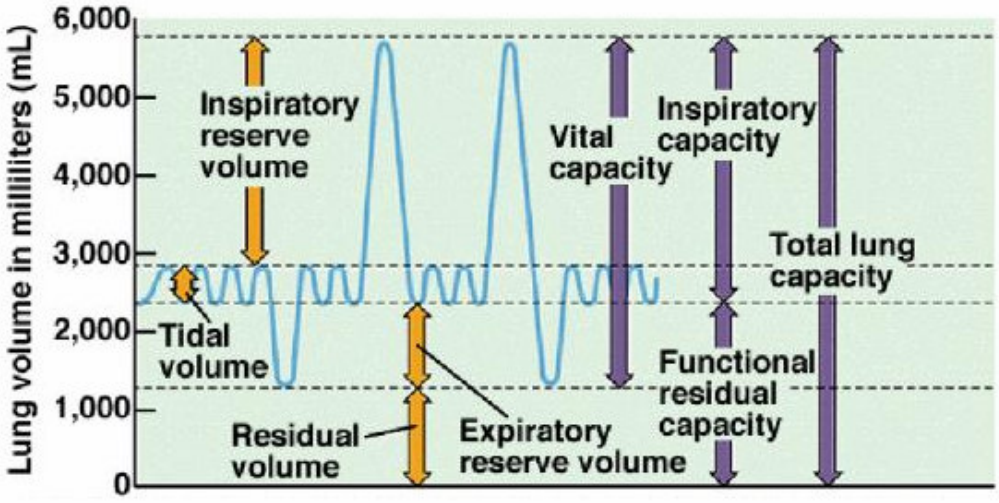


Figure 1.6 - Respiratory Volumes and Capacities (Tamarkin, 2006)

Because gas exchange occurs in the alveolus, exhaled breath that originates from the upper airways (dead-space) should be excluded from analysis. More specifically, excluding the dead-space from the analysis excludes chemicals from the analysis that originate from the atmosphere rather than from gas exchange with blood cells. Typically, the dead-space volume of an average human is 0.15 litres (Tamarkin, 2006). This value is 30% of a tidal volume or approximately 3.125% of a typical vital capacity.

To date, little research has examined the effect of breath delivery method on breath compound concentrations. Variations in delivery method may include collection of breath tidal volumes, vital capacities or set breath flow-rates. Additionally, the

physical apparatus used to collect or analyse breath may vary in length, operating temperature, flow control/restriction and other design parameters. All of these parameters may affect the concentrations and/or interactions of compounds in the sample, thus affecting the resulting "true" value available to be measured by a given analysis system. For example, nitric oxide (NO) is a marker of asthma (Dweik, 2005; Gustafsson, 2005) and the commercially available instrument NIOX (NIOX, 2004) is used as a tool solely for NO detection. However, patients providing a sample on the NIOX instrument need to provide breath at a set flow-rate, back pressure and resistance during exhalation. The developers of the NIOX instrument experimentally defined an exhalation flow-rate that would cause the soft palate in the back of the pharynx to lift, allowing the analysis of NO that originates only from the lungs and not the portion that resides in the nasal cavity (recommended guidelines for standardised testing of exhaled NO (Silkoff, 1999)). Similarly, variations in sample delivery may cause noticeable effects with other compounds, for example highly water-soluble molecules like ammonia may vary in measured concentration depending on the humidity of the sample.

Chemical Analysis of Breath – Mass Spectrometry

Measurement of compounds found in breath has previously been difficult to achieve with sufficient sensitivity and has also been time consuming. Olfaction is one of the last senses to be converted into an electronic/measurable form, but technology is now such that it can be put to practical use. Mass spectrometry (MS) is an analytical technique used for chemical detection and many variations now exist (School of Chemistry, University of Bristol, 2005).

Mass spectrometry (MS) is based upon the scientific method of measuring the mass-to-charge (m/z) ratio of ions (a molecule with an overall net charge). It produces a mass spectrum of a sample to find its composition, and is typically achieved by ionizing the sample analyte and separating ions of differing masses. Variations of mass spectrometers that have been available since the late 1800s, these are summarised in the following paragraphs.

Gas Chromatography Mass Spectrometry (GC-MS) is a common technique where a gas chromatograph is used to separate compounds. The separated compounds are fed onto an ion source where a voltage is applied to emit electrons to ionise the compounds. The ionised compounds fragment to produce predictable patterns, detected by the mass analyser. A similar technique to GC-MS is Liquid Chromatography Mass Spectrometry (LC-MS) where the sample is in a liquid form, usually a combination of water and organic solvents rather than using a gas sample.

Flowing Afterglow Mass Spectrometry (FA-MS) emerged in the 1960's when flowing afterglow technology and quadrupole mass filters were combined with mass analysers. This approach was the first variation of mass spectrometry that offered real time analysis of humid gas samples including breath as well as the headspace above urine and serum samples (Smith and Spanel, 2004). Operation of the FA-MS involves a weak microwave discharge produced in helium or argon carrier gas, which leads to the creation of the H_3O+ precursor ion. The precursor reacts with sample gasses in a flow tube, and product masses are separated by a quadrupole before analysis.

Selected Ion Flow Tube Mass Spectrometry (SIFT-MS) is a sensitive and quantitative technique that was conceived and developed by Adams and Smith (1976) and was spawned from FA-MS analysers (Smith & Spanel, 2004). Advancements of SIFT-MS from FA-MS involved production of precursor ions outside the flow tube, thus allowing selection of ions injected into the flow tube.

1.4 Preface

This thesis discusses in detail the operation of SIFT-MS and its applications, in particular, the collection and analysis of breath samples for clinical diagnosis using the SIFT-MS technique. Two methods of sampling breath with SIFT-MS are described, firstly the direct sampling into the SIFT-MS instrument and secondly, collecting a sample remotely into storage media that can be analysed at a later stage. The direct device is discussed but is a small portion of this thesis however, several aspects of the direct device are also applicable to the remote collection apparatus. The majority of this thesis discusses the design and manufacture of the remote collection apparatus that is compatible with SIFT-MS for research purposes.

This thesis is outlined as follows:

- Detailed description of the SIFT-MS technique, a comparison with the GC-MS technique and applications of SIFT-MS.
- A review of other breath collection devices previously used for clinical breath testing.
- Design of a direct breath collection device for a SIFT-MS instrument that standardises and optimises breath analysis.

- Identification of requirements for a remote breath collection device attuned with SIFT-MS.
- Hardware design and selection for the remote breath collection device.
- Software architecture and electronic hardware design for remote breath collection device.
- Testing and validation of the remote breath collection device.
- Conclusion and future recommendations for future 2nd generation breath collection devices for SIFT-MS instruments.

2

SIFT-MS Technology

Initially developed for the study of kinetics on gas phase ion-neutral reactions, SIFT-MS now has vast kinetic data and is presently used for the detection of volatile organic compounds (VOCs) (Smith and Spanel, 2004). Advantages of the SIFT-MS approach include the ability to measure VOCs in real time (less than 250 milliseconds (ms)) and sensitivity to low ppb levels. At present, most clinicians involved in breath testing research utilise GC-MS technology because the SIFT-MS technique was previously limited to universities due to its large size. Development of the SIFT-MS technique by Syft Technologies Ltd (Christchurch, New Zealand) has resulted in an instrument that offers prospects for research and commercial applications.

2.1 A Comparison of SIFT-MS with GC-MS

Previously, methods of analysing breath have largely been limited to GC-MS, making it a gold standard of sorts. With the introduction of SIFT-MS, a new opportunity has emerged that allows easier and faster analysis of breath (Syft, 2004c). SIFT-MS and GC-MS are currently the most common forms of mass spectrometry. However, GC-MS instrumentation has been commercially available for some time, while SIFT-MS instrumentation is only just emerging into the industry.

Perhaps the most significant advantage of SIFT-MS is the sub-second time it takes to scan a gas sample, versus the 10-30 minutes required for GC-MS. SIFT-MS samples also require little preparation (which could potentially compromise instrument sensitivity), as the process requires no chromatographic separation. A major component in the time taken for a GC-MS scan is the need for chromatographic separation. Separation of compounds requires an appropriate choice of column for scans with GC-MS and is better suited to analysing samples that are thermally desorbed. Whole air samples can be analysed using GC-MS, but the process is complicated and requires multiple separation columns resulting in a longer scan time than a thermally desorbed sample.

In contrast to GC-MS, SIFT-MS is designed to analyse whole air samples without the need for chromatographic separation, which in-turn significantly reduces a sample scan time. Along with the ability to sample whole gas samples, SIFT-MS instrumentation has inlet ports for thermal desorption. Thermal desorption allows the concentration of samples, allowing SIFT-MS to detect compounds that occur in parts per trillion (ppt) concentration in individual breaths, when collected over several breaths. This aspect makes the sensitivity of SIFT-MS more comparable with that of GC-MS.

Operating costs for both instruments are similar. GC-MS and SIFT-MS have maintenance costs of approximately 5% of the instrument cost per annum. GC-MS requires helium as a carrier gas. However, SIFT-MS requires argon as well as helium as a carrier gas. Hence, the overall running costs are similar.

Finally, GC-MS can detect compounds down to ppb levels (and ppt levels for some molecules). However, it is not a practical technique for the detection of smaller molecules, such as ammonia and formaldehyde (Smith & Spanel, 2005). SIFT-MS can detect these smaller molecules to ppb levels. This last aspect is a major advantage, as these molecules occur in breath and may have clinical diagnostic use.

2.2 The Science Behind SIFT-MS

As discussed previously, mass spectrometers measure the m/z ratio of ions. Therefore, sample gasses analysed using SIFT-MS are ionised using precursor ions (precursors). Precursors are used to ionise sample molecules, essentially initiating chemical reactions that produce ions monitored by the mass spectrometer. Put simply, the net charge on the precursor is transferred to the product ions. Therefore, by monitoring the remaining precursor ions, product ions and their ratios, the original molecule that entered the SIFT-MS instrument and its concentration can be determined. The SIFT-MS instrument used in the experiments in this thesis, uses three precursor ions (H_3O+, NO+ and O_2+) to ionise VOCs in a sample of gas. The precursor ions used in SIFT-MS are chosen for two reasons. Firstly is their ability to react with VOCs in sample gasses to form products in the detectable range of the device, which is 10 to 240 atomic mass units (amu). The second reason H_3O+, NO+ and O_2+ are used in SIFT-MS is because these precursors do not react with the major constituents of air, N_2, O_2, H_2 and CO_2.

Essentially, the SIFT-MS instrument can be divided into three sections (five key components) as shown by the schematic in Figure 2.1. These three sections are:

- the upstream chamber
- the flow tube
- the downstream chamber

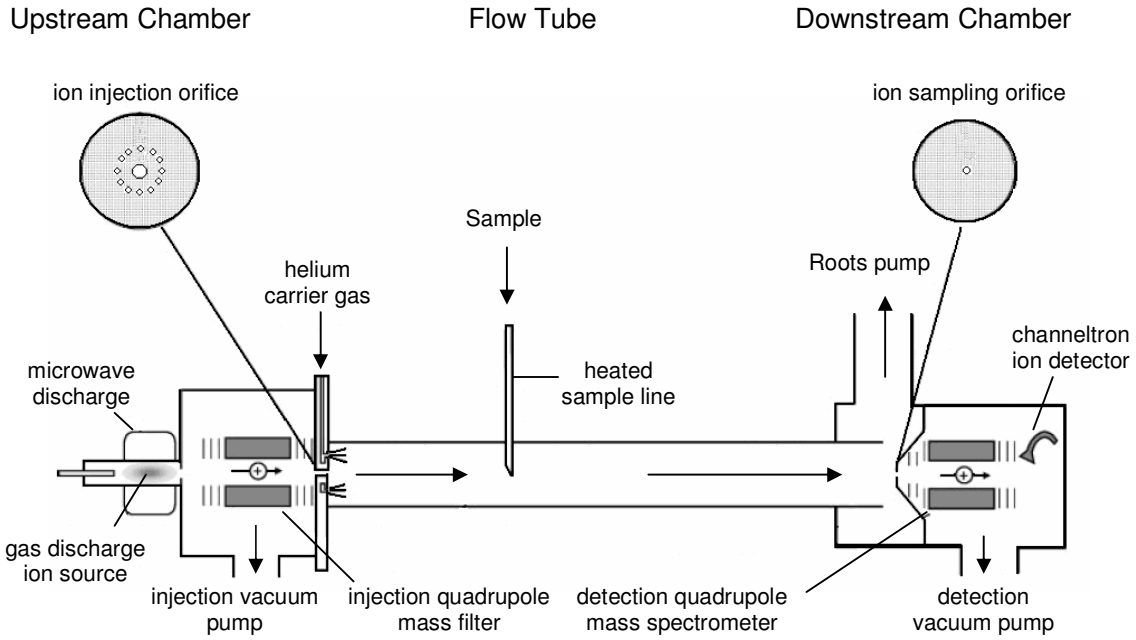


Figure 2.1 - SIFT-MS Schematic (Smith & Spanel, 2004)

Upstream Chamber:

The upstream chamber consists of a microwave resonator and a quadrupole mass filter. The microwave resonator produces the precursors, H_3O+, NO+ and O_2+ by ionising either water vapour or air. These ions are required for ionisation of the sample gasses injected into SIFT-MS.

After the production of precursor ions in the microwave resonator, they travel into a quadrupole mass filter that selects a precursor to react with sample gasses in the flow

tube. A quadrupole mass filter works by guiding ions of a particular mass-to-charge (m/z) ratio towards a destination. In the case of the upstream quadrupole on SIFT-MS instruments, this destination is the Venturi orifice leading into the flow tube. The quadrupole consists of four parallel metal rods, where opposing rods are electrically connected as shown in Figure 2.2. Radio frequency voltages are applied to each pair of rods for the selection of ions with particular m/z ratios. Other ions that travel down the quadrupole will oscillate in an unstable manner, collide into the quadrupole rods, and fail to proceed to the target destination.

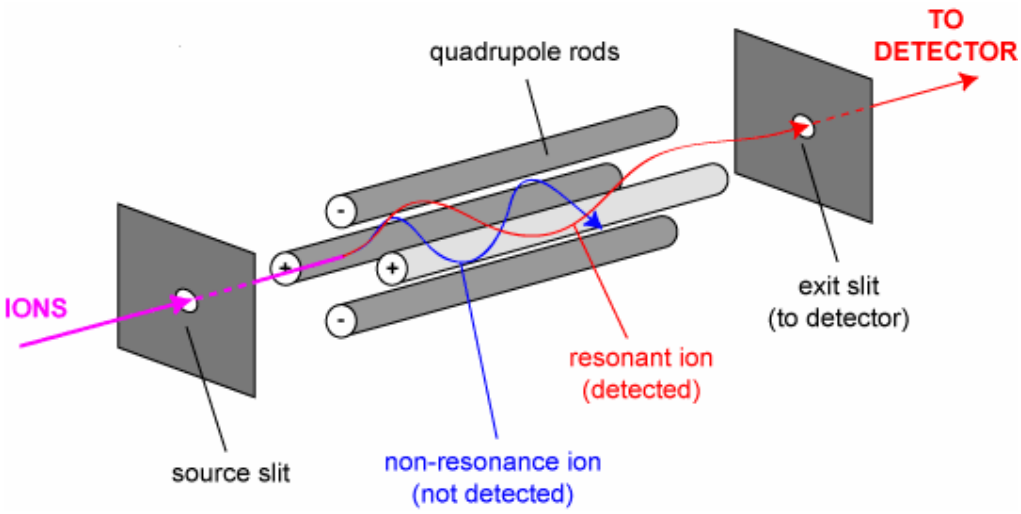


Figure 2.2 - Quadrupole Mass Filter Schematic (School of Chemistry, 2005)

Mass analysers operate on the basis of the dynamics of charged particles in electric and magnetic fields in a vacuum. All mass analysers work by the following fundamental physics equation:

$$(m/q)a = E + v \times B \qquad (2.1)$$

Where:

m	=	Mass of ion
a	=	Ion acceleration
q	=	Ionic charge
E	=	Electric field
v x B	=	Vector cross product of ion velocity of ion and magnetic field.

Equation (2.1) comes from the Lorentz force law and Newton's second law of motion. The two are equated using Newton's third law of motion.

Flow Tube:

After selection of the precursor ions and injection into the flow-tube via the venturi orifice, ions travel into a stream of helium. The helium thermalises the ions and acts as a carrier gas, referred to as the primary carrier gas in SIFT-MS scans. Further downstream, the sample gas is injected into the flow tube and a second carrier gas, argon, is added to improve diffusion properties of the ion stream. Once the sample gasses are injected into the flow tube they react with precursor to form product ions.

To calculate the concentration of the sample gasses, SIFT-MS requires the flow-rate of gas entering the flow tube. Therefore, a capillary tube is used as the sample inlet. A capillary is a fine length of tube approximately 1.5 mm in diameter. The pressure differential across the capillary tube is constant and therefore provides a constant flow-rate of sample into the SIFT-MS flow tube. The length of the capillary controls the sample flow rate into the SIFT-MS flow tube, where a longer tube will give a

lower sample flow rate. Each capillary needs to be calibrated before use as the length and internal finish of the tube will affect the flow-rate through it.

Downstream Chamber:

Product ions travel into the downstream chamber from the flow tube where a second quadrupole selects product masses to pass through to the mass analyser. The detector records the charge induced when an ion travels past it, and then returns a raw voltage, which is interpreted as a 'counts per second' (cps) reading. The cps reading is then received by the software running the SIFT-MS instrument, which calculates the concentration of the compound based on this reading in units of ppb.

2.3 Operation and Scans

When performing scans on SIFT-MS instruments there are several settings that must be adjusted, such as the settle interval, precursor interval, product interval and sample settle time. These settings affect the accuracy of SIFT-MS and are dependent on the number of sample molecules monitored. Adjustment of the precursor and product intervals control how fast the upstream and downstream quadrupoles select precursor ions and product masses respectively. Typically, these values are in 10's of milliseconds. Choosing the times for product and precursor intervals is dependent on how many compounds are analysed in a sample. The larger the number of compounds monitored, the longer the intervals need to be to ensure they are all captured. Shorter intervals cause more variation in concentration reading because the quadrupoles and detector have less time to switch between selected masses.

Performing a scan using SIFT-MS is achieved with two modes of operation, mass scans and selected ion mode (SIM) scans. Mass scans are typically used when a gas sample is analysed and the compounds of interest are not known. Mass scans produce plots showing the masses found in the sample on the x-axis and their concentrations (in ppb or cps) on the y-axis. An example of a mass scan plot is shown in Figure 2.1.

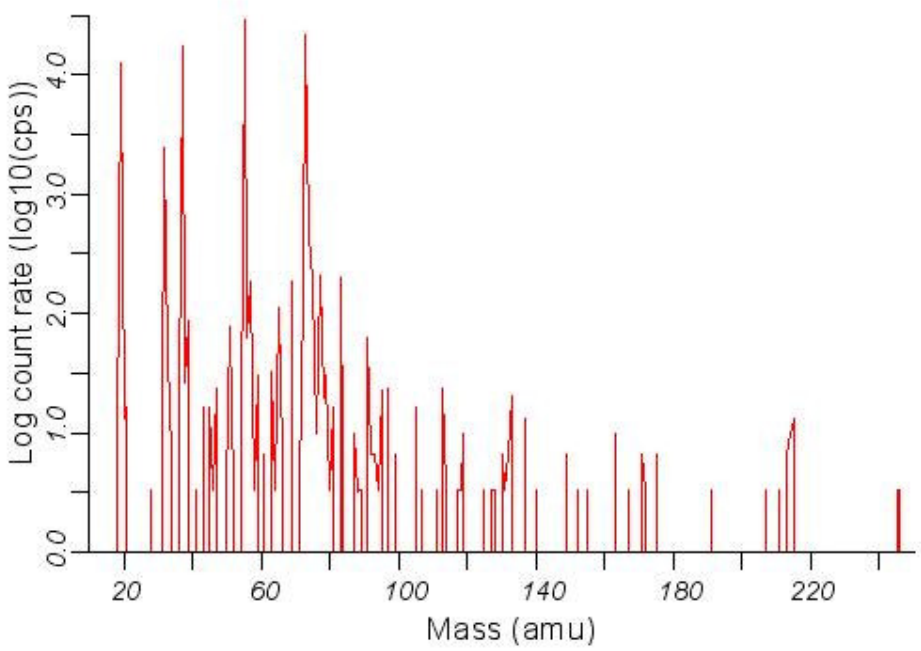


Figure 2.3 - Plot of Mass Scan from SIFT-MS

Selected ion mode (SIM) scans on SIFT-MS are performed when the compounds of interest in a sample are known. These scans give the concentration of a sample measured by SIFT-MS over time. Before performing a SIM scan a kinetics file is required that gives information about the compounds being monitored and the precursor(s) used. The kinetics informs the software as to which compounds are being monitored, the product masses produced from each precursor and their respective

ratios. The overall concentration given is an average of the level obtained from each precursor reacted with the molecule. An example of a SIM scan is shown in Figure 2.4.

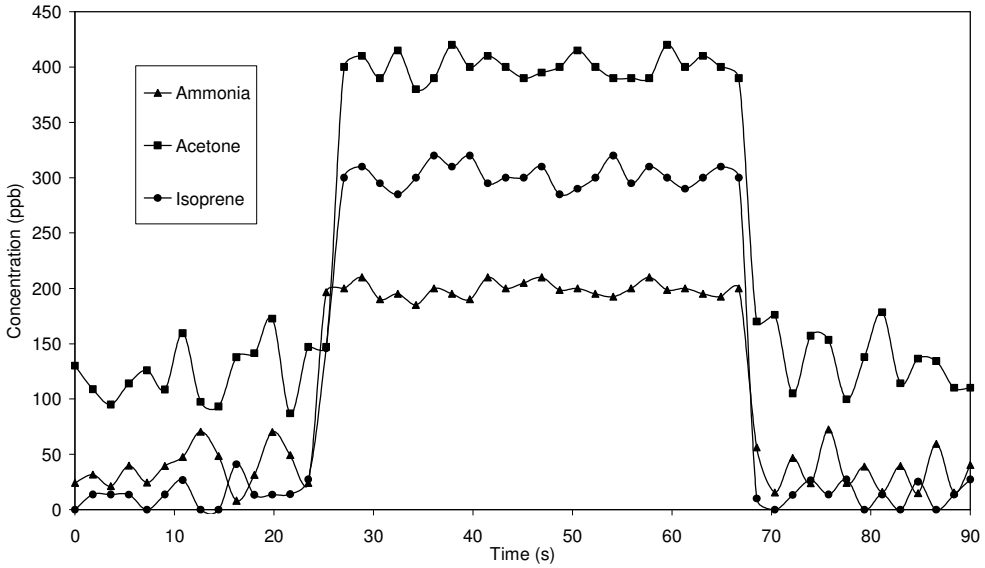


Figure 2.4 - SIM Scan from SIFT-MS

2.4 Concentration Calculation

During a SIM scan of a sample, SIFT-MS returns raw voltages, referred to as counts, to the software that controls the instrument. These counts are then converted into a concentration in ppb. Several steps are then taken to calculate the concentration of an analyte measured by SIFT-MS (Syft, 2004b). The first step uses the rate law to describe the reaction between an ion and a neutral molecule:

$$I = I_0 e^{-(k[A]t)} \tag{2.2}$$

Where:

I	=	the ion concentration at time t
I_0	=	the ion concentration at time zero
k	=	the rate coefficient
$[A]$	=	the concentration of the neutral analyte
t	=	the reaction time

The counts of product ion (P) can be viewed in terms of the precursor ions:

$$P = I_0 - I \tag{2.3}$$

Substituting Equation (2.3) into Equation (2.2) and taking the ratio of the result over Equation (2.3) yields:

$$\frac{P}{I} = \frac{I_0\left(1 - e^{-(k[A]t)}\right)}{I_o e^{-(k[A]t)}} \tag{2.4}$$

Eliminating I_0 and taking the limit of $k[A]t << 1$ which is always the case at trace concentrations, Equation (2.4) simplifies to:

$$\frac{P}{I} = k[A]t \tag{2.5}$$

Equation (2.5) gives a concentration of the compound in the flow tube for a single product and precursor cluster. Rearranging Equation (2.5) and making allowance for

multiple precursors and products, and the differing diffusion rates of the precursor and product ions (D_i), the concentration of analyte in the flow tube is given by:

$$[A] = \frac{1}{t_r} \frac{\sum_{1}^{n} \frac{I_{si} + I_{\sec i}}{b_r}}{\sum_{1}^{n} k_i I_{pi}} D_f \qquad (2.6)$$

Where:

$[A]$	=	analyte concentration (molecules cm^{-3})
t	=	reaction time (seconds)
I_{pi}	=	precursor ion counts (counts per second, cps)
I_{si}	=	primary product counts (cps)
I_{seci}	=	secondary product counts (cps)
k_i	=	rate coefficient (molecules cm^3 s^{-1})
b_r	=	branching ratio
D_f	=	diffusion factor, usually assumed to be 1.

Calculation of Equation (2.6) requires calculation the reaction time and ion velocity (v_i) before proceeding. Both the reaction time and ion velocity require the calculation of the carrier gas velocity, v_g.

$$v_g = \frac{4}{\pi D_t^2} \frac{\Phi_c \times T_g}{P_g \times T_{ref}} \qquad (2.7)$$

Where:

Φ_c = gas flow rate (Torr m^3 s^{-1})

P_g = pressure in the flow tube (Torr)

T_g = temperature of the gas in the flow tube (Kelvin, K)

T_{ref} = reference temperature for rate coefficients (273 K)

D_t = flow tube diameter (m)

Using the carrier gas velocity defined:

$$v_i = c\, v_g \tag{2.8}$$

Where c is the axial correction factor and is typically 1.5 for a straight flow tube.

Then t is defined as:

$$t = \frac{l}{v_i} \tag{2.9}$$

Where:

l = the reaction length (metres)

The quantity [A] is an instantaneous measurement of the concentration of compound in the flow tube. It is then used to calculate the concentration, $[A]_{ppb}$ in the sample using:

$$[A]_{ppb} = \frac{[A]T_g R\Phi_{Total}}{N_a \Phi_R p_g}\left(1\times10^{12}\right) \tag{2.10}$$

Where:

R	=	universal gas constant (62.4 LtorrK^{-1}mol^{-1})
N_a	=	Avogadro's number (6.022×10^{23} mol^{-1})
$[A]$	=	analyte concentration (molecules cm^{-3})
T_g	=	temperature of the gas in the flow tube (Kelvin, K)
Φ_{Total}	=	total flow in flow tube (torr Ls^{-1})
Φ_S	=	sample flow through capillary (torr Ls^{-1})
p_g	=	flow tube pressure (torr)

The constant 1x10^{12} in Equation (2.10) arises due to conversion to a ppb concentration and conversion of units for R.

In summary, Equations (2.2) – (2.10) are used to determine the reported concentration. The initial measurements of each product ions are I_{si} and I_{seci} in Equation (2.6) are the raw measurements from the detector in SIFT-MS and the calculated result, as derived, is the concentration of the observed VOC.

2.5 Applications of SIFT-MS

Numerous applications for SIFT-MS exist, some which are currently being examined. Applications of VOC detection fall into two primary categories. Firstly, the detection of a unique compound where it does not normally exist. Secondly, the detection and quantification of a compound to a specific level for comparison with a norm or standard. Specific possible SIFT-MS applications include (Syft, 2006):

- General Applications
  - Flavour release research.
  - Analysis of landfill gas.
- Healthcare and medical diagnostics
  - Screening for conditions such as diabetes.
  - Detection of carcinoma.
  - Evaluation of renal function in critically ill patients.
  - Diagnosis of ischemic bowel and shock in critically ill patients.
  - Diagnosis and monitoring of schizophrenia and bipolar disorders.
  - Early real-time diagnosis of fungal infections in bone marrow transplant patients.
- Industrial and Food Production Process Control
  - Real-time (production line) detection of contamination and spoiling and quality control purposes.
- Ambient Air Monitoring
  - Real time monitoring of air pollution and tracing of possible pollution sources.
- Fumigant Detection
  - On-site detection of fumigants in shipping containers.
- Homeland Security
  - Detection of VOCs associated with explosives.

- Petroleum Exploration
  - Testing of hydrocarbon plumes to detect the presence and composition of oil and gas deposits.

Mass spectrometers have been in existence for over 100 years now and have been used in numerous applications. SIFT-MS has many applications and potential applications that are highly desirable in today's world, it offers advancements in sectors such as fuels, security, health and safety and clinical applications. Specifically in the area of healthcare, SIFT-MS offers sensitive analysis that is much faster than many current methods and will provide opportunity for new studies of health research.

2.6 Chapter Summary

SIFT-MS technology has existed for approximately 30 years and was conceived through the development of FA-MS technology. Initially developed for the study of kinetics on gas phase ion-neutral reactions, SIFT-MS is now used for the detection VOCs. Advantages of SIFT-MS over other mass spectrometry techniques include:

- The ability to sample whole air samples or the headspace above samples
- Real time analysis of samples
- Analysis of humid samples
- Samples do not require chromatographic separation.

The development of the smaller SIFT-MS instrument has allowed the technique to become commercially available. The initial cost and running costs of SIFT-MS are similar to the widely used GC-MS technique. However, SIFT-MS offers many

advantages over GC-MS, perhaps the most advantageous is that SIFT-MS detects compounds down to ppb levels in real time versus the 10 – 30 minutes taken to analyse a sample using GC-MS.

Current developments will find new uses for SIFT-MS technology in the commercial scene especially in areas of fuel exploration, security, health and safety and clinical applications, some of which are presently put into practice such as bio-security in customs and research into clinical studies.

3

Direct Breath Collection Device for SIFT-MS Instruments

Currently there is no standardised method of sampling breath with SIFT-MS instruments. Because a SIFT-MS instrument has the ability to analyse whole air samples, a standardised method for direct breath analysis would be highly advantageous for research into breath testing for clinical diagnosis. Standardising the way in which breath is sampled will allow a consistent method of measuring analytes that could become an industry and research standard.

Previous methods of testing breath using the SIFT-MS technique involved blowing past the capillary inlet using straws. However, simply blowing past the capillary with a straw is not an ideal way of providing a sample as the subject may not be able to reach the inlet. In addition, different levels of condensation, moisture and temperature conditions will result in sample variability. Therefore, a direct breath collection device for a SIFT-MS instrument is required which not only standardises, but also optimises, breath collection.

3.1 Design Specifications

1. Ergonomics: accommodating use of the device by people of various sizes, and physical abilities.
2. Materials: materials used on the device must not release VOCs that might contaminate breath samples.
3. Ease of use: the device must not provide too much breathing resistance for the subject providing the sample. However, the device must also sample in a fashion best suited to the SIFT-MS instrument, which requires adequate exposure time to the breath and obtain multiple VOC level readings.
4. Heating: the sampling device must be heated sufficiently to prevent the condensation of breath vapour on the internal surfaces of the device.
5. Safety: users of the device must not be harmed by elevated temperatures of the device operating conditions.

3.2 Ergonomics

A breath collection device for a SIFT-MS instrument has to incorporate some degree of flexibility to allow for a range of user anthropometrics. Figure 3.1 shows a possible range of movement required of the direct breath collection device for a SIFT-MS instrument. To achieve this range of movement two design approaches were considered, a flexible tube extension from the original plumbing, and a rigid tube with a rotating joint.

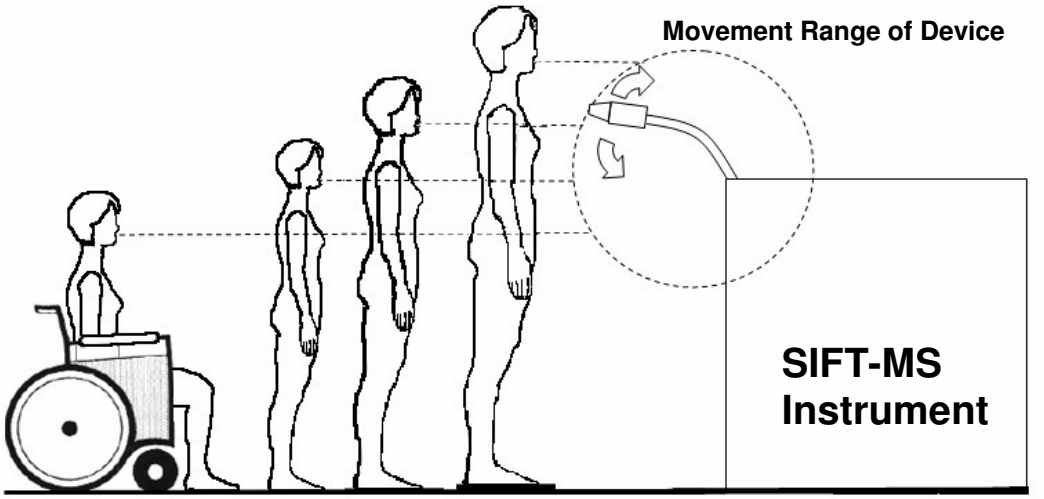


Figure 3.1 - Direct Device Range of Movement

Options for flexible attachments to the SIFT-MS instrument plumbing included Fluorinated Ethylene Propylene (FEP) tubing and flexible stainless steel pipe. FEP is a material similar to Poly Tetra Fluoro Ethylene (PTFE), also known under the DuPont trademark name as Teflon™. The FEP tube was tested as a possible extension to the instrument inlet, however mass scans showed that significant levels of VOCs were emitted from FEP at elevated temperatures. A possible source of the emitted VOCs is from plasticiser used when manufacturing the FEP tubing. Figure 3.2 and Figure 3.3 show VOC emission by SIFT-MS mass scans from the FEP tubing heated to 100°C and levels of VOCs at 23°C (room temperature) respectively. Because the FEP tubing emitted high levels of VOCs when heated, it was deemed not suitable for use on a direct breath collection device.

A stainless steel pipe was tested as a possible component to the direct breath collection device to allow the movement required. The stainless steel pipe investigated was constructed from bellows which allowed it to flex. Tests showed that these bellows had a large internal surface area that caused compounds (for example.

ammonia) to reside and accumulate inside the tube and slowly release over time. Figure 3.4 shows the difference in decay rates of ammonia residing in a flexible stainless steel tube and a rigid tube.

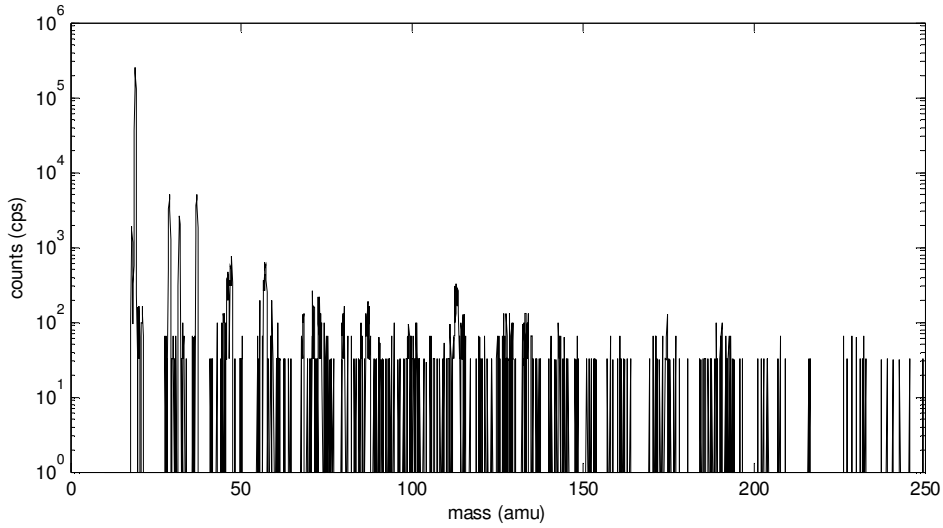


Figure 3.2 - Mass scan of FEP tube heated to 100 °C while sampling nitrogen

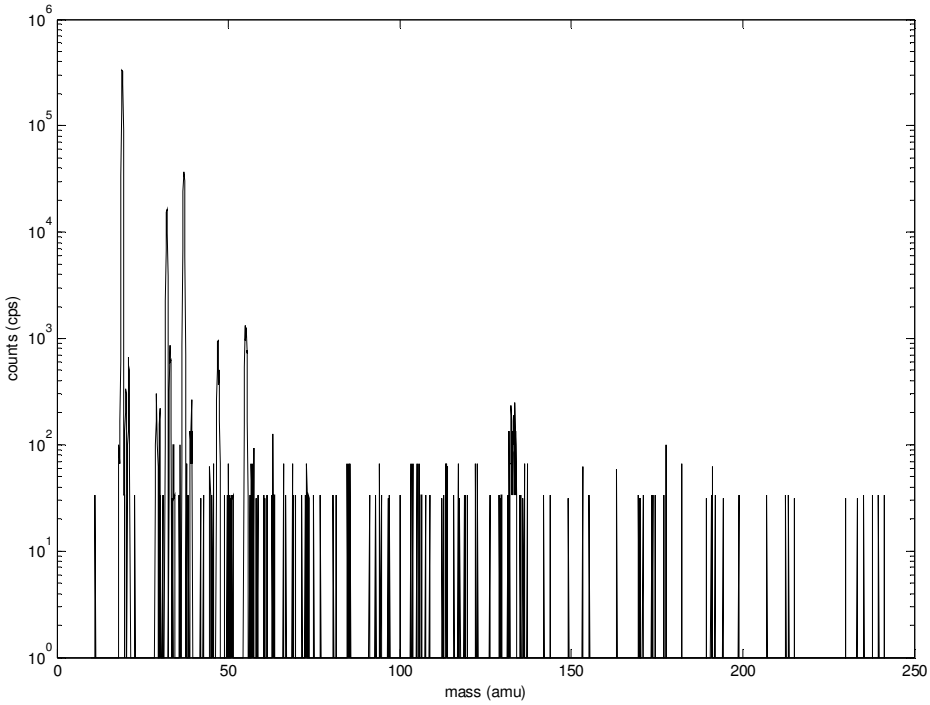


Figure 3.3 - Mass scan of FEP tube at room temperature (23 °C) while sampling nitrogen

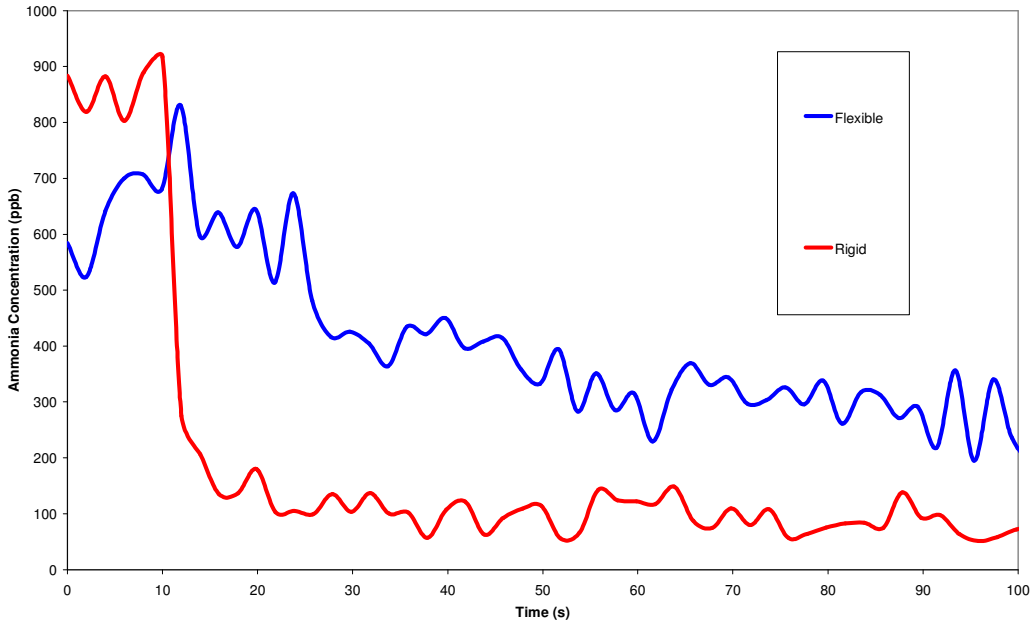


Figure 3.4 - Ammonia Decay in Flexible and Rigid Stainless Steel Tubes

Tests showed that both the FEP and flexible stainless steel tubes were not suitable alternatives to provide flexible access and direct breath collection for SIFT-MS instruments. Therefore, the direct breath collection device designed consists of rigid tube that incorporates a rotating joint to adjust position.

3.3 Materials Testing

It was important that the materials considered could operate indefinitely at their required temperature and did not emit VOC's that would contaminate samples. This requirement was tested by heating the materials to their operating temperatures and sampling the headspace air above them, or sampling nitrogen though tubes made from the material. Mass scans were then compared, indicating whether significant VOCs were emitted.

FEP, nitrile rubber and stainless steel were tested at their expected operating temperatures, as shown in Figures 3.2, 3.3, 3.5 and 3.6. All materials emitted VOCs when heated. Stainless steel emitted the least amount of VOC's. FEP emitted more VOC's during heating, as shown by comparing Figure 3.2 and Figure 3.3.

VOC emission was only problematic for those materials with a large surface area in contact with the breath. For all other materials, a small amount of VOC emission was acceptable. Therefore, the tubing in direct contact with breath flow was made from stainless steel.

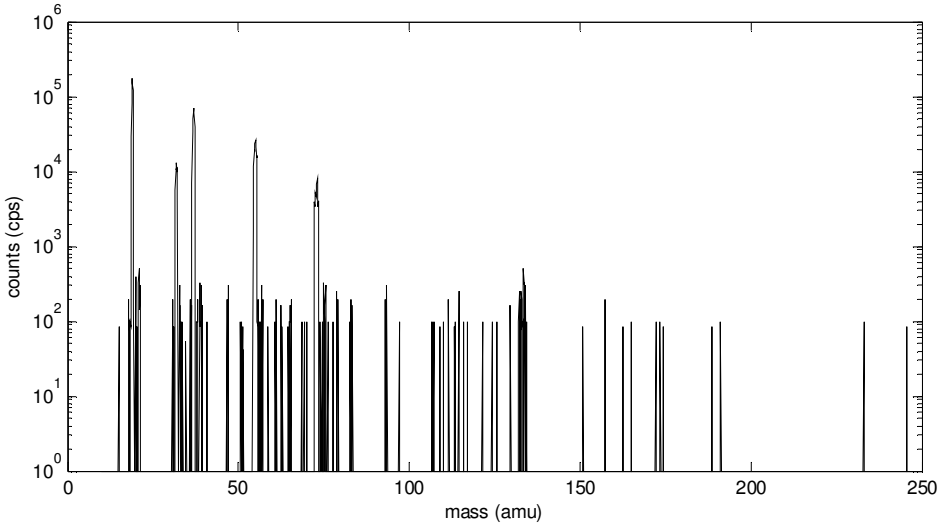


Figure 3.5 - Mass scan of stainless steel tube heated to 100 °C while sampling nitrogen

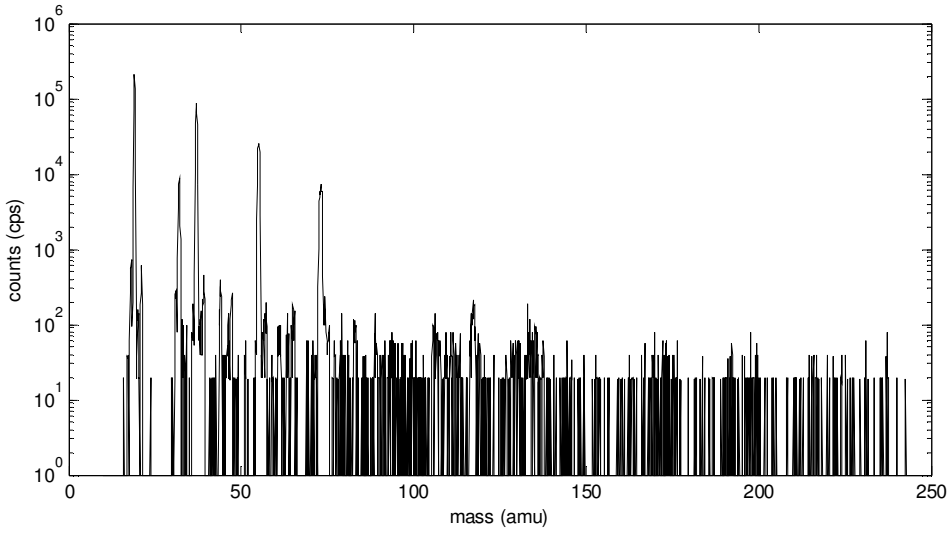


Figure 3.6 - Mass scan of headspace above nitrile rubber foam heated to 100 °C

3.4 Breath Flow Restriction

The subject using the direct breath collection device must introduce their breath to the SIFT-MS instrument capillary tube for analysis. From the capillary, the sample is rapidly injected into the SIFT-MS flowtube. The amount of resistance the subject experiences will depend partly on the length of tube prior to the capillary. Therefore, to minimise resistance, the capillary was placed as close to the user's mouth as possible. This criterion ensures the shortest possible distance is travelled by the breath to the capillary. Because each unit of the direct breath collection device will have a capillary, individual calibrations for the capillary tube on each unit are required. An example of calibration calculations for the capillary is included in Appendix A1.

The first prototype was designed to provide minimal resistance to the user of the device, by using a large inlet and exhaust. However, this prototype device was not

suited to the sampling method of a SIFT-MS instrument. In particular, the breath sample was exhausted from the inlet too quickly, resulting in single point measurements, as shown in Figure 3.7. Figure 3.8 shows a cross section design drawing of the first direct breath collection prototype.

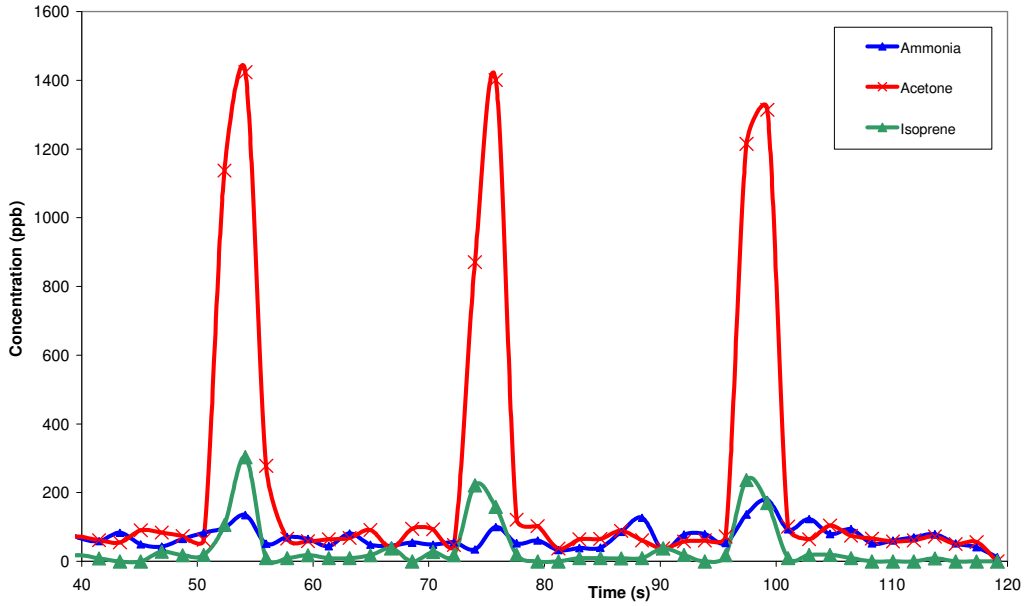


Figure 3.7 - Single Point Measurements from First Direct Breath Prototype

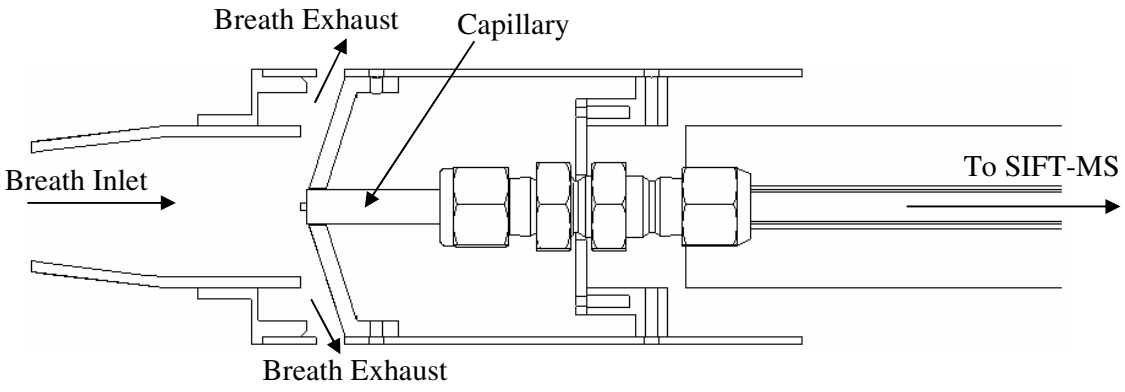


Figure 3.8 - Cross Section of First Direct Breath Prototype

Ideally, a SIFT-MS instrument has to be exposed to a sample long enough to obtain 3-4 point measurements to have a statistically reliable reading. Figure 3.9 shows an example of an ideal reading from a SIFT-MS instrument. To obtain a reading with multiple point measurements for an exhaled breath, a straw was used to extend the time of exposure the capillary had to the breath sample. The straw adds a restriction that effectively stretches the breath sample time, provided the use continues to supply enough input pressure. Therefore, the inlet and exhaust for the direct breath collection device were reduced to 6.5 mm outer diameter (O.D) and made from stainless steel tubing.

Figure 3.10 shows the capillary, inlet and exhaust of the direct breath collection device design, as housed inside its aluminium case. A complete set of workshop drawings for the direct device is included in Appendix A2

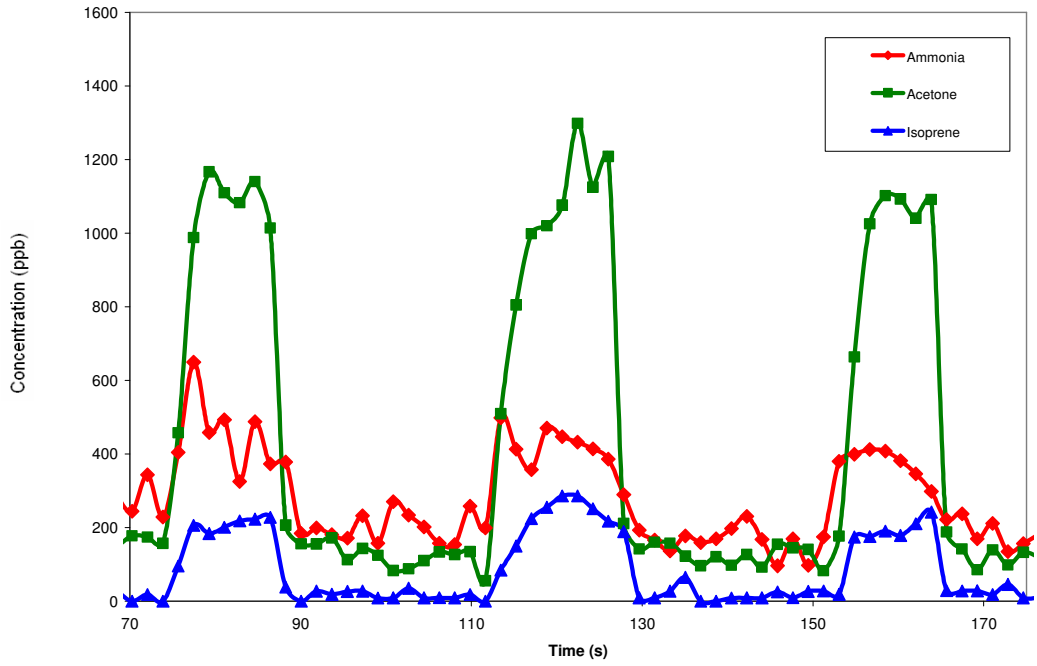


Figure 3.9 - Multiple point VOC measurements on a SIFT-MS Instrument SIM Scan

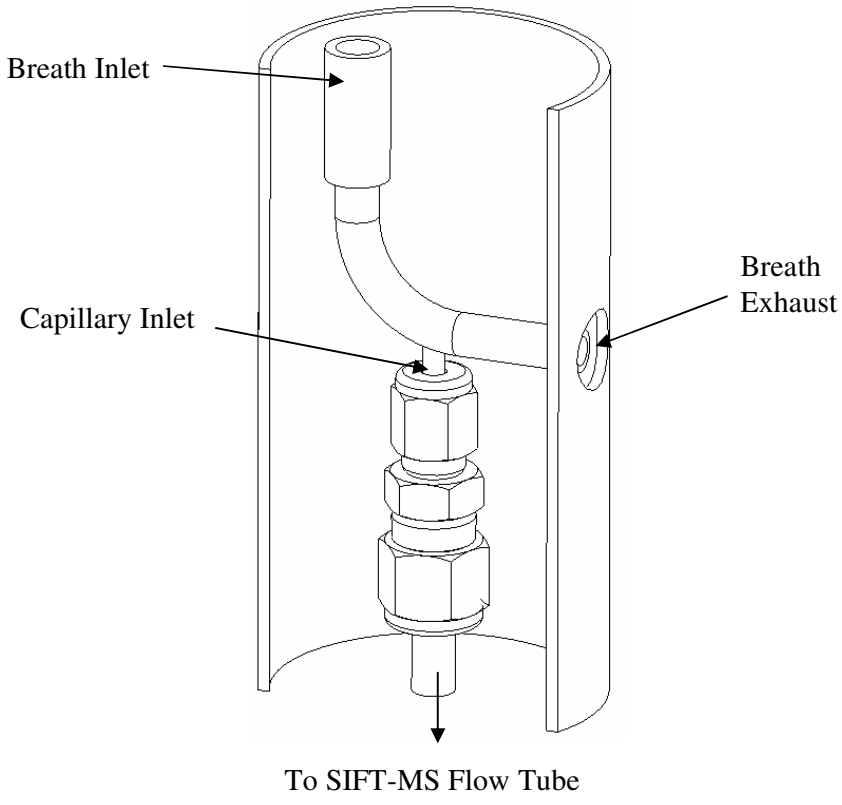


Figure 3.10 - Direct Breath Collection Device for SIFT-MS Instruments

By using a straw combined with a short length of pipe to the capillary, the resistance the patient experiences is minimised. In addition to minimising resistance, the straw allows an extended period of breath exposure for the SIFT-MS instrument to sample. Thus the sample collection process is optimised to best match patient and instrument requirements.

3.5 Operating Temperature

Because a capillary tube has a very small diameter it can be blocked easily by condensing water vapour or other foreign materials. To ensure that water would not condense within the capillary tube it must be heated above 100°C. Currently, SIFT-

MS instrument lines are heated to 120°C, which was determined by the operating limits of the materials and the heating method.

Because polar compounds such as ammonia have a high affinity for water, they are likely to be trapped in condensed moisture inside the device. For this reason if water vapour was condensing inside the device there would be an observed lower level of polar compounds relative to expected or known values. Figure 3.11 and Figure 3.12 show that the levels of acetone and ammonia, which are polar, have constant concentrations when measured at 80°C to 120°C, indicating that water vapour is not condensing using this design. However, to ensure that the capillary will not be blocked, the direct breath collection device is operated at 120°C, which is greater than the boiling point of water.

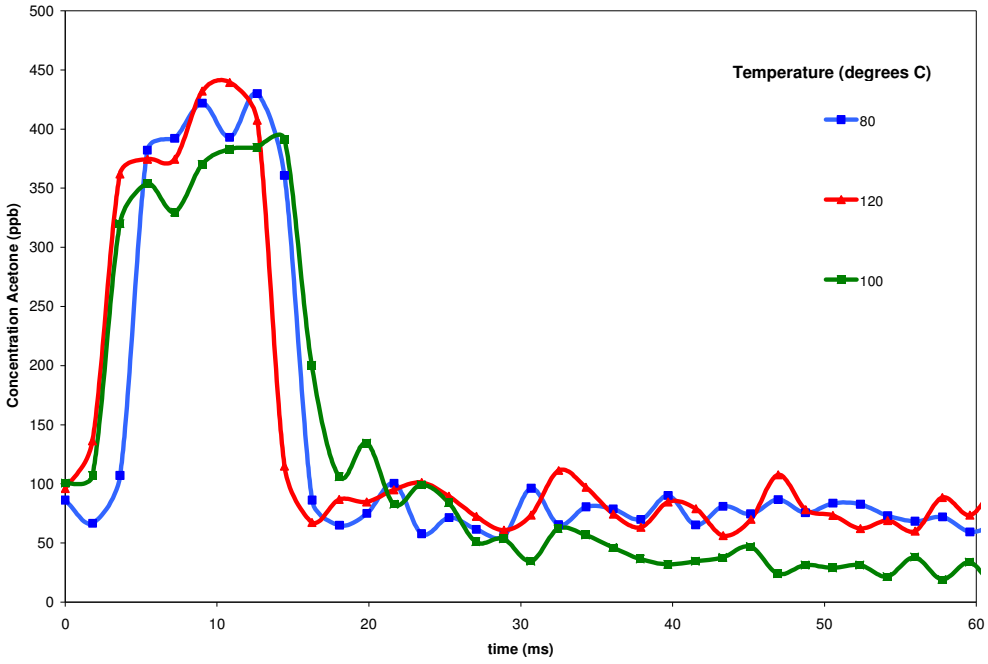


Figure 3.11 - Acetone Concentration with Temperature Variation

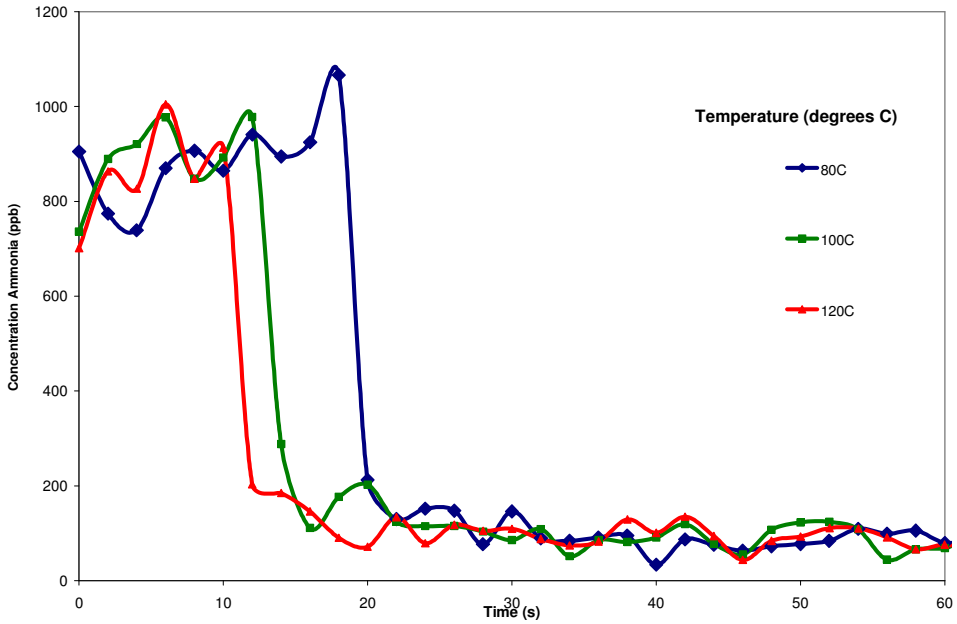


Figure 3.12 - Ammonia Concentration with Temperature Variation

Previously, the method of heating the sample lines on the SIFT-MS instruments involved using an electrical heat tape. Heat tape is essentially aluminium tape with a heating wire/element passed inside, with one side covered in adhesive coated fibreglass cloth. This method of heating works well but it is difficult to work, and difficult to modify once in place. A device for direct breath collection for a SIFT-MS instrument may require regular sterilisation, where the device inlet and exhaust system will have to be autoclaved. Therefore, because the device will have to be disassembled for sterilisation, heat tape would not be a suitable heating method.

Heating System

The chosen method of heating the direct breath collection device is silicon heating wires, commonly found in electric blankets. The silicon coated wires are relatively inert if run within their specified power rating. For assembly, the silicon wires are

easily wound round the internals of the device and can easily be removed for sterilisation of the device.

The direct breath collection device utilises two heating wires, one for the capillary and the second for the remainder of the tubing and fittings. By using two heating wires, the temperature of the capillary can be monitored and controlled at the specified operating temperature (120°C) independently from the rest of the device. The second heating wire also ensures that the rest of the device is approximately 120°C as it will have different heating requirements from the capillary.

The heating and temperature of the direct device is monitored via thermocouple wires and each heating wire is controlled by independent temperature controllers. Temperature controllers are essentially intelligent relay switches that monitor the temperature of the device and control its temperature. When the device cools down, the temperature controller will switch on the heating elements to heat the device to the specified temperature. The silicon wires operate from AC power which is stepped down to 20V from 240V mains by a transformer. Figure 3.13 shows a schematic of the wiring method of the heating wires for the direct breath collection device.

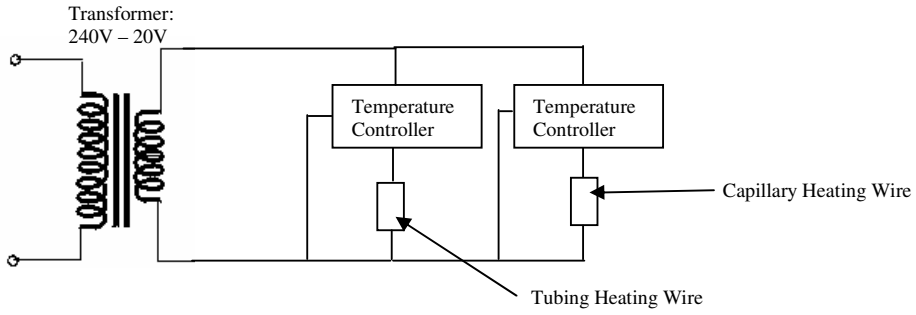


Figure 3.13 - Wiring Schematic of Heating Circuit for the Direct Breath Collection Device

3.6 Patient Safety

To ensure that patients would not come in contact with the elevated temperatures of the direct breath collection device, the heated sections were insulated. Insulation materials considered included nitrile rubber foam and vermiculite coated fibreglass. Both, nitrile rubber and vermiculite coated fiberglass can emit VOCs when heated. However, mass scans showed high levels of VOC emission at operating temperatures. Figure 3.6 shows an example of VOC emission from the nitrile rubber foam insulation that was observed in a mass scan on SIFT-MS.

VOC emission from insulating materials is eliminated by using an air gap between the heated section of the device and the device case, as shown in the design of the direct device (Figure 3.10). Because air has a low heat conductivity, stagnant air acts as an insulator; it is used in insulation applications such as double glazed windows. Therefore, by using an air gap as an insulator, sample contamination from VOC emission is minimised and patients are protected from high operating temperatures of the device.

3.7 Chapter Summary

To satisfy the range of user anthropometrics, a rotating joint was used to attach the direct breath collection device to the SIFT-MS instrument. Flexible tubes were also considered as possible methods to achieve similar user flexibility. The two options were FEP tubing and flexible stainless steel. However, FEP tubing had high emission of VOCs at the required operating temperature and flexible stainless steel tubing led

to trapping and slow release of polar molecules. Therefore, rigid stainless steel tubing was used for the direct breath collection device.

To minimise flow resistance experienced by the patient, the capillary (from which the SIFT-MS instrument samples) was placed as close as possible to the users mouth. However, to optimise sample collection, the inlet and exhaust had to be reduced to 6.35 mm O.D stainless steel tube and straws used as disposable mouthpieces. By reducing the inlet and exhaust the sample collection by SIFT-MS was optimised.

Suitable materials for the device included stainless steel and FEP at lower temperatures and out of direct contact with the breath flow. These materials were chosen to minimise the amount of VOC emission from the device. The optimal operating temperature of the device was found to be 100°C - 120°C, which ensures that water vapour will not condense inside the capillary and cause blockage. This temperature was also limited by the materials and current heating method of the SIFT-MS instrument lines. The device is heated with two silicon heating wires that operate from 20V AC power that is stepped down from mains with a transformer. Both heating wires are independently controlled with temperature controllers. To ensure patient safety from the elevated device operating temperature, an air gap and FEP coated device case are used as an insulator which also minimised external VOC contamination of samples from the insulating material.

4

Alveolar Breath Collection Devices

This chapter reviews previously designed alveolar breath collection devices designed for research or commercial use. Investigation of these devices provides information as to the best design approach for a breath collection device for SIFT-MS applications. Each section of the chapter covers a specific device, with sections relating to its use, design, aims, method of operation and materials required.

4.1 Breath Collecting Apparatus (BCA) (Phillips, 1997)

Use:

The BCA, designed by Phillips (1997) is used to collect alveolar breath samples onto sorbent traps for analysis with the GC-MS technique. Using breath samples collected with the BCA, the levels of VOCs in the room air were compared to expired alveolar breath. This device was used for research purposes and is not commercially available.

Design Aims:

- Subject comfort – the subject providing the breath sample does not experience any resistance during exhalation
- Subject safety – use of sterile disposable mouth pieces are used with inlet and outlet valves to prevent any potential hazard of exposure to infection

- Alveolar sampling – breath sample collected excludes expired deadspace volume
- Concentration of sample – breath was concentrated onto sorbent traps made with specially selected adsorbent materials that are thermally desorbed later.

Operation Method:

During operation, the subject wears a nose clip and breathes through a disposable mouthpiece into the BCA. Exhaled breath is stored in a long tubular reservoir that is open to the atmosphere at the end opposite from the subject. As the subject exhales into the device, the residing air inside the BCA is exhausted out the other end of the reservoir. Collection of the sample is through an inlet close to the subject's mouth as shown schematically in Figure 4.1.

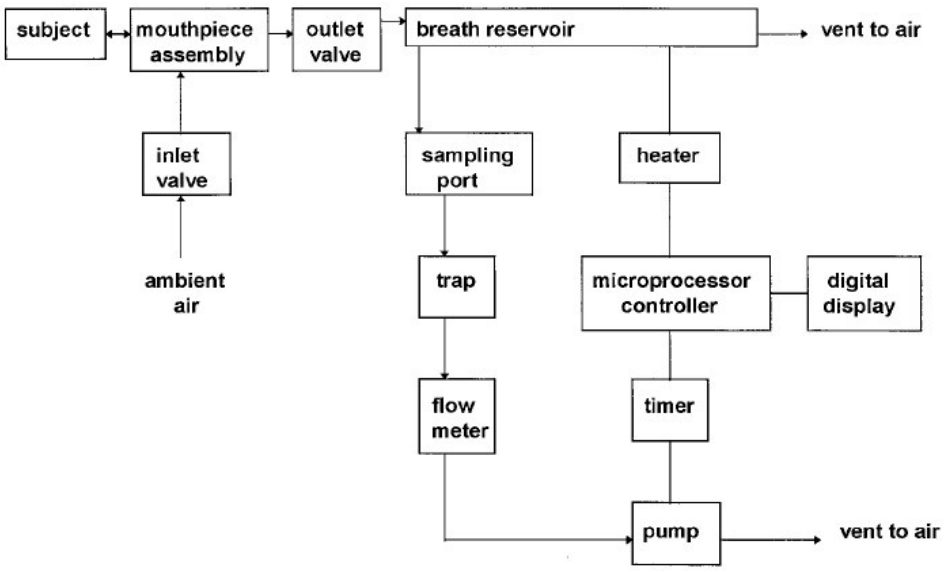


Figure 4.1 - Operation of the Breath Collecting Apparatus (Phillips, 1997)

By collecting the breath sample closest to the subject's mouth, the deadspace volume that is exhaled first will be at the opposite end of the reservoir, furthest from the

sampling port, thus allowing collection of the alveolar breath sample. This method of collecting breath into a long tubular reservoir, called the Haldane-Priestly tube, is used in a number of studies.

To prevent condensation of exhaled breath vapour on the internal surfaces of the BCA, the breath reservoir tubing is heated to 40°C. Subjects are required to breathe into the device for approximately five minutes to collect the necessary 10 litres of sample required for the sorbent trap materials used.

Materials:

To prevent contamination of breath samples from external VOC sources, the BCA is constructed from stainless steel, except for the disposable plastic mouthpiece. To prevent/limit chemical contamination, the disposable mouthpiece is manufactured with a low content of plasticizer. Specially selected adsorbent materials are used to collect and concentrate breath samples for later desorption and GC-MS analysis.

4.2 Portable Spirometer/Alveolar Breath Collection System (Raymer *et al.*, 1990)

Use:

The alveolar breath collection device designed by Raymer *et al.* (1990) was used for breath testing research and is not a commercially available product. The device collects samples of alveolar breath into evacuated stainless steel containers. The collected samples were analysed using the GC-MS technique.

Design Aims:

- Portable device – the unit is stored in a case for transportability.
- Components of the device are easily sterilised therefore reusable.

Operation Method:

This alveolar breath collection device works on the same principle as the Haldane-Priestly tube, where the exhaled breath is stored in a long tubular reservoir and sampled from the subject end. The reservoir on this device is a Teflon tube 760 cm long and 1.3 cm internal diameter (i.d), providing 1 litre of expired sample. The breath sample is collected from the subject end of the reservoir and is stored in an evacuated stainless steel canister. Figure 4.2 shows a drawing of the device and its storage case.

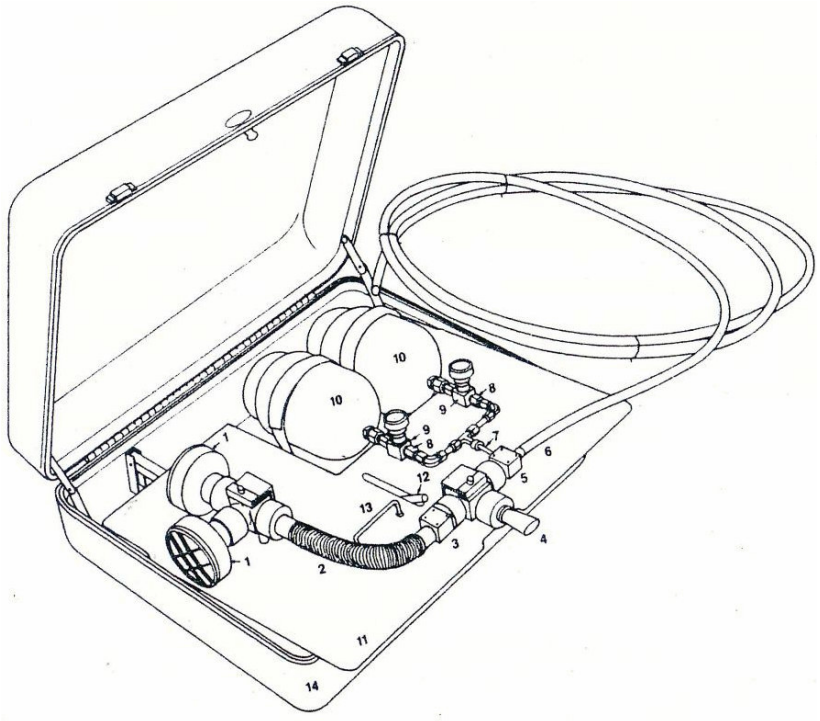


Figure 4.2 - Portable breath collection system (Raymer et al., 1990) Numbered parts are identified as follows: (1) organic vapour respirator cartridge; (2) flexible disposable polyethylene tubing; (3) one-way Teflon inhale valve; (4) mouth bit; (5) one-way Teflon exhale valve; (6) Teflon tube to

contain exhaled breah; (7) Sampling port; (8) flow restricting orifice; (9) on-off valve; (10) polished canisters; (11) aluminium platform; (12) handle; (13) handle; (14) aluminium case.

To prevent influence of VOC concentrations from ambient air sources, subjects inhale filtered VOC free air before exhaling into the device. To verify the device a CO_2 mass spectrometer was used to monitor the CO_2 levels of the samples collected in the stainless steel canisters.

Materials:

This portable breath collection system uses stainless steel canisters for the storage of breath samples. These stainless steel canisters have a highly polished interior surface to prevent the adhesion of VOCs to the canister surface. A number of components are made from Teflon, such as the reservoir tubing, valves and mouthpiece. A section of polyethylene flexible tubing is included to make the device more ergonomic. Other materials include carbon respirator cartridges used to filter ambient air to provide the VOC free inhaled air for the subject.

4.3 End-Expiratory Air Sampling Device (Yeung *et al.*, 1991)

Use:

The end-expiratory air sampling device by Yeung *et al*. (1991) is used to collect breath samples for research on breath hydrogen concentrations in infants. Samples are used to monitor breath hydrogen for clinical diagnosis of lactose malabsorption. The device collects and stores breath samples in a syringe and they are analysed using the

GC-MS technique. This device was designed for research and is not commercially available.

Design Aims:

- Find a non-invasive method to diagnose lactose malabsorption in infants
- Alveolar breath collection from infants
- Analysis of sample using the GC-MS technique.

Operation Method:

This end-expiratory sampling device measures exhaled breath from the infant using seven tungsten hot-wires. The hot-wires are part of a wheatstone bridge circuit and the exhaled breath cools them, resulting in a voltage imbalance that, once calibrated, can be measured as a flow-rate. Nasal prongs are used to direct the exhaled breath into the device. A programmed control module measures the exhaled breath from the subject and the sample is extracted appropriately with a syringe in 0.1 ml increments. The time required to collect the 1 ml sample required was 0.5 – 4 minutes depending on the time taken for the infant to accept the nasal prong. Figure 4.3 provides a schematic representation of this end-expiratory air-sampling device.

Materials:

This end-expiratory air sampling device uses a glass syringe to store the collected breath samples. Other materials used on the device were not explicitly mentioned in the documentation.

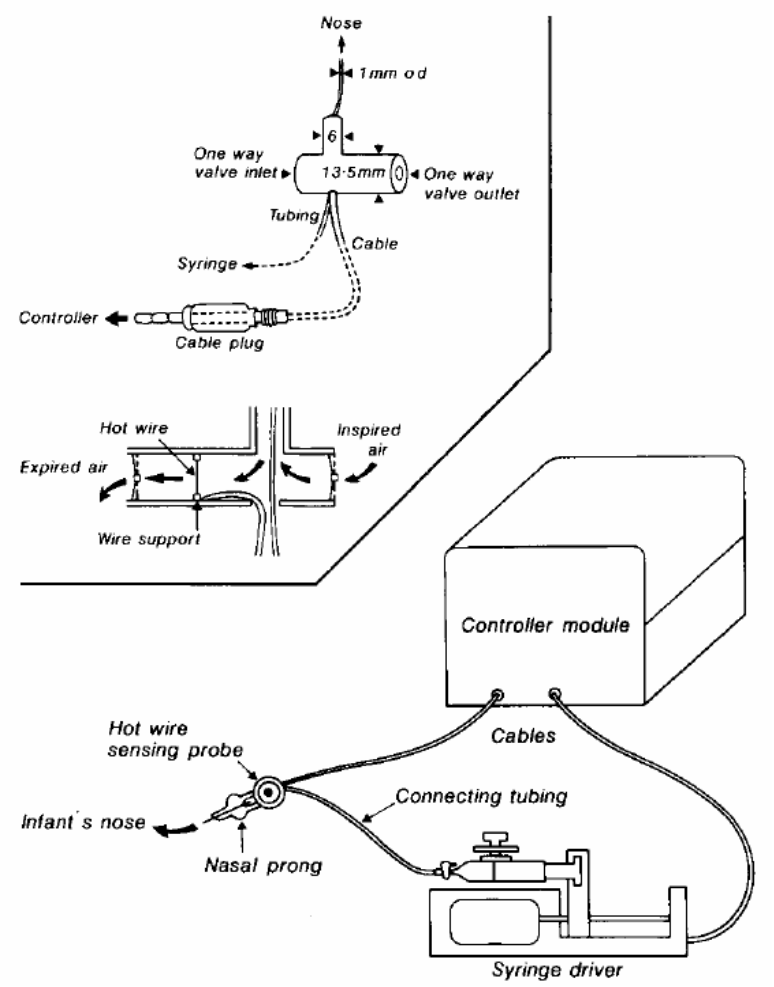


Figure 4.3 - Schematic of End-Expiratory Air Sampling Device (Yeung et al., 1991)

4.4 Breath Sampler for Solvent Analysis (Dyne *et al.*, 1997)

Use:

The device by Dyne *et al.* (1997) is designed for collection of alveolar breath samples and storage onto an adsorption tube. The adsorbed sample is then thermally desorbed and analysed with the GC-MS technique. This device is commercially available, manufactured by Markes International Ltd under the name "Bio-VOC sampler".

Design Aims:

- Collection of alveolar breath sample
- Device is small and portable
- Robust and easy to use
- Interfaces with readily available laboratory equipment
- The commercially available version is a single use/disposable system.

Operational Method:

This breath collection device is another design based on the Haldane-Priestly tube. Breath is exhaled into the device and is exhausted at the other end. When the subject stops exhaling, one-way valves at both ends of the reservoir hold the alveolar sample inside the device.

The device consists of a rigid casing that houses a collapsible Tedlar sampling bag, approximately 85 ml in volume. Once the breath is captured inside the sampling bag, the device is attached onto a thermal desorption tube where the sample is stored for analysis with GC-MS at a later stage. The breath sample is stored on the sorbent material by collapsing the Tedlar bag inside the casing and forcing the sample through the sorbent tube. Figure 4.4 shows a cross sectional diagram of the gas sampler by Dyne *et al.*

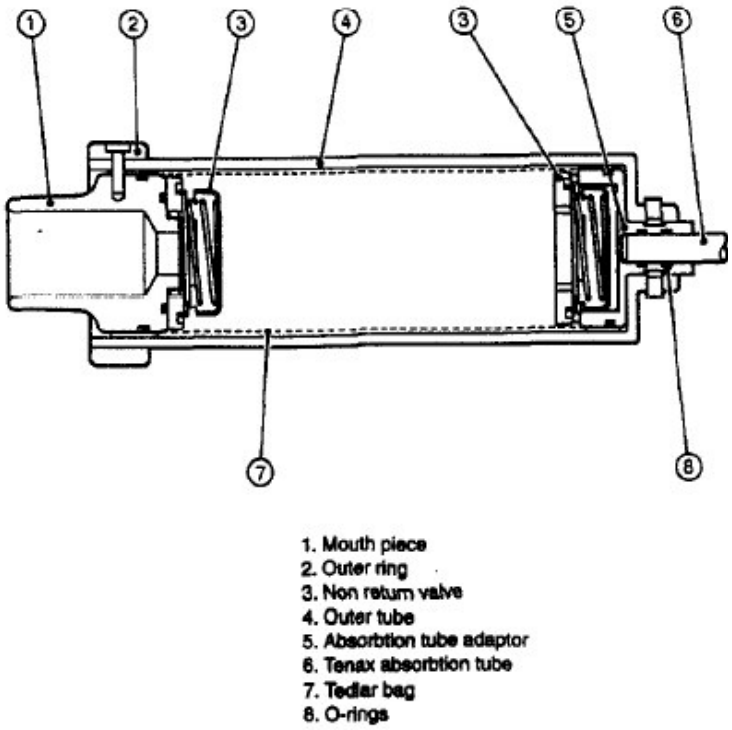


Figure 4.4- Cross Section of Breath Sampler (Dyne *et al.*, 1997)

Materials:

The outer casing of this device was constructed from aluminium due to its lightweight properties and it would not come in contact with the breath sample. Brass rings are used to hold the gas-sampling bag open inside the device. Disposable mouthpieces come with the device and are made from cardboard. The gas sample bag inside the device is made from Tedlar.

The commercially available version, the BioVOC sampler, is made with a plastic external case and does not use the internal Tedlar bag. Because a Tedlar bag is not used inside the device, the brass rings are no longer required and the one-way valve at the exhaust end of the device is removed. These changes are possible without contamination issues as the device is used only once and discarded.

4.5 CO_2 Alveolar Gas Sampler (Schubert *et al.*, 2001)

Use:

The CO_2 controlled alveolar gas sampler by Schubert *et al.* (2000) is used to collect breath samples from mechanically ventilated patients. Exhaled CO_2 levels are monitored to detect the alveolar breath region, which is approximately 6%. Alveolar breath samples are stored on activated carbon sorbent traps. The stored samples are thermally desorbed and analysed using the GC-MS technique. This device is not commercially available and was used for research purposes only.

Design Aims:

- To evaluate the method of using CO_2 as a trigger to sample exhaled alveolar gasses on mechanically ventilated patients
- To assess the effect of using CO_2 as a trigger of alveolar breath on the concentrations of exhaled compounds.

Operation Method:

This alveolar breath collection device uses a 930 Siemens-Elema CO_2 analyser to detect the presence of alveolar breath in mechanically ventilated patients. The 930 Siemens-Elema analyser uses infrared adsorption to recognise CO_2 that originates from alveolar gasses at a concentration of 5 – 6%. When the CO_2 analyser detects breath levels by crossing this CO_2 threshold, a two-way solenoid valve is activated to collect the sample onto activated charcoal sorbent traps. Figure 4.5 shows a schematic of the CO_2 controlled alveolar gas sampler.

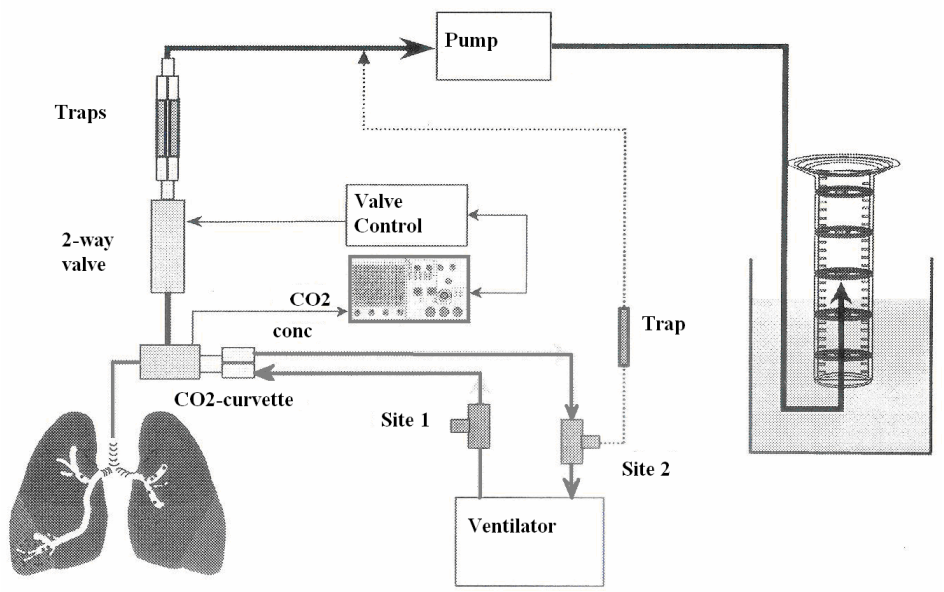


Figure 4.5- Schematic drawing of CO2 Controlled Alveolar Sampler (Schubert *et al.*, 2001)

Materials:

The materials used were standard medical grade plastics, glass and stainless steel components. Activated charcoal is used as the adsorbent sampling material.

4.6 Alveosampler (Quintron)

Use:

The Alveosampler is a disposable version of the Haldane-Preistly tube and is a commercially available product manufactured by Quintron. It permits single-patient use of an alveolar breath sample device that stores the collected sample in a standard syringe.

Design Aims:

- Disposable device for single patient use to prevent cross-infection and/or alveolar contamination between subjects
- Permits the use of a standard syringe to collect the breath sample
- Use of disposable gas sample bags saves time by not having to sterilise them.

Operational Method:

The Alveosampler is a commercially available device that is based on the Haldane-Priestly tube, where the breath is exhaled into a reservoir and the sample is collected from the subject end of the reservoir. The Alveosampler differs from the standard Haldane-Priestly tube because the reservoir is a gas sampling bag rather than a rigid tube. Figure 4.6 shows an image of the manufactured Alveosampler.

Figure 4.6 - The Alveosampler by Quintron

Materials:

The majority of the Alveosampler is made from medical grade plastics. However, the gas sample bag material was not specified. Mouthpieces for the Alveosampler are made from tubular cardboard, which are cheap and easy to dispose.

4.7 Chapter Summary

A majority of the previously designed alveolar breath sampling devices are based on the Haldane-Priestly tube. The Haldane-Priestly tube collects an alveolar breath sample but does not allow the flexibility of collecting breath from any specified region of an exhalation, nor does it allow for the separate collection of the deadspace volume. Thus, it is effectively a feed-forward design that may not work in extreme cases where the exhaled breath volume from a particular subject is significantly larger or smaller than normal. More importantly, the Haldane-Priestly tube does not offer programmable flexibility to collect the best sample.

Two of the devices collected breath samples by other means. Schubert *et al.* (2001) measured the volume of an exhaled breath. Yeung *et al.* (1991) measured the exhaled alveolar CO_2 concentration to collect breath samples. Both of these approaches also did not offer the flexibility in use required as a research device for SIFT-MS breath sample collection.

Apart from the device by Schubert *et al.* (2001), the reviewed devices included the use of sorbent traps to store the gas sample. Stored samples were then analysed using the GC-MS technique after thermal desorption. All these devices were thus primarily designed to interface with GC-MS and thermal desorption instruments, and are not necessarily the ideal solution for the SIFT-MS technique. Although SIFT-MS can analyse thermally desorbed samples, quantitative analysis is much more complex than analysis of whole air samples. In addition, the time required for thermal desorption of collected samples voids the real time ability of SIFT-MS and also requires further

complex equipment. Thermal desorption is also not suited for collection and analysis of highly water soluble compounds such as ammonia and ethanol.

The alveolar breath collection device by Schubert *et al.* (2001) did collect whole air samples, but the volume of sample collected was only 1 ml in 0.1 ml increments per breath. The minimum volume of sample required for an analysis with a SIFT-MS instrument would be hundreds of millilitres. Thus, the alveolar breath collection device by Schubert *et al.* (2001) would be a very time consuming method to collect the sample volume required for analysis using the SIFT-MS technique.

All the reviewed devices did not offer collection of samples in the form of whole air that would simplify analysis in comparison to analysis of thermally desorbed sample when using the SIFT-MS technique. Additionally, they did not provide the desired programmable flexibility desired for the study of sample delivery effects on the concentration of VOCs measured from exhaled breath. Therefore, there is a need for a suitable device for the remote collection of breath samples for analysis using the SIFT-MS technique.

5

Remote Collection of Breath Samples for SIFT-MS

Even with the reduction of its size, SIFT-MS is still a large instrument and the smallest generation of the device weighs approximately 400 kilograms. The pumps required to create vacuum pressures in the flow-tube, and the upstream and downstream chambers create the majority of weight associated with SIFT-MS. Aside from the overall weight, SIFT-MS also requires continued sources of helium, argon and power. Therefore, it would not be feasible, currently, to transport SIFT-MS around a hospital to perform breath tests.

Patients must therefore come to the instrument to provide a breath sample. However, in clinical applications, subjects may not be able to come to the instrument for a variety of clinical reasons. A much more practical approach would be to remotely collect breath samples from the patient and transport the collected samples to SIFT-MS for analysis. Therefore, a device is required for the remote collection of alveolar, and other forms of breath samples. This chapter examines the requirements for a remote breath collection device attuned with SIFT-MS.

5.1 Design Aims

A device is required that will aid investigators in the understanding of the effects of sample delivery on VOC concentrations. Therefore, a suitable device is required that determines when to collect the breath sample at a specific point in the subjects exhalation. However, the device must also offer the flexibility of collection points that may be required for different clinical applications and provide relevant feedback to the physician. The overall goal is to design a device that is easily utilised as a research and clinical tool primarily, although not exclusively, for respiratory physicians.

5.2 Storage Media

Adsorbent Materials:

Before designing the remote breath collection apparatus, selection of an appropriate storage media for the collected samples is necessary. Previous methods of analysing breath samples involved analysis using GC-MS and appropriate remote breath collection devices were developed, as discussed in Chapter 3. However, these devices used sorbent materials such as Tenax, activated charcoal, Carboxen and Carbopac for the collected breath samples. SIFT-MS can analyse samples stored on adsorbents with thermal desorption, but it is not the best approach to analyse gas samples due to difficulty with accurate quantification.

Because of measurement variation during a SIFT-MS scan of a sample, the instrument requires an exposure to the sample gasses for 4-5 seconds to obtain multiple measurements of the VOCs in the sample. Multiple measurements points provide a

statistically reliable analysis and reduce the speculation of instrument noise and scatter if only a single measurement was given. When analysing samples concentrated onto adsorbents, thermal desorption instruments rapidly inject the sample into the SIFT-MS flow-tube. The result of the injected sample from the thermal desorption instrument is a large concentration spike. By analysing the area under the concentration spike and with knowledge of the volume of sample collected, a quantitative sample concentration can be determined. However, another problem arises in that, thermal desorption instruments take approximately 15 minutes per sample, which voids the ability of SIFT-MS to analyse samples quickly and minimises throughput thus negating a major advantage of this technology.

Another type of adsorbent that was investigated was Solid Phase Micro-Extraction (SPME), which consists of a fine fibre(s) that is coated in a small volume of liquid phase coating. As gas flows past the fibre, compounds are trapped on it in a liquid phase. This method was not perused due to difficulties in quantification because of the calibration method and the heterogeneous nature of VOCs in breath samples. Also, each fibre has to be calibrated individually due to differences in manufacturing.

Tedlar Bags:

Because the SIFT-MS technique can analyse whole air samples with no preparation, a potentially more suitable storage media would be a gas-sampling bag. Gas sampling bags can be analysed by SIFT-MS without the need to modify the inlet or instrument software. There are also many different materials available for gas sampling bags. However, the most common and reliable are Tedlar bags. Hence, the remote breath collection device was designed to collect samples into Tedlar or similar bags.

For detailed documentation on the appropriate choice of storage media Neilson (2006) provides a detailed comparison of different collection media and studies the integrity of samples stored in Tedlar bags over a period of six hours. Analytes examined in the Tedlar bags included pentane, acetone, isoprene and ammonia. These analytes were selected as they are currently under investigation as markers for breath testing for clinical applications using the SIFT-MS technique.

5.3 Collection Triggers

The analytes of interest, providing physiological information about a subject are exhaled in the alveolar portion of breath. Therefore, a method is required to determine when the alveolar portion is exhaled. Most other devices reviewed employ the Haldane-Priestly tube principle. However, this method does not offer the flexibility required for the research intended of the device. The methods investigated for the detection of alveolar breath included using exhaled breath volume, exhaled CO_2 concentration, water vapour, and other VOCs commonly found in different parts of breath.

The first attempted method of detecting alveolar breath involved monitoring exhaled CO_2 to indicate exhalation of analytes that are of interest. Exhalation of alveolar breath occurs during the plateau region of the expiration curve, as shown in Figure 5.1. This approach is well validated in detecting alveolar samples based on increasing gas exchange to CO_2 that occurs only in the alveoli. Available CO_2 detectors vary in size, cost and response times. A Nellcor NPB-75 handheld capnograph (CO_2 detector) unit, (Tyco healthcare) was tested for suitability. However, the NPB-75 had a

response time of 2.5 seconds, which was too slow to obtain a CO_2 reading to decide when to collect alveolar breath samples. All detectors of the same price and size as the NPB-75 had similar response times. Ideally, the capnograph would need to have a sub-second response time to be suitable as a sensor for real time detection of alveolar breath.

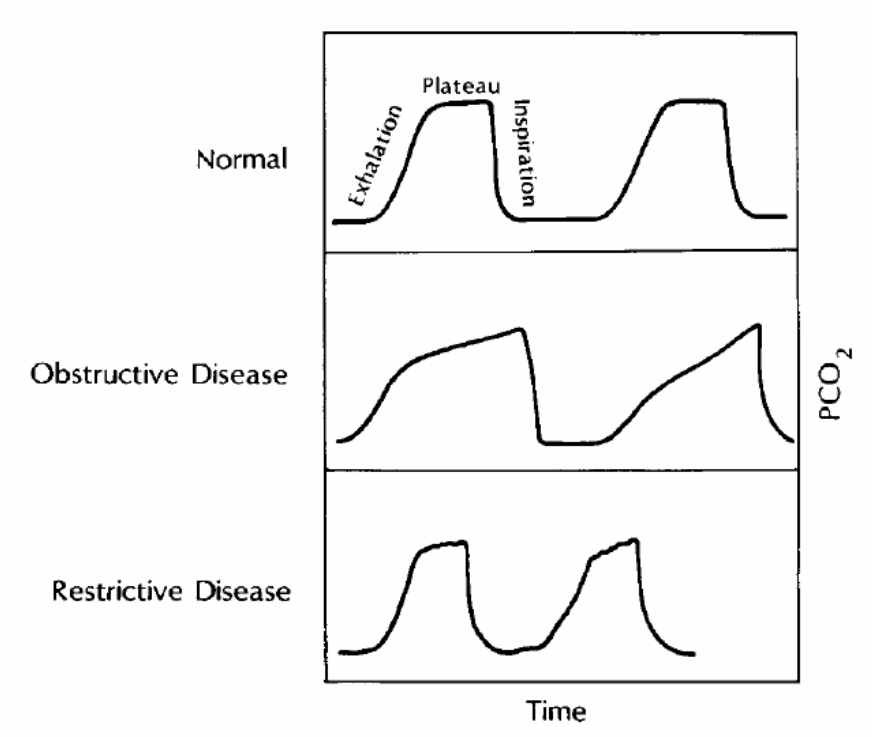


Figure 5.1 - Alveolar CO_2 patterns (Soubani, 2001)

A detector of suitable response time was sourced, but it was a desktop unit that weighed 5 kilograms and cost approximately $20,000 for unit and software, which was considered too expensive for the purposes of this study. Therefore, the prospect of using CO_2 as a suitable trigger of alveolar breath was abandoned due to cost and response times of the detector.

Other triggers suggested included other breath components, water clusters and measuring the volume of the breath exhaled. Recognition of alveolar breath is achievable by monitoring other commonly occurring analytes found in breath or looking for water clusters (masses 55 and 72 on SIFT-MS). However, detection of other analytes and water clusters would require SIFT-MS to detect in real time, which would not be appropriate for use with a remote breath collection device. Therefore, use of other analytes and water clusters as triggers of alveolar breath was not investigated further.

Because the deadspace region is approximately 30% of a tidal volume, or 3.125% of a vital capacity, collection of breath samples can be achieved by measuring a person's exhalation volume and flowrate. This approach requires a flow measurement sensor and a method of obtaining the exhaled flow in an electronic form, all of which is easily achievable.

Figure 5.2 shows two slow vital capacity manoeuvres measured with a pressure transducer and mass flow meter, and captured with a data acquisition system connected to a standard personal computer (PC). Also shown on Figure 5.2 is the rise in breath acetone, ammonia and water clusters 55 and 73 measured on a SIFT-MS instrument synchronised with the measured breath exhalations indicating that it is a suitable indicator of alveolar breath. More specifically, given 1-3 sample breaths from a subject that have a consistent exhalation profile, samples can be collected from subsequent breaths. Figure 5.3 shows the envisaged operation of the device, where the operator can choose points on a breath exhalation profile at which to collect a sample based on exhaled breath percentages and a sample exhalation profile of the subject.

Given the rapid, linear and dynamic response from the exhaled breath measurement this approach was selected as the method of alveolar breath identification.

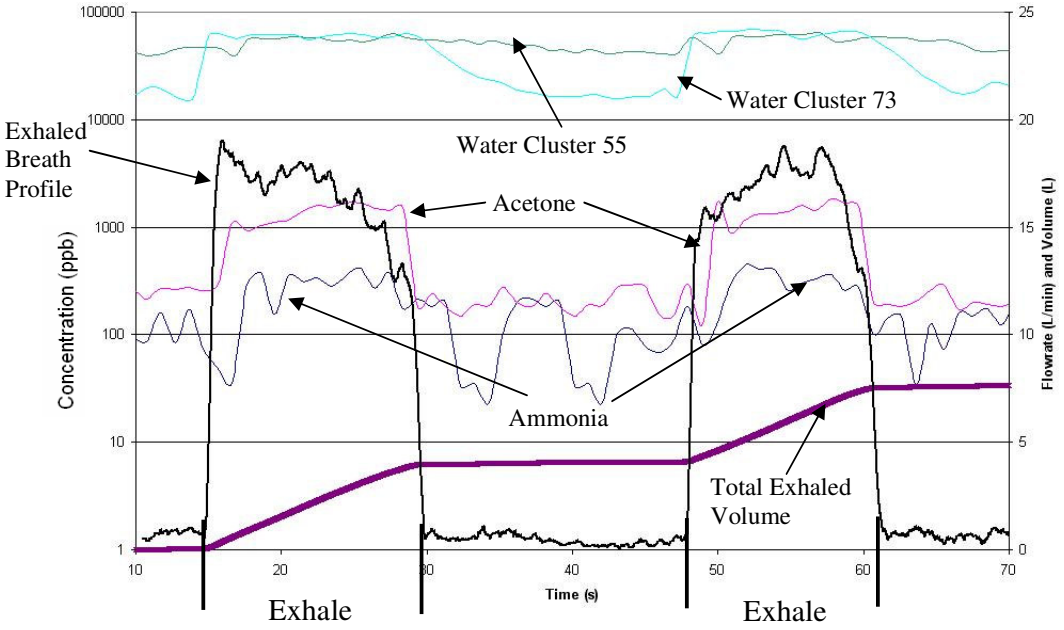


Figure 5.2 - Slow vital capacity measurement of ammonia, acetone, flow-rate and water clusters 55 & 73

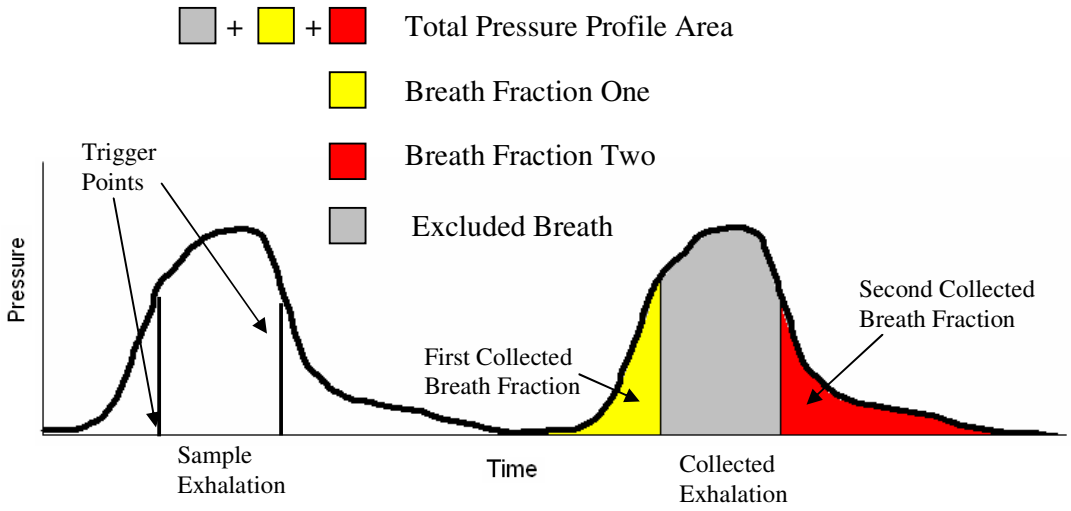


Figure 5.3 – Expected Exhalation Profiles Observed by Remote Breath Collection Device

5.4 Contamination of Samples from External VOC Sources

To prevent the contamination of collected breath samples from external VOC sources a careful selection of materials is required for components that are in direct contact with the breath flow and collected sample. Tests carried out from designing the direct breath-sampling device for SIFT-MS, showed that almost all materials emitted VOCs, especially at elevated temperatures around 100°C. Stainless steel was the most reliable material for the minimum emission of VOCs over the range of operating temperatures (20°C - 120°C). In addition, FEP was found to be a suitable material at relatively low temperatures (much lower than 100°C).

Breath is exhaled at ~37°C (body temperature) and consists of approximately 6% water vapour at 100% relative humidity when exhaled. Therefore, condensation of moisture will occur on surfaces less than 37°C. The condensed breath vapour on the surfaces of the device will trap VOCs within water droplets and may potentially release them during the collection of samples from another patient creating a new source of sample contamination. To prevent condensation occurring on the internal surfaces of the device these surfaces must be heated to a temperature above 37°C.

5.5 Operation System for Breath Collection Device

An operating system is needed to run all the components of the device, analyse information received about the exhalations, and present this data to the operator in a useful form. Two options were available for the operating system for the proposed breath collection device, use of appropriate software and hardware to run off a

standard PC with custom software and connections, or a standalone device controlled by a programmable microcontroller.

PC Operated Device:

To control the breath collection device, a PC would require software that would receive information and control the electronic hardware, as well as analogue to digital (A-D) converters and digital to analogue (D-A) converters used in data acquisition. Because A-D and D-A cards are not standard on PCs it would be complicated to implement using a laptop computer, therefore a desktop PC is required. Software packages that have the ability to communicate and command electronic components include MATLAB and Labview. However, advantages of operating the breath collection device off a PC over a microcontroller include:

- Program is easily modified to suit the application
- Information can easily be displayed graphically and pictorially
- Large storage space for software and applications.

Microcontroller Operating System:

A microcontroller is a computer chip that can be programmed to perform duties for a particular application. Different microcontrollers are available and their abilities are varied, therefore the choice of microcontroller is dependent on its application. However, most microcontrollers are designed to send and receive information from and to electronic components. Hence, they have A-D and D-A capabilities built into them. If programmed properly, microcontrollers can communicate to other devices

such as LCD displays, application programmable interfaces (API) and solenoid valves, pressure transducers. Therefore, the advantages of microcontrollers include:

- Standalone device (does not need a separate computer to run)
- Smaller power requirements
- Final product would be cheaper than PC based system with only necessary features
- No time required for boot up of device
- Smaller overall package and thus greater portability than a desktop PC.

Selected Operating System:

The chosen operating system selected for the breath collection device was the microcontroller due to its portability. Operation with a microcontroller should also be faster, as there is no waiting time while the device is booting up. In addition, operation of the device from a PC would cost much more as it would require the necessary A-D, D-A cards and software as well as the purchase of the PC. In contrast, for the breath collection device, only a relatively simple, low-cost microcontroller was required.

5.6 Chapter Summary

This chapter on remote collection has identified the need for a device that will allow investigators to study the effects of sample delivery on VOC concentration. Previously designed breath collection devices stored samples onto adsorbents that were analysed using the GC-MS technique, and are not suited to store breath samples for analysis using the SIFT-MS technique. The ideal storage media for the collected

breath samples was Tedlar bags or a similar system/approach. By using Tedlar bags, breath could be stored as a whole air sample, which is the ideal form for analysis using the SIFT-MS method.

The chosen breath fractionation method for the device involved measurement of breath through the device and sample collection based on the exhaled profile. The measurement method was selected after a short trial of examining the use of exhaled CO_2, other analytes, or water clusters, as triggers of alveolar breath. Investigation into using exhaled CO_2 as a trigger for alveolar breath showed that the available capnographs had slow response times and were very expensive. Monitoring other analytes and water clusters required the use of SIFT-MS to detect them in real time, and were abandoned as suitable triggers for collection of alveolar breath. Hence, a flow based method was developed using simple pressure and validated flow sensors.

To prevent the contamination of collected samples from external VOC sources, parts of the device that are in direct contact with the exhaled breath need to be heated above 37°C and must be made from either stainless steel or FEP. Finally, after investigation into suitable operating systems for the device, a microcontroller was chosen over a PC. Use of a microcontroller would ultimately result in the device being easily transported, cheap, fast and easy to use.

6

Remote Breath Collection - Hardware Design

6.1 Design Specifications

After identification of requirements for the remote breath collection device the device hardware design specifications were outlined. These specifications include:

1. Minimum flow resistance: for clinical reasons, the patient providing the breath sample should not experience unnecessary resistance while exhaling into the device.
2. Contamination: subjects should not be exposed to risk of infection from any previous subjects who used the device.
3. Condensation: to avoid condensation during collection, the parts in direct contact with the subjects' breath must be heated above 37°C and can only be made from stainless steel or FEP.
4. Sample size: device should be large enough to collect samples into two 1 litre Tedlar bags.
5. Simplicity: the device must easily be dismantled for sterilisation as parts of the device in direct contact with the breath flow may need to be autoclaved regularly.

6. No pump contact: collected breath sample must not pass through any pumps, as this may create another external VOC source and loss of analyte on the pump internals.
7. Automatic Sampling: the device must automatically collect the samples at the relevant collection points without the operator actively collecting them.

6.2 Tedlar™ Bag Sample Collection

The general air sampling methods document by the United States Environmental Protection Agency (2003) provides guidelines for the collection of samples into Tedlar bags. Tedlar bags should be placed into a box or chamber in which a vacuum pressure is created. The inlet fitting on the Tedlar bag should be attached to tubing that leads to the air being sampled. The vacuum pressure in the box will cause the air sample to be drawn through the tubing and into the Tedlar bag, providing a method of sample collection that does not involve the sample passing through a pump, meeting specification six above. Figure 6.1 shows a schematic representation of the standard Tedlar bag sampling method.

Other commercially available air sampling products have employed this method of sample collection, these include the Syft Sample Case by Syft Technologies Ltd, and the Vac-U-Tube and the Vac-U-Chamber by SKC. However, these commercially available devices collect samples by manual operation from the user. For example, the switches on the Syft Sample Case shown in Figure 6.2 must be turned by the user to initiate the collection of a sample. In the current research, the breath collection device

is automated using program controlled solenoid valves that are opened to collect the appropriate (typically alveolar) breath sample.

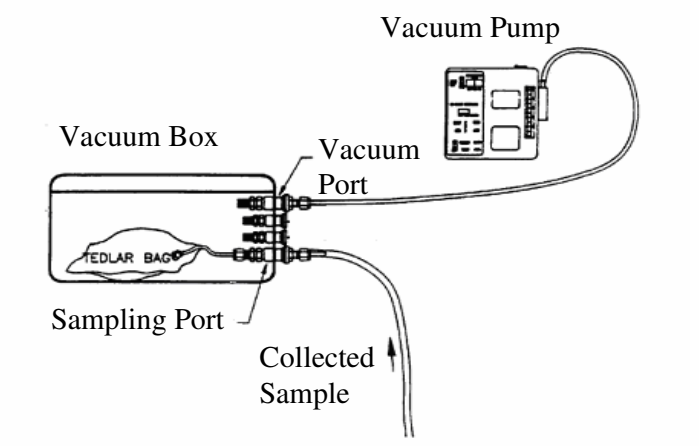


Figure 6.1 - Tedlar Bag Sampling Apparatus (United States Environmental Agency, 2003)

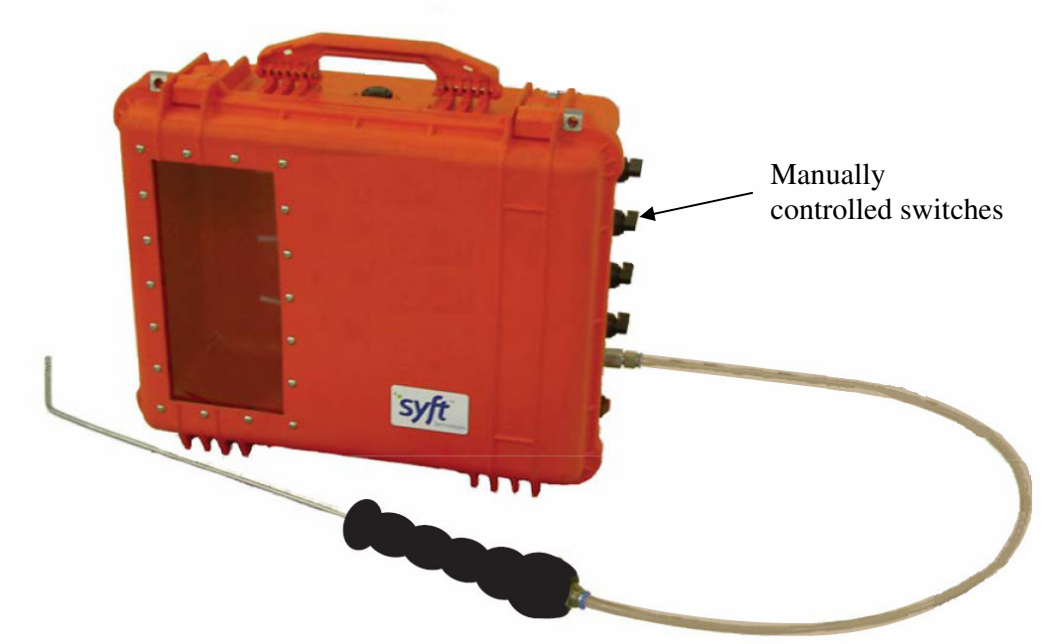


Figure 6.2 - Syft Sample Case

Vacuum Pump:

To create a partial vacuum inside the case that holds the Tedlar bags a pump is needed to remove air residing inside the case. The pump used for the Syft Sample Case is a battery operated rotary pump that is capable of evacuating sufficient air from a 50 litre container to collect samples into three 1 litre Tedlar bags. However, the pump used in the Syft Sample Case is too noisy for a device intended for use in hospitals and clinics. Therefore, a diaphragm pump was specified for the breath collection device, as they are much quieter than rotary pumps. Diaphragm pumps work by oscillating a rubber cup (the diaphragm), which causes the displacement of air. By directing the displaced air, the device can pump air from/to a desired destination.

Battery operated diaphragm pumps are commercially available and are commonly used for air sample collection onto adsorbent tubes. However, the pumps used for collecting samples on adsorbent tubes did not create the necessary vacuum required to collect breath samples into the Tedlar bags. They are also expensive, as they are designed to operate at a precise and constant flow rate.

A diaphragm pump that was originally intended to provide air for a large aquarium was used for the breath collection device. The aquarium pump operates on mains power and provides sufficient partial vacuum pressure to draw the breath sample into the Tedlar bags through tubing of 3.2 mm i.d. Figure 6.3 illustrates the test case used to determine if the aquarium pump would be capable of providing the necessary vacuum pressure required to collect the sample into the Tedlar bags.

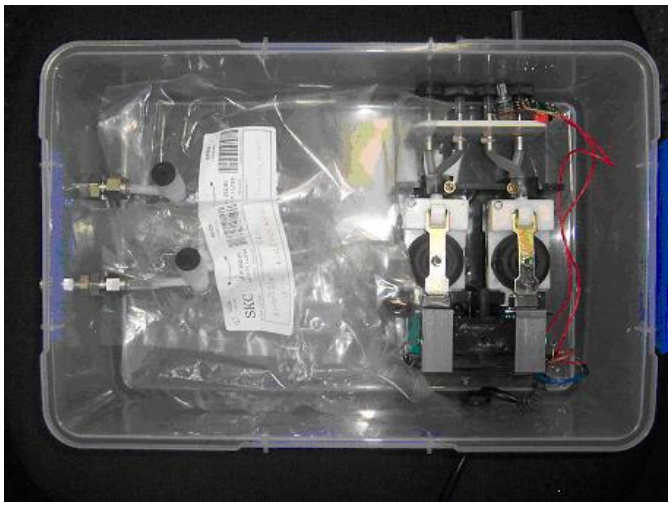


Figure 6.3 - Pump Testing for Breath Collection Device

Final Device Sample Collection Case:

The case for the breath collection device was designed with the intent to make it as easy and fast as possible to set up for sample collection. Previous sample cases utilised a box inside which the Tedlar bags were placed, with a sealable lid placed on top. On this breath collection device, the valves on the Tedlar bag valves are inserted into the attachment points, as shown in Figure 6.4, and the Perspex case is placed over them.

In operation, the edges of the Perspex case create a seal against the base of the breath collection device. The pump creates the necessary vacuum pressure inside the case to collect samples into the Tedlar bags. The pump is situated in its own individual case, directly attached to the Perspex box. As the pump evacuates the air inside the case, the Perspex is pulled down against the base and self-enforces the seal. Overall this design allows easy access to the Tedlar bags and the user can clearly see when they are full. A full set of workshop drawings is included in Appendix B1.

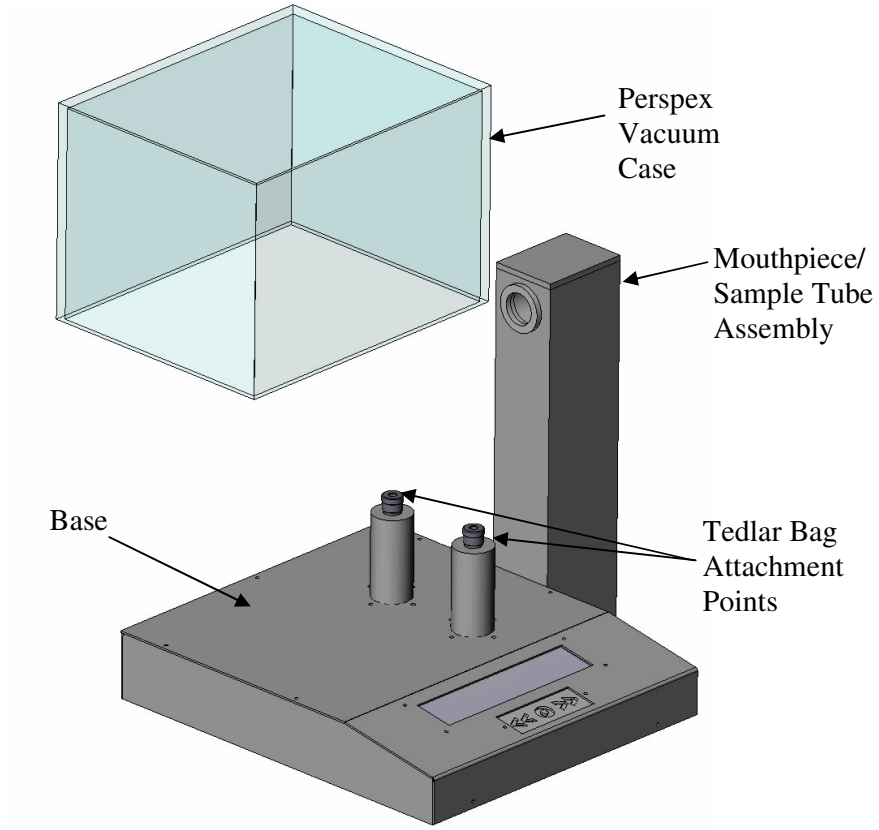


Figure 6.4 - Breath Collection Device

6.3 Flow Measurement

To measure the volume of breath exhaled into the device, several methods of flow measurement were investigated. The investigated flow measurement devices included Venturi tubes, hot wire anemometers, averaging pitot tubes and anubars. Each sensor had specific attributes that affect device operation.

Venturi Flow Tube:

The Venturi flow tube is a commonly used device for flow measurement in commercial applications. A Venturi flow tube is essentially a section of pipe with a converging section, the throat, and a diverging section at the exhaust of the pipe. By

measuring the fluid pressure at the inlet and throat, the mass flow, $\dot{m}$, rate is determined by using the known cross sectional areas and the pressure drop across the converging section of the Venturi flow tube.

$$\dot{m} = C_d A_2 \sqrt{\frac{2\rho\Delta p}{(1-\beta)}} \tag{6.1}$$

Where:

C_d	=	Discharge coefficient
A_2	=	Restriction area
ρ	=	Density of the fluid
Δp	=	Change in pressure at throat and inlet
β	=	Area ratio (restriction/inlet)
$\dot{m}$	=	Mass flow rate

Figure 6.5 shows the cross section of a venturi and the pressure taps at the inlet and throat where pressure readings are measured with a pressure transducer.

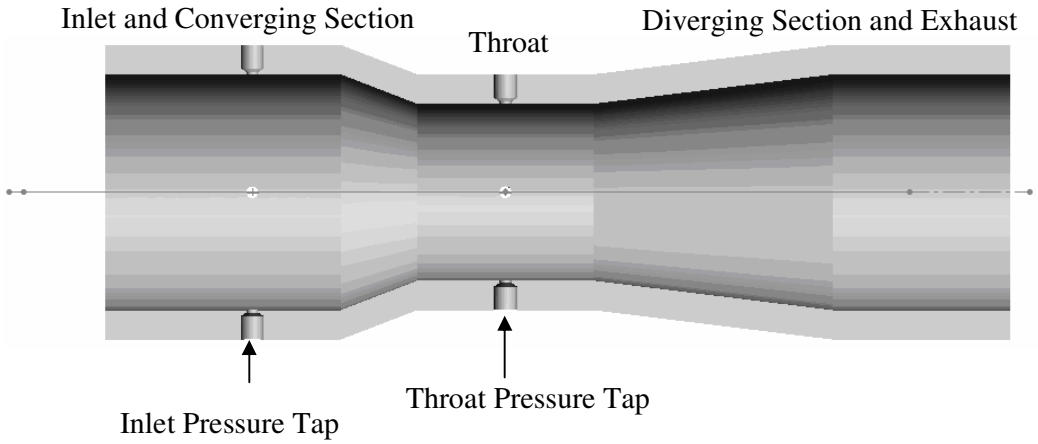


Figure 6.5 - Venturi Flow Meter Cross Section

Advantages of Venturi flow meters include:

- Robust design that is not easily damaged
- They are only sensitive to density changes, which have been found to be negligible in breath.

Disadvantages of the Venturi flow meter include:

- Precise machining is required to manufacture the Venturi tube
- Needs to be manufactured to design standards (British Standard (BS) 1042) that may make the device larger than desired for this application.

Hot-Wire Anemometers:

A hot-wire anemometer is a sensor used for measuring the flow of a gaseous fluid. It consists of a fine wire, commonly tungsten, that is heated using current and set perpendicular to the fluid flow. As the gas flows past the sensor, the wire is cooled and the temperature drop changes the resistance of the wire. The wire acts as a resistor in a Wheatstone bridge circuit, thus allowing the measurement of the resistance change as the fluid flows past the sensor. The temperature drop and thus the resistance change of the wire is proportional to the mass flow of the fluid past the sensor. Hence, a simple calibration of the hot-wire anemometer allows high resolution measurement of the fluid flow, which can be integrated to obtain volume.

Occasionally, hot-wire anemometers are placed at the throat of a Venturi flow meter and replace the need for pressure taps and the pressure transducer. However, the hot-wire anemometer does not measure the pressure drop across the converging section of the Venturi tube to calculate the fluid flow. Instead, the hot-wire anemometer uses the

converging section to ensure that the fluid has a fully developed flow profile, which prevents a noisy signal due to fluid turbulence. However, the hot-wire anemometer does not necessarily require a Venturi flow tube arrangement and can be used equally effectively inside a length of straight pipe.

Advantages of a hot-wire anemometer for this application include:

- Easy to manufacture
- Suitable calibration will give accurate measurements
- Fluid does not have to travel through pressure taps to reach sensor.

Disadvantages of the hot-wire anemometer for this application include:

- They are fragile and may easily break. Therefore, they are not suitable to be autoclaved for easy sterilisation
- Relies on the specific heat of breath, which can vary with varying temperature and humidity.

Averaging Pitot Tubes and Anubars:

Averaging pitot tubes or anubars consist of a section of pipe with two pressure taps in the centre of the fluid flow. The first pressure tap faces the fluid flow inside the pipe and the second tap is positioned behind the first, but faces in the opposite direction to the fluid flow. The purpose of the second pressure tap is to measure the stagnant pressure of the fluid flow and hence obtain a differential pressure reading between the two pressure taps to assess flow rate. This approach is shown schematically in Figure 6.6.

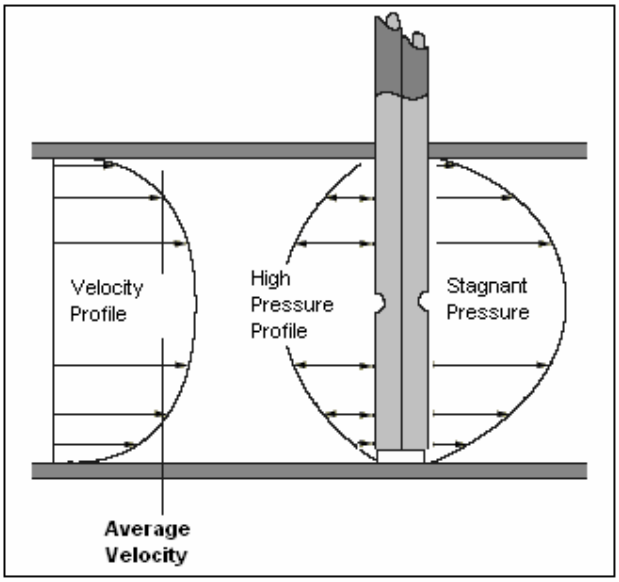


Figure 6.6 - Cross Section of an Anubar Flow Meter (OMEGA Engineering)

Averaging Pitot tubes differ from anubars as they have multiple pressure taps to obtain an average of the fluid velocity profile, as shown schematically in Figure 6.7. In contrast, Anubars only have a single pressure tap on either side, the high-pressure profile and the stagnant pressure profile. As fluid flows into the pressure tap, the pressure increase is measured electronically with a pressure transducer.

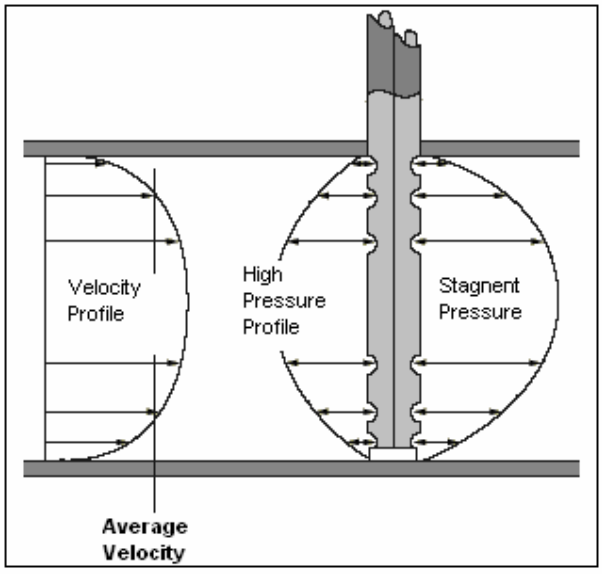


Figure 6.7 - Cross Section of an Averaging Pitot Tube (OMEGA Engineering)

Advantages of the Anubar or averaging Pitot tube for this application include:

- Easy and cheap to manufacture, both devices are pressure taps running into the centre of the sample flow
- Robust designs and can be easily sterilised in an autoclave.

Disadvantages of the Anubar or averaging Pitot tube for this application include:

- Pressure taps could potentially get blocked with breath vapour.

Selected Flow Measurement Method:

The method of flow measurement selected was the averaging Pitot tube and the anubar methods because they are easy to machine and are robust designs. Pressure readings from the taps can be read with a suitable commercially available pressure transducer connected to the pressure taps. Because both designs are relatively simple to make, prototypes of each design were manufactured for testing to determine which design would work best. Appendix B1 provides workshop drawings for both the Anubar and averaging Pitot tube flow meters available with the alveolar breath sample collection device designed in this thesis.

The averaging Pitot tube used in this device was not made from circular section pipe. Instead, it was machined from a solid hexagonal rod, which incorporated both the high pressure and stagnant pressure taps. The stagnant pressure region of the averaging Pitot tube used a single pressure tap located at the centre of the flow and is shown in Figure 6.8. Multiple pressure taps were not implemented on the stagnant pressure region of the averaging Pitot tube as they were not considered to be required for the operation of this device.

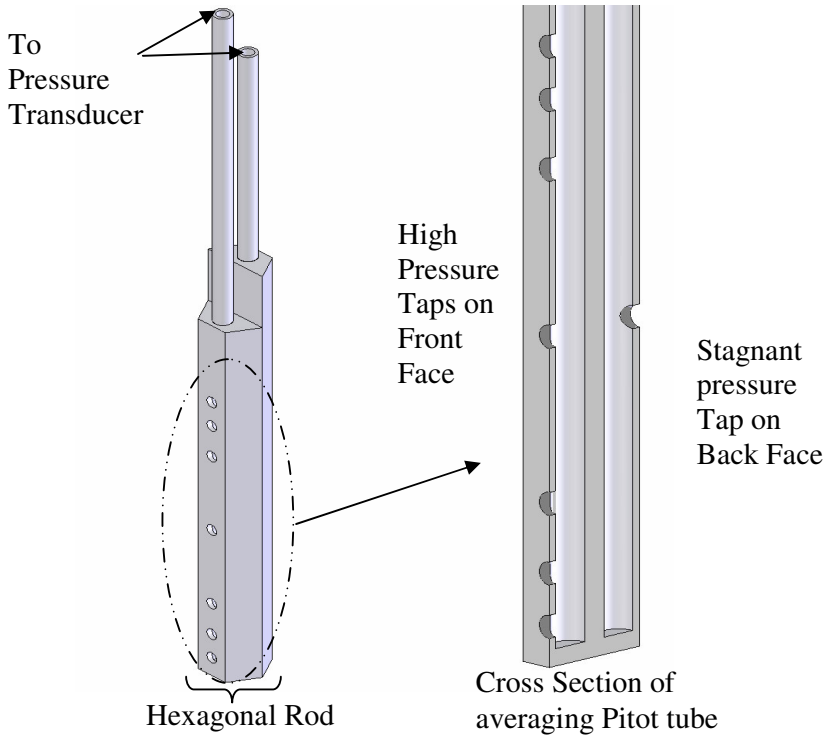


Figure 6.8 - Cross Section of Averaging Pitot tube made from Hexagonal Rod

6.4 Breath Collection Inlets

The breath collection inlets are situated next to the pitot tube/anubar and are made from 3.125 mm stainless steel tube cut at a 45° angle with the opening facing the breath flow. The 3.125 mm tubing was selected to minimise the dead space volume, but was large enough so that there would not be too much resistance to collect the sample. Figure 6.9 shows an enlarged view of a breath inlet and the placement of the two breath inlets inside the sample tube, which are placed on either side of the pitot tube/anubar sensor.

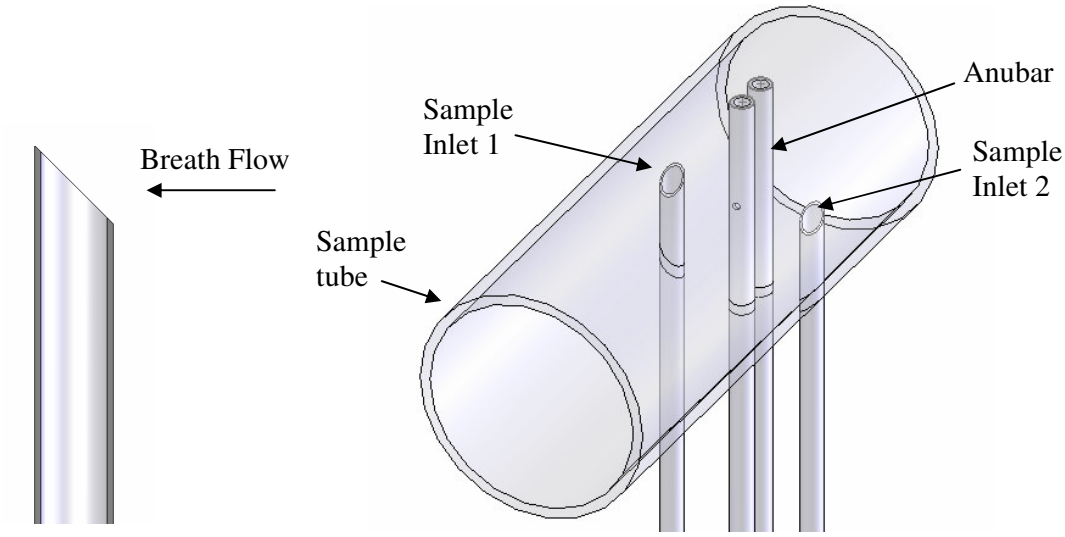


Figure 6.9 - Sample Inlets inside Sample Tube

Once outside the sample tube, the breath inlets are connected directly to the solenoid sampling valves. Stainless steel tubing is connected to the other side of the solenoid valve and is maintained at the same partial vacuum pressure as the device sampling case with Tedlar bags. When the solenoid is opened the pressure difference between the case and the opening to the breath inlet draws the sample in through the breath inlet and into the Tedlar bag. The solenoid valves are connected as close as possible to the breath inlet to minimise the deadspace volume from the solenoid to the breath collection point, thus minimising room air residing inside the tubing prior to breath collection.

6.5 Solenoid Valves

The solenoid valves used on the device were 2-port VDW series solenoids from SMC Christchurch, New Zealand. They were chosen for their small size and low power requirements. Two different seal materials are available for the series VDW solenoids,

a fluoroelastomer material called Viton from DuPont (FKM seals) and a nitrile rubber (NBR). The NBR seals are more commonly available, but FKM seals were used on the device as they have a greater chemical resistance, are generally a more chemically inert material, hence avoiding a contamination source of VOCs. Though FKM seals are chemically inert, Viton has a tendency to adsorb acetone. Therefore, regular replacement of solenoids will be required to limit the introduction of acetone from the valve seals and minimise sample contamination from external sources.

To match the size of the stainless steel tubing, the orifice size of the solenoids were 3.2 mm in diameter. This size does not impose an added resistance for the device to collect the sample. Solenoids in the VDW series vary in voltage requirements, and the valves selected operate off 12 volts (V) direct current (DC), which is provided in a portable device from a power supply.

6.6 Mouthpieces

There are a range of mouthpieces currently used in clinical applications for breath testing. However, all of the different variations of mouthpieces have essentially the same diameter. Therefore, the inlet for the patient to use the device was designed to attach with currently used disposable mouthpieces. The majority of available mouthpieces are made from plastic that have incorporated biological filters to prevent the possible spread of infection from patient to patient, although some are disposable cardboard tubes without filters. Figure 6.10 shows the spacer that holds the sample tube inside its cover and is the attachment point for the disposable mouthpiece. This

spacer is made from FEP and machined so the mouthpiece attachment is 30 mm in diameter.

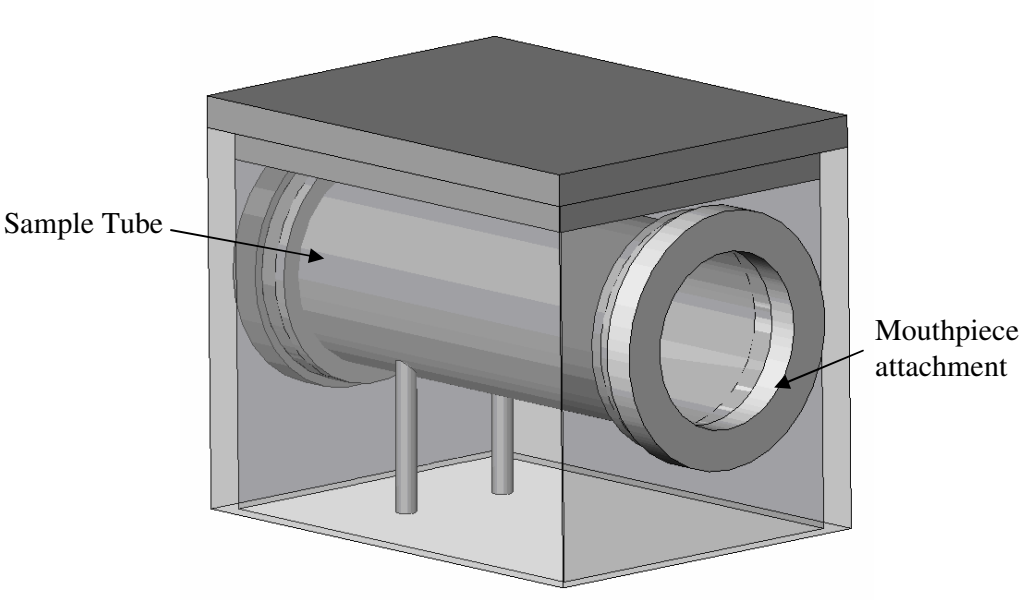


Figure 6.10 - Attachment Point for Mouthpiece

6.7 Chapter Summary

The breath collection apparatus for the SIFT-MS technique has been designed to collect samples into 1 litre Tedlar bags in accordance with standard air sampling guidelines. Two Tedlar bags are inserted into the device and a Perspex case placed over them. A partial vacuum pressure is created inside the Perspex case and is evacuated by a diaphragm pump, selected for its high pumping capacity and low operating noise.

Collection of breath samples is triggered by the exhaled breath profile that is measured by either an anubar flow meter or an averaging pitot tube. Both designs measure the flow inside the device by pressure taps connected to a pressure transducer. The device has incorporates the use of currently available disposable mouthpieces, manufactured to a standard diameter. Disposable mouthpieces with biological filters prevent subjects' exposure to infection risk. Solenoids are used to initiate sample collection and are placed as close as possible to the breath collection inlet to minimise the amount of deadspace volume collected.

7

Remote Breath Collection - Software Design and Electronics

This Chapter discusses the design and development of the software and electronic hardware for the breath collection device. It also discusses the detailed software architecture developed for the device to provide the collection flexibility required to study the many different physiological parameters of breath delivery that may affect VOC concentration.

7.1 Design Specifications

The design specifications for the device software and electronic hardware include:

1. Breath collection flexibility: The software should allow the operator to collect a sample at any point during a breath exhalation
2. Patient training: Exhalation from the subject must be as similar as possible between breaths. Therefore, some patient training may be required before collection of the sample
3. Consistency: Electronic components, such as the pressure transducer, should provide consistent measurements and should not display any signs of drifting signal

4. Intuitive: Software should be simple and easy to use, and minimal technician training should be required to operate the device.

7.2 Pressure Transducer Selection

Selection of an appropriate pressure transducer was an involved process, as there are many sensors available with varying capabilities and costs. The first pressure transducer, used for the proof of concept device, was a differential sensor manufactured by Motorola with a range of 0 – 2.5 kilopascals (kPa). However, the Motorola pressure transducer was one of the cheaper sensors available and did not include temperature compensation. It also required high levels of amplification and a frequency filter to eliminate noise due to mains power interference.

Results from an experimental rig showed the different types of breathing that were effectively sensed by the Motorola sensor. Different breathing styles indicate the effective limits of the pressure transducer range. In this case, it could register light and shallow breathing, as well as sharp short breaths. These breath profiles are shown in Figure 7.1.

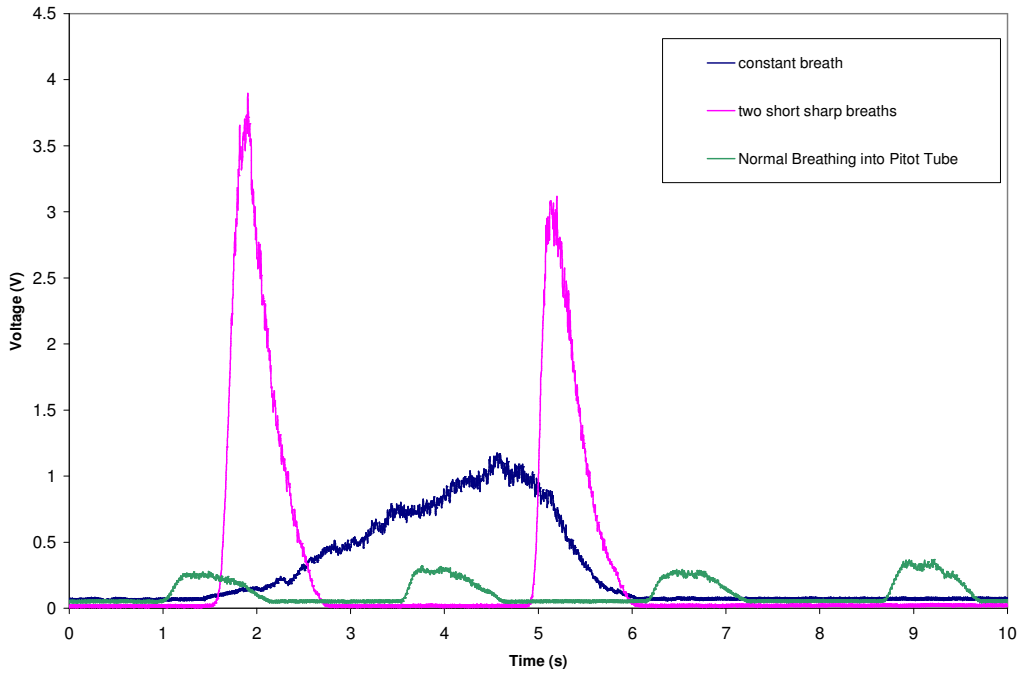


Figure 7.1 - Breath Profiles with Motorola Pressure Transducer

The Motorola pressure transducer was later replaced with a differential pressure transducer manufactured by Honeywell, which included some level of thermal compensation. However, the Honeywell pressure transducer posed other problems similar to the Motorola sensor. Amplification was still required with the Honeywell pressure transducer, but its signal did not require filtering. Amplification of the pressure transducer could easily be implemented by the microcontroller used to run the device. Although the Honeywell sensor had thermal compensation, sudden changes in temperature affected its zero pressure voltage reading. To prevent the resultant drift in voltage due to thermal gradients, the pressure transducer needed to be well insulated inside the device and zero voltage readings were taken before each subject providing a breath sample.

Another suitable pressure transducer was the Suresense™ pressure transducer by Honeywell with a range of 0 – 1.25 kPa. The Suresense™ pressure transducer is self amplifying, has a high level of thermal compensation and is therefore not affected by sudden temperature changes. On start-up, the zero pressure voltages of the Suresense™ sensor varied by 0.005 V, which is very minimal. This resolution allowed very fine recognition of an exhalation inside the sample tube of the breath collection device. Hence, the final choice of pressure transducer was the Suresense™ by Honeywell.

7.3 User Interface Electronics

LCD Screen:

The LCD panel on the breath collection device is a 40 x 4 character display. The technician is led through a series of menus displayed on the LCD panel to set the collection points on an exhaled breath profile. The LCD panel is mounted underneath a sheet of Perspex to prevent damage and allow it to be easily wiped down.

User Controls:

Several different user interface options were considered for the breath collection device. These options included individual buttons, rotary sliders and linear sliders. All of these options used capacitive touch sensing to activate the input.

Capacitive touch sense buttons are essentially a pad with a capacitance across it that is mounted underneath a sheet of plastic. When the user places their finger on top of the plastic, the capacitance across the pad changes and an integrated circuit (IC) registers

this capacitance change as the button being pressed. In the case of the rotary and linear sliders, the IC returns a number between 1 and 256 that represents a position on the slider. Because the capacitive touch sense buttons are not mechanical, they are more robust and can be wiped down without damage while the device is turned off.

The rotary slider was initially implemented into the device controls. However, using sliders was not very intuitive and involved relatively complex program code. After testing the interface with a number of individuals, it was apparent that the sliders were difficult to adjust to a specific desired number and that adjustment in increments using individual buttons was simpler. The sliders could be programmed to act as individual buttons, as well as sliders, but it is cheaper to implement separate buttons on the device. In addition the code required is significantly simpler.

The breath collection device is controlled by three capacitive touch sense buttons that are used to navigate through the different menus and adjust variables displayed on the LCD panel. These buttons send digital signals to the microcontroller. The three buttons on the device are labelled left, right, and enter. The left and right buttons are used to navigate within the menus and adjust variables. The enter button is used to select variables to change and move between menus during operation.

7.4 Software Proof of Concept

Before programming the microcontroller, a proof of concept program was coded using dSpace™, a real-time prototype module, and Simulink™, both of which run on a standard PC. The proof of concept program used a recorded raw pressure transducer

voltage as a zero pressure level. Once the voltage from the pressure transducer increased over a specified threshold above the zero voltage level, the program registered a breath being exhaled. Similarly, when the voltage fell back below the threshold the exhalation was complete.

The program measured three sample breaths into the device that were then used to estimate the exhalation profile. After the exhalation profile was obtained, subsequent breaths were fractioned and collected into the proof of concept device shown in Figure 6.3. This proof of concept program was to confirm that breaths could be fractioned based on percentages of an exhalation profile and that the 3.125 mm tubing used for the device was sufficient to prevent creating an added restriction in sample collection.

7.5 Microcontroller – Programmable System on a Chip (PSoC)

The microcontroller used on the final prototype breath collection device was a PSoC CY8C27443-24PXI, manufactured by Cypress Microsystems Ltd. The PSoC microcontroller was chosen because it is relatively cheap and comes with a range of built-in user routines used to drive other components, such as the LCD display. These built-in routines make interface development significantly simpler. The CY8C27443-24PXI has 3 ports, with 8 pins on each port.

Analog and Digital Components:

The family of products that the CY8C27443-24PXI comes from is called the PSoC mixed-signal array. These products compose a line of chips that integrate microcontrollers with the analog and digital components that would typically surround it in an embedded system. More specifically, the microcontroller used on the breath collection device had room for 12 analog and 8 digital components.

PSoC Designer™ is a development program used to program microcontrollers from Cypress Microsystems. The development program provides a graphical-user-interface (GUI) approach to configure the PSoC and select the digital and analog components required. Figure 7.2 shows a screen shot of the GUI used for component selection/placement in the PSoC Designer™.

Analog components used on the breath collection device included

- Programmable gain amplifier (PGA) that takes the signal from the pressure transducer and amplifies it as required. The Suresense™ pressure transducer employed did not require amplification, as it was self amplified. Hence, the gain on the PGA was set to 1 in this case.

- Delta-Sigma-11 (DELSIG11) A-D converter that converts the raw voltage signal from the pressure transducer into an 11-bit binary number, and then stores it as a 16-bit variable. The DELSIG11 uses the successive approximation technique to convert the analog voltage of the pressure transducer to a digital number. To obtain a true value of the input voltage the DELSIG11 reads and compares the signal 976 times to obtain an accurate

figure for the sensor voltage. The DELSIG11 also has a digital block that is used for the timing of the A-D converter, as well as its analog block, as shown in Figure 7.2.

Digital components used in the breath collection device included

- A 16-bit timer that is used to prompt the pressure transducer sensor reading from the DELLSIG11. The timer ensures that the sample time between each reading is the same for both the average breath profile measurement and collection of breath fractionations. A timer is used because both steps in the breath collection device could take different times depending on the length of code. The breath collection device takes pressure measurements during an exhalation at 100 Hz. A 16-bit timer occupies 2 of the 12 available spaces for analog devices.

- A 16-bit pulse-width-modulator (PWM) was used to control a piezo-electric speaker. By sending a series of pulses to the speaker a sound was generated when a button was pressed on the device. By changing the pulse timing on the PWM, different tones were produced from the speaker. Producing sound enhanced user feedback when operating the device, allowing the person to know when they had pushed a button, as well as seeing a change on the LCD display.

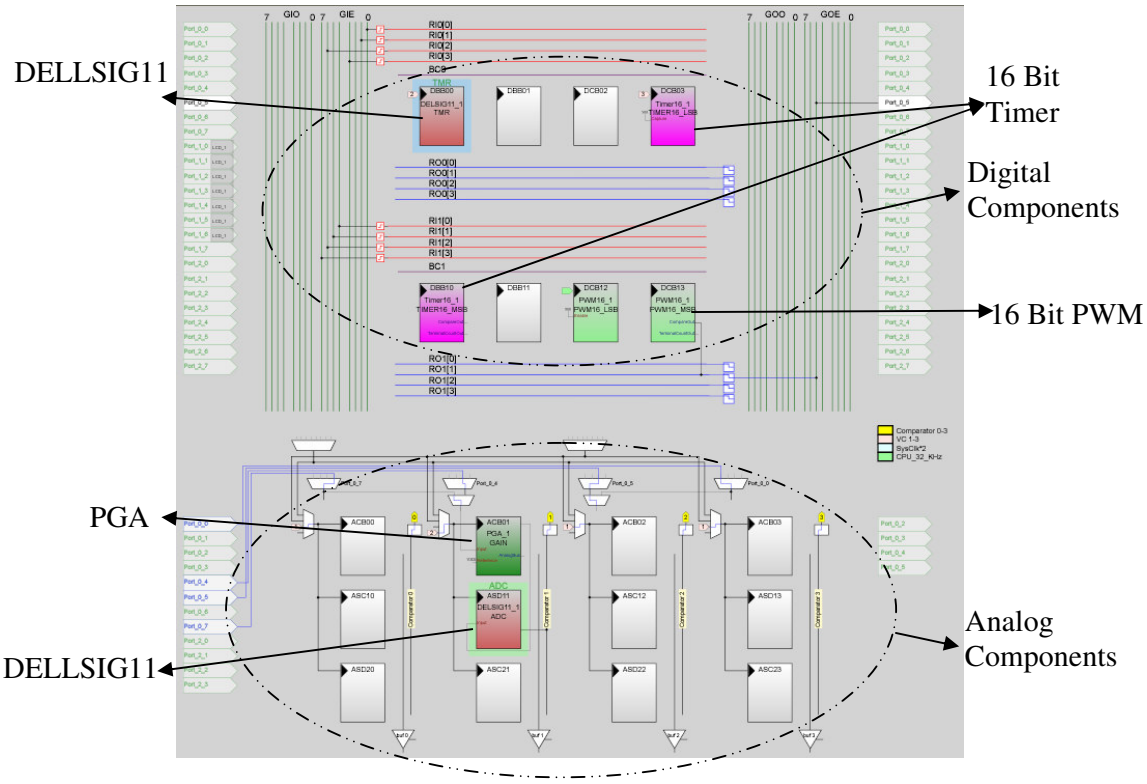


Figure 7.2- Components used on the Breath Collection Device

Programming Language:

Two programming languages are available using the PSoC Designer™, C and Assembly. The breath collection device was programmed using C because of its ability to calculate floating point arithmetic. Writing code in C creates programs that are larger and slower than programs written in Assembly. However, for the breath collection device, program speed was not a major issue given the long time periods of a breath sample compared to PSoC chip clock speeds. In addition, the CY8C27443-24PXI has 2 Kb of read only memory (ROM), so program size was also not an issue in this case. The user routines, generated by the PSoC developers program, used to drive the LCD panel and other devices, were written in assembly.

Port and Pin Selection on a PSoC:

The CY8C27443-24PXI has 28 pins. Of these 28 pins, 24 are assigned to the 3 ports and the remaining 4 are used for power. Eight pins are assigned to each port on the PSoC and each has its own limited capabilities. Therefore, careful selection of pins/ports is required for the components that run off the PSoC. For example, the signal from the pressure transducer is analog and can only be connected to pins 1 – 4. Figure 7.3 shows the pins on the PSoC chip, their respective allocations, and their port numbers.

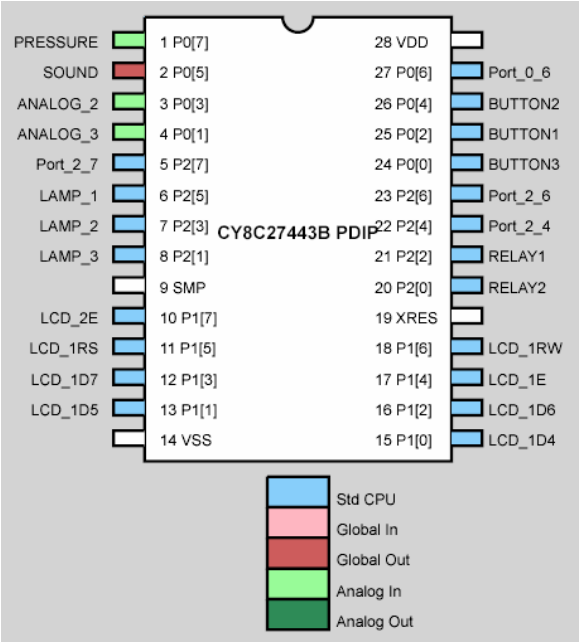


Figure 7.3 - Pin Selection on PSoC

To read or change the state of pins on the PSoC, the program must call the appropriate port and pin number. For example, a solenoid valve is connected to pin 21 which is pin 2 on port 2. Therefore, to open that solenoid, the output of port 2 pin 2 must be set to 1. The different pins on the ports are called by numbers in hexadecimal. Therefore,

the pin is read by logical operations on the respective port. To open the solenoid valve the following command line is issued:

PRT2DR = PRT2DR OR 0x04

In the command line, the number 0x04 represents pin 2 on a port and by performing a logical OR operation all the pins on port 2 remain unchanged apart from pin 2. This command line sets pin 2 of port 2 to be switched high, thus activating the solenoid valve connected to pin 21. Similarly, to close the solenoid the same line of code is implemented, except the OR is replaced with a NOT-AND logical operator. Table 7.1 lists all the pins on the PSoC, the components they are connected to on the breath collection device, and their respective port/pin allocations.

Pin	Label	Function	Input/Output	Port	Port Pin Number
1	PRESSURE	Pressure Transducer Signal Input	Input	0	7
2	SOUND	Pizo Speaker	Output	0	5
3	ANALOG_2	Additional Analog Input Signal	Input	0	3
4	ANALOG_3	Additional Analog Input Signal	Input	0	1
5	Port_2_7			2	7
6	LAMP_1	LED - Output	Output	2	5
7	LAMP_2	LED - Output	Output	2	3
8	LAMP_3	LED - Output	Output	2	1
9		Gound Connection		N/A	N/A
10	LCD_2E	LCD Bottom two rows	Output	1	7
11	LCD_1RS	LCD Pin	Output	1	5
12	LCD_1D7	LCD Pin	Output	1	3
13	LCD_1D5	LCD Pin	Output	1	1
14		VSS - Ground connection		N/A	N/A
15	LCD_1D4	LCD Pin	Output	1	0
16	LCD_1D6	LCD Pin	Output	1	2
17	LCD_1E	LCD Top two rows	Output	1	4
18	LCD_1RW	LCD Pin	Output	1	6
19		XRES - Input Line		N/A	N/A
20	RELAY_2	Solenoid Valve Output	Output	2	0
21	RELAY_1	Solenoid Valve Output	Output	2	1
22	Port_2_4			2	4
23	Port_2_6			2	6
24	BUTTON3	Right - Input	Input	0	0
25	BUTTON2	Enter - Input	Input	0	2
26	BUTTON1	Left - Input	Input	0	4
27	Port_0_6			0	6
28		VDD - Supply voltage		N/A	N/A

Table 7.1 – Pin Allocations on the PSoC Microcontroller

Modification of LCD User Routine:

The LCD user routine written by the PSoC developer program can only operate displays that are 80 characters in size. The LCD panel used on the breath collection device displays 160 characters. Therefore, modification of the generated user routine was required, or a suitable LCD routine had to be written.

The LCD panel used on the breath collection device has two pin inputs that drive the top and bottom half of the display. Therefore, the LCD user routine was modified to send the command to the LCD via a second pin on the PSoC to write information to the bottom half of the display. Before commanding the LCD panel, appropriate selection of output pin is required, depending on which half of the display the message is written to. The two pins that control the top and bottom halves on the LCD screen are labelled in Table 7.1 as LCD_1E (top two rows of LCD) and LCD_2E (bottom two rows of LCD). The LCD routine was modified so that a variable (lcd_e) in the code was used to assign the output pin for the LCD. The variable, lcd_e, was set to either 0x10 or 0x80 to write to the top or bottom half of the display respectively. The modified LCD module is included in Appendix B3.

Power Supply and Circuit Board:

The CY8C27443-24PXI operates off 5 VDC, but some of the components, such as the solenoid valves and the back light for the LCD panel, required a 12 VDC supply of power. Therefore, the power supply for the breath collection device provided 12 V power to the system. The circuit board that housed the PSoC dropped the voltage from the power supply down to 5 VDC and regulated the output power to the chip. The components that required 12 VDC received power from a separate power rail on the

circuit board and received the operation signal from the PSoC. The design and manufacture of the circuit board for the PSoC was completed by Rodney Elliott in the Applied Mechanics Laboratory, Department of Mechanical Engineering, University of Canterbury.

7.6 Software Architecture

The software written for the breath collection device was structured using a number of sequences, each with its own state-based system. Upon start-up, the program displays a welcome message that is shown for several seconds. It then enters the first sequence of the program. Figure 7.4 shows the five sequences that make up the breath collection device program and the steps taken from each sequence throughout the program.

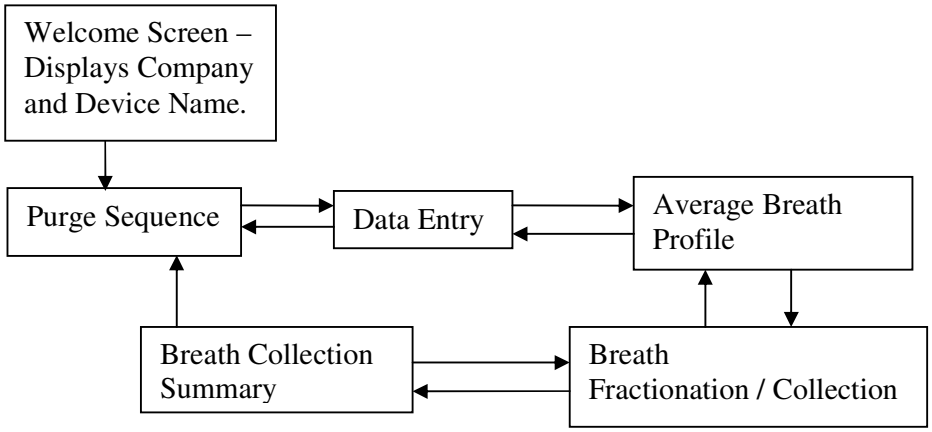


Figure 7.4 - Program Sequences for Breath Collection Device

Figures 7.5 to 7.14, at the end of this section of the chapter, show state diagrams of the five different sequences and their respective display on the LCD panel. The LCD

panel displays have a small arrow (→) in the bottom right corner, this arrow moves around the screen to indicate the variable/command that will be selected when the enter button is pressed. When a variable is selected to be changed, the selected variable is identified by a small square.

In the bottom left and right corners of the screens are the symbols, << and >>, which when selected lead to the previous or next screen. The sequences in the breath collection device software operate on a state-based system. Each sequence has a number of states and each one performs a specific task. The sequence figures show the input required for movement between states. These inputs are the three buttons on the device panel, left, right and enter.

Purge Sequence:

The purge sequence is where the operator can purge room air through the valves and pipes of the breath collection device in between subjects. In this sequence of the program, the operator can choose the length of time that the system is purged. Purging of the system opens both solenoid valves and is performed without the Tedlar bags inside the collection case. Figure 7.5 shows a state diagram and Figure 7.6 shows the display on the LCD panel for the purge sequence.

Data Entry Sequence:

After the valves and plumbing of the breath collection device have been purged and the operator is ready to continue with the breath collection, the device requires information on where to fraction the breath profile. In the data entry sequence, the two breath fraction percentages are entered into the device's random access memory

(RAM). As well as the breath fractionation points, a tolerance is included. The tolerance specifies the acceptable breath-to-breath variation. This tolerance is also used for patient training when the device is measuring the average breath profile. Figure 7.8 shows a screen shot of the data entry sequence displayed on the breath collection device.

Figure 7.7 shows the state diagram for the data entry sequence that shows the button inputs required by the user to adjust the values used for breath fractionation. The sequences used on the breath collection device program have two states per variable/command: the parent state and the action state. Once in a parent state the enter button causes the program to enter the action state where a particular variable is changed or the program changes sequences.

Obtain Average Breath Sequence:

After the breath fractionation data and tolerance is entered into the breath collection device, an average breath profile is required before fractionation of the subject's breath. In the average breath sequence, the subject must exhale three consecutive breaths that do not differ by the tolerance specified by the clinician. During this sequence the subject receives some training before providing the exhaled breaths to be fractioned. From this sequence the operator may return back to the data entry states to change variables if desired, but cannot proceed to the breath fractionation sequence until the three average breath exhalations are obtained by the device. Figure 7.9 and Figure 7.10 show the states to obtain the average breath profile and respective screen display.

Breath Collection Sequence:

After the average breath profile is obtained, the device is ready to start collecting fractions of breath from the subject. Figure 7.11 shows the breath fractionation sequence in the device software. To help the subject keep their breaths within the breath tolerance profile, LED lights are used that show when the subject has exhaled 20%, 50% and 80% of their average breath profile. In the breath collection and average breath sequences, the program changes state by detecting an exhalation into the device. It also performs operations, such as operating the solenoid valves and LEDs, during this period.

Collection Summary:

During the breath collection sequence, the software keeps track of the number of breaths that the subject exhales into the device as well as the number of breaths that fall outside the tolerance. After the fractionation and collection of breaths from the subject, the clinician selects next on the sequence menu when the Tedlar bags are full and the device displays a summary of the collection. By viewing the number of breaths out of tolerance, the operator can decide if the collected breath samples in the Tedlar bags are suitable for testing. If too many of the collected breaths are outside the specified tolerance, the operator may wish to place new Tedlar bags and restart the sample collection. Figure 7.13 and Figure 7.14 show the breath collection summary sequence and screen display respectively. A copy of the software code, written in C, is included in Appendix B2.

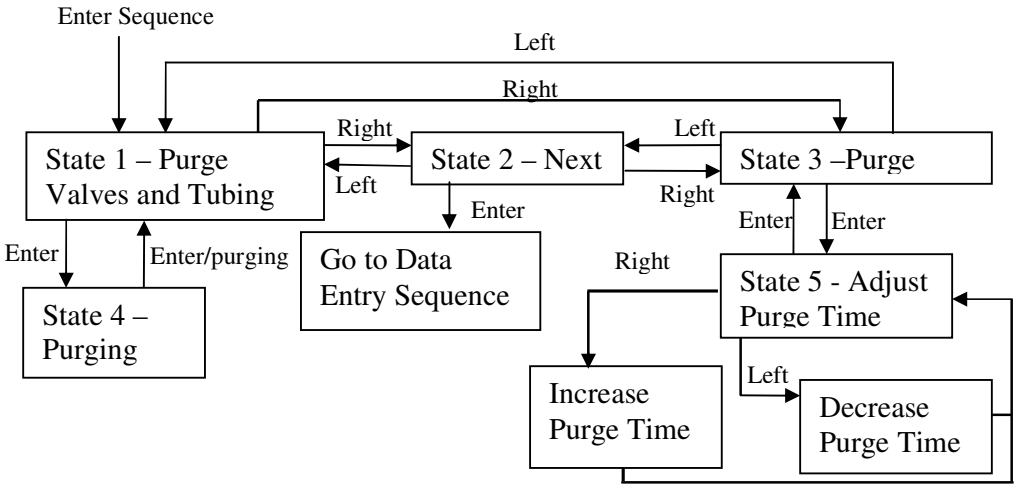


Figure 7.5 - Purge Sequence

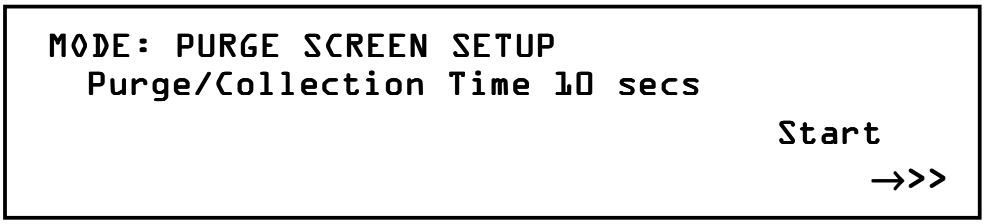


Figure 7.6 - Purge Screen

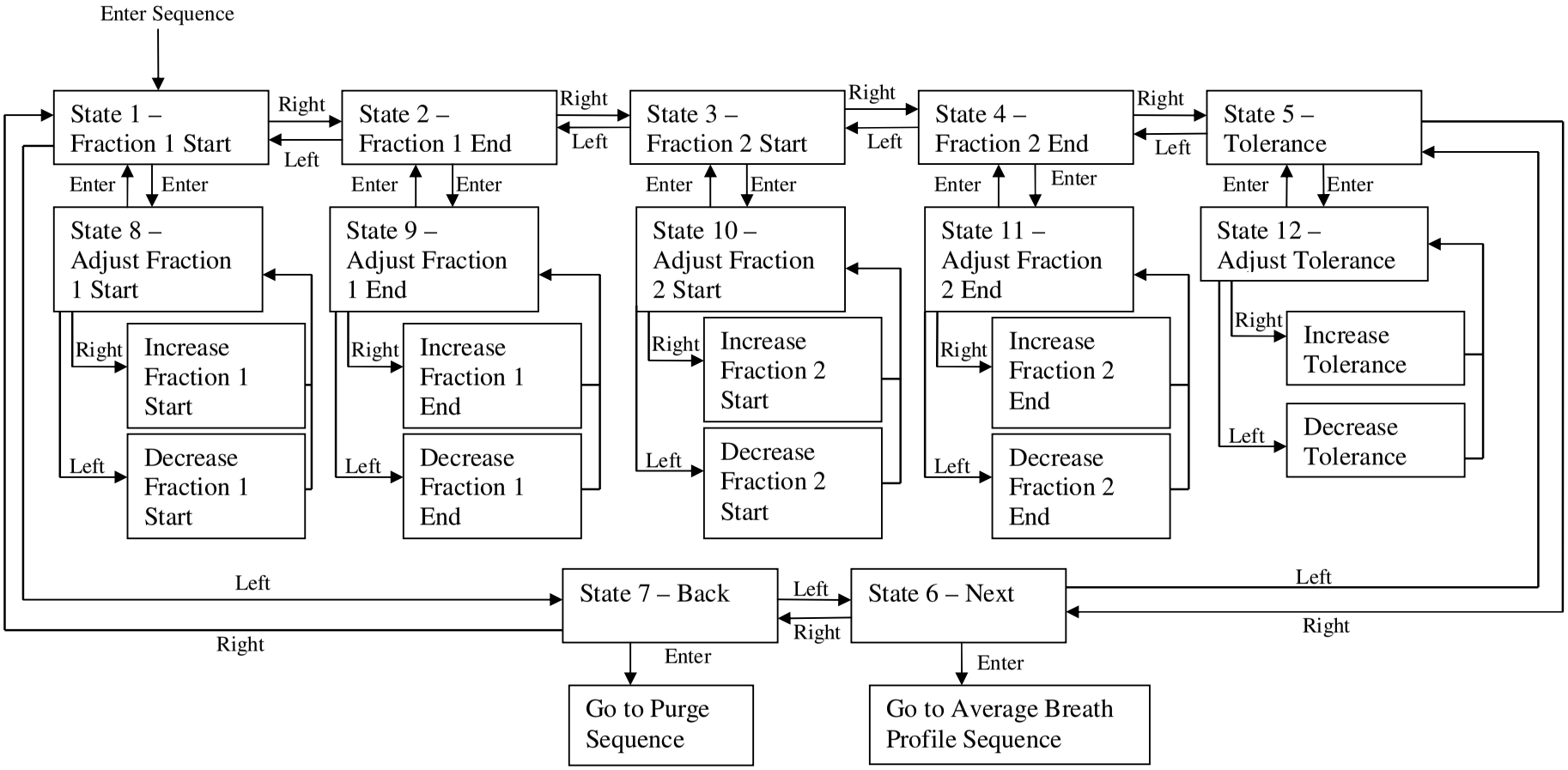


Figure 7.7 - Data Entry Sequence

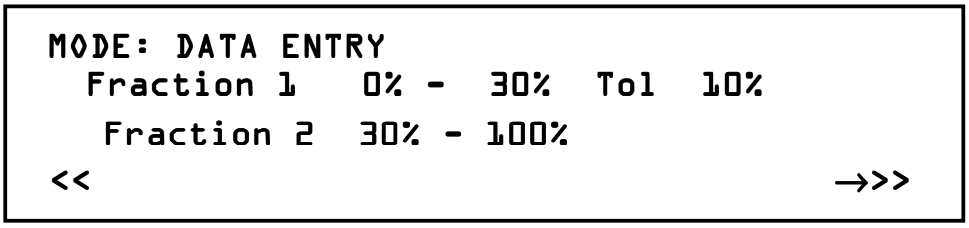


Figure 7.8 - Data Entry Screen

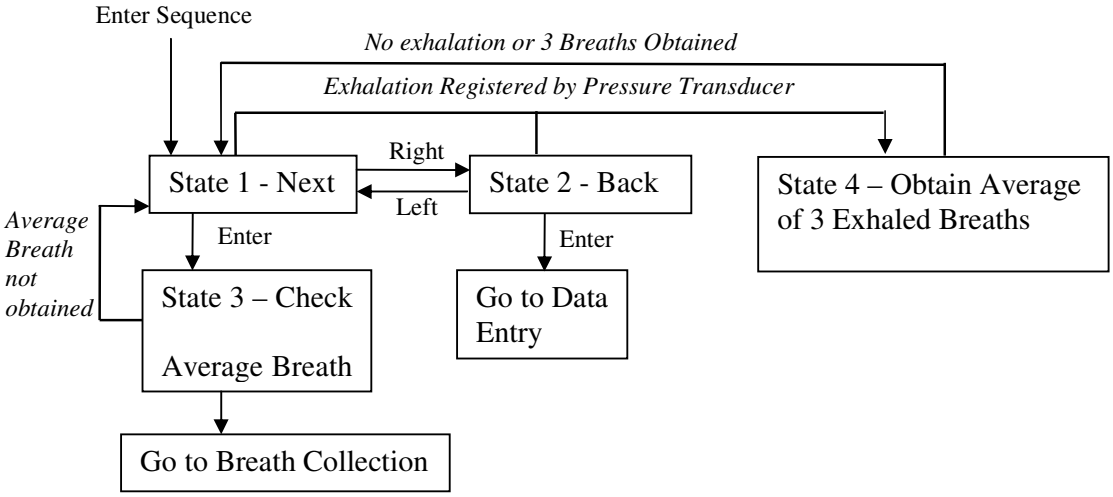


Figure 7.9 - Average Breath Measurement Sequence

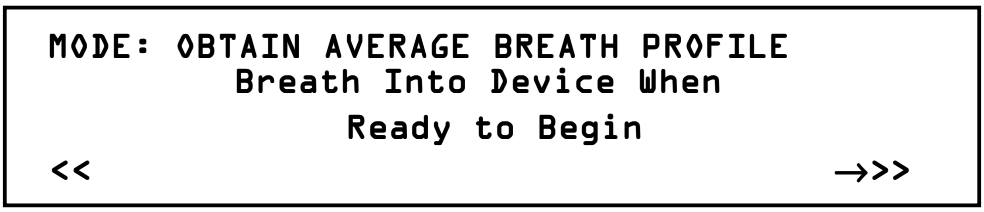


Figure 7.10 - Obtain Average Breath Profile Screen

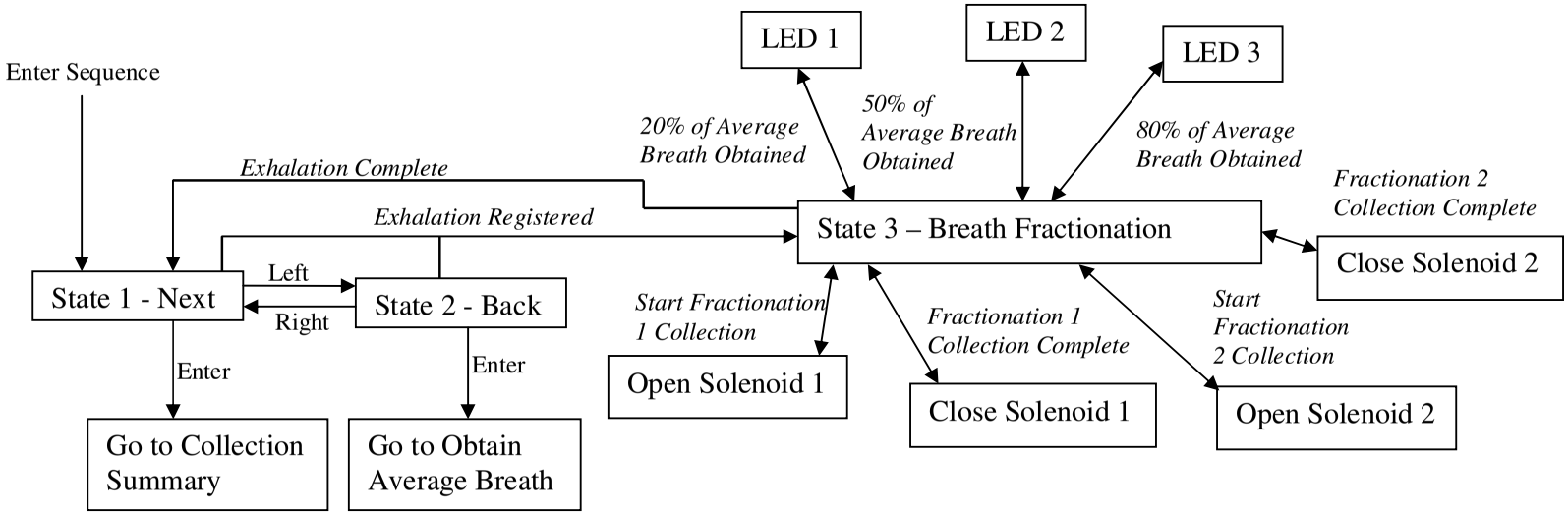


Figure 7.11 - Breath Fractionation Sequence

```
MODE: BREATH FRACTIONATION
  Breath Into Device To Collect Sample
         Select >> When Finished
<<                                  →>>
```

Figure 7.12 - Breath Fractionation Screen

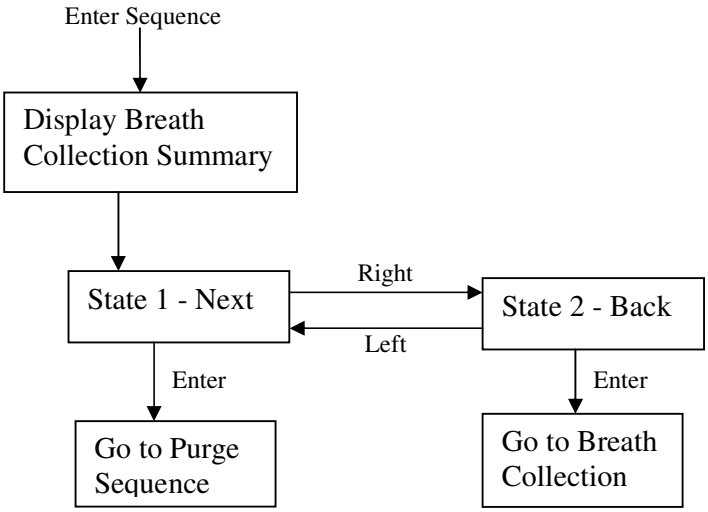


Figure 7.13 - Breath Fractionation Summary

```
MODE: COLLECTION SUMMARY
  Breaths Collected  7    Tol 10%
   Breaths out of Tol 2
<<                 → Return to Purge Mode
```

Figure 7.14 - Collection Summary Screen

7.7 Chapter Summary

The initial stages of the software and electronic hardware design involved selection and testing of appropriate pressure transducers. Three pressure transducers were tested; the Suresense™ was selected because it was thermally stable and self amplified, which helped reduce associated signal noise. Other electronics included the selection of user interface components, including the LCD panel and device controls. The controls implemented on the breath collection device use capacitive touch sense buttons because they are robust and can be easily wiped down.

A software proof of concept was created using Simulink and dSpace™, which showed that breath could be fractioned on percentages of an example exhaled breath profile. The microcontroller used for the final breath collection prototype device was a CY8C27443-24PXI PSoC from Cypress Microsystems. A PSoC was used as they can integrate analog and digital components in an embedded system.

The software architecture implemented on the breath collection device involved five individual sequences, each operating with its own finite state system. The five sequences in the breath collection device software are the purging, data entry, average breath, breath fractionation and breath collection summary sequences. The state-based system allows a program to remain idle in a particular state within a function until an input is received by the user.

To provide patient training, a tolerance is applied that specifies how much breath profiles are allowed to differ by during the measurement of an average breath profile. During the collection of breath samples, LEDs inform the subject when they have

exhaled 20, 50 and 80% of their average breath profile to aid the subject's exhalation consistency. When the breath collection is complete the device displays the number of breaths fractionated, the specified tolerance and the number of breaths collected that were outside the tolerance, aiding the operator in deciding whether to keep the sample for testing.

8

Testing and Validation of the Remote Breath Collection Device

This Chapter presents the test procedures used to validate the remote breath collection device, and the results obtained. In particular, the Anubar and averaging Pitot tube breath flow measurements are compared for use in signal measurement and assessing system response. Testing procedures involved the following series of investigations:

- Breath signal measurement – This test compares the effects of restriction sizes on the exhaust of the sample tube, as well as the system response using both the Anubar and averaging Pitot tube sensors

- Alveolar sample collection – Testing of breath samples collected into Tedlar bags using the device with the restrictor and sample tube selected as a result from the signal measurement test series.

8.1 Breath Signal Measurement

As a person exhales into the sample tube of the breath collection device, the breath profile is affected by the pressure in the sample tube. By adding a restriction onto the exhaust of the sample tube, the pressure measurements are increased in magnitude and thus the noise associated with the pressure transducer signal is minimised. The diameter of the sample tube is 30 mm and the restrictors reduced the exhaust diameter to a smaller value. Two different exhaust diameters were examined, 17 mm and 7 mm. To measure the effects of restricting the exhaust diameters on the pressure transducer signal and solenoid valve firing times, their respective voltages were measured using the dSpace™ system, sampling at 100 Hz, which is the same sampling frequency as the CY8C27433-24PXI PSoC.

Three exhalation variations were examined for both exhaust sizes and both sample tubes, Anubar and averaging Pitot tube. Sensors were tested for signal clarity and the resultant accurate solenoid firing times for the entire collection system. These three exhalation variations were forced vital capacities (VCs), slow VCs and tidal volumes. These represent three extremes of likely breath testing modalities. Overall, these tests span a range of exhalation types, sensor types and the resulting system response to ensure a system that is as robust as possible.

The subject providing the breaths for testing the remote collection device provided three breaths, which are used by the device as an average breath profile. Subsequent breaths were kept consistent to the average breath profile by allowing the subject to view the LED indicator, which showed them how far through the profile they had exhaled. By using this method of training the subject to breathe into the device, the

true operation in practice was tested. Figures 8.1 to 8.12 show the captured pressure transducer and solenoid valve signals during operation of the breath collection device with the two sample tubes and restriction sizes.

Results from the breath signal testing indicated that the 7 mm exhaust diameter produced a clearer signal from the pressure transducer and that the solenoid valves were always fired at the appropriate times. When the 17 mm exhaust was used, the signal from the pressure transducer was noisy and often fell below the zero voltage level during an exhalation. Consequently, the solenoid valves were repeatedly opened and closed multiple times during a single breath and would not fractionate breath samples adequately. Examples of this latter case can be seen in the plots of Figures 8.1, 8.2, 8.3 and 8.9.

A comparison of the Anubar and averaging Pitot tube showed that signals from the averaging Pitot tube were clearer with less associated noise. In some cases, signals obtained using the 17 mm exhaust with the averaging Pitot tube were smoother than signals from the Anubar with a 7 mm exhaust. For example, the forced and slow vital capacity exhalations successfully fractioned with the 17 mm exhaust shown in Figure 8.7 and Figure 8.8. Although the Anubar is easier to machine, the averaging Pitot tube provides smoother pressure profiles that do not result in unwanted firing of the solenoid valves. Hence, the averaging Pitot tube is the flow measurement method of choice for the remote breath collection device.

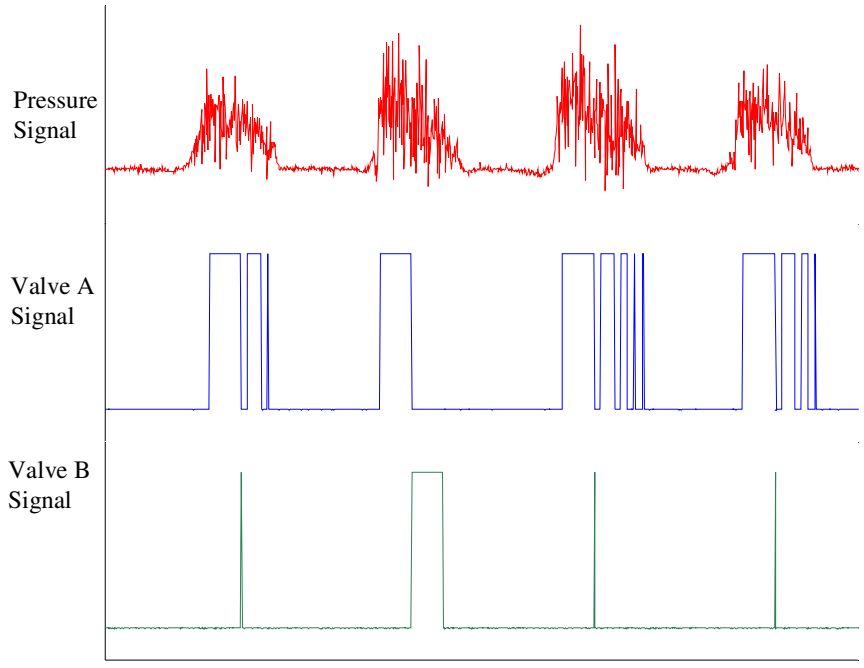


Figure 8.1- Forced VC using Anubar with 17 mm exhaust diameter

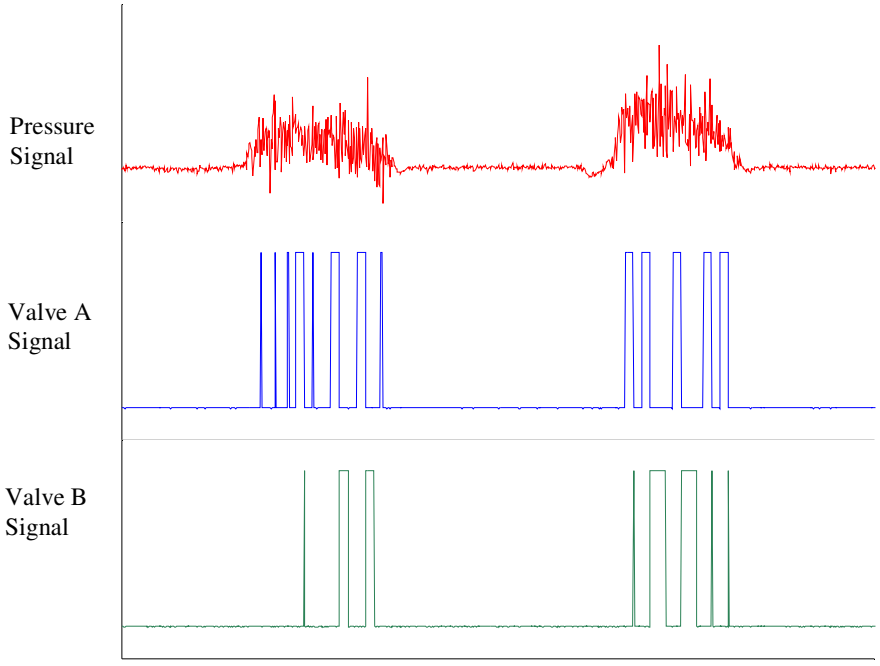


Figure 8.2 - Slow VC using Anubar with 17 mm exhaust diameter

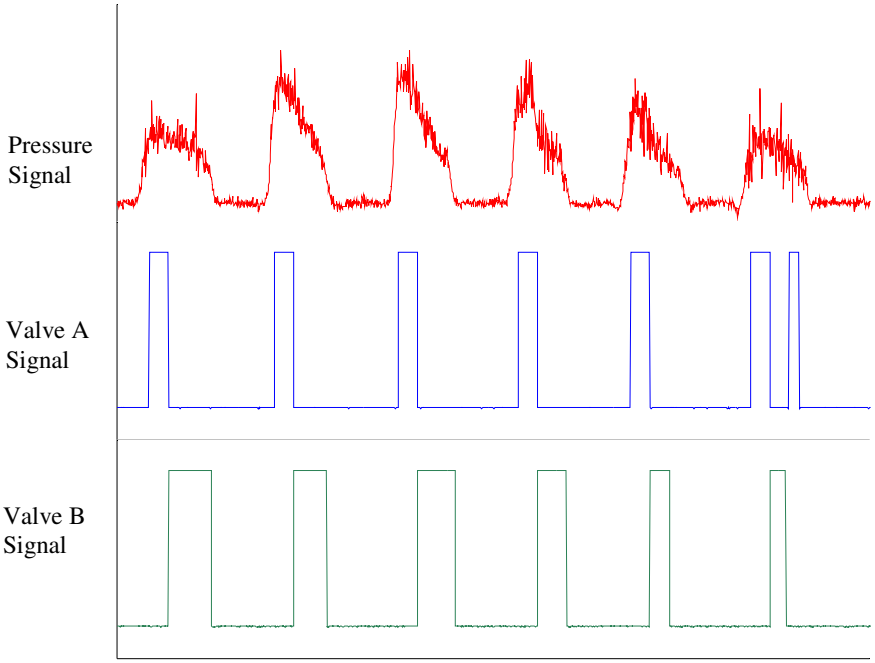


Figure 8.3 - Tidal Volumes using Anubar with 17 mm exhaust diameter

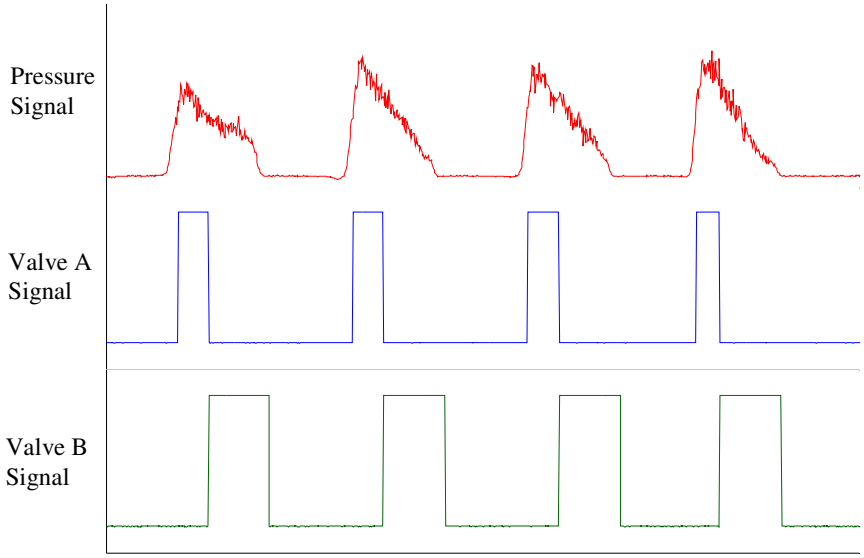


Figure 8.4 - Forced VC using Anubar with 7 mm exhaust diameter

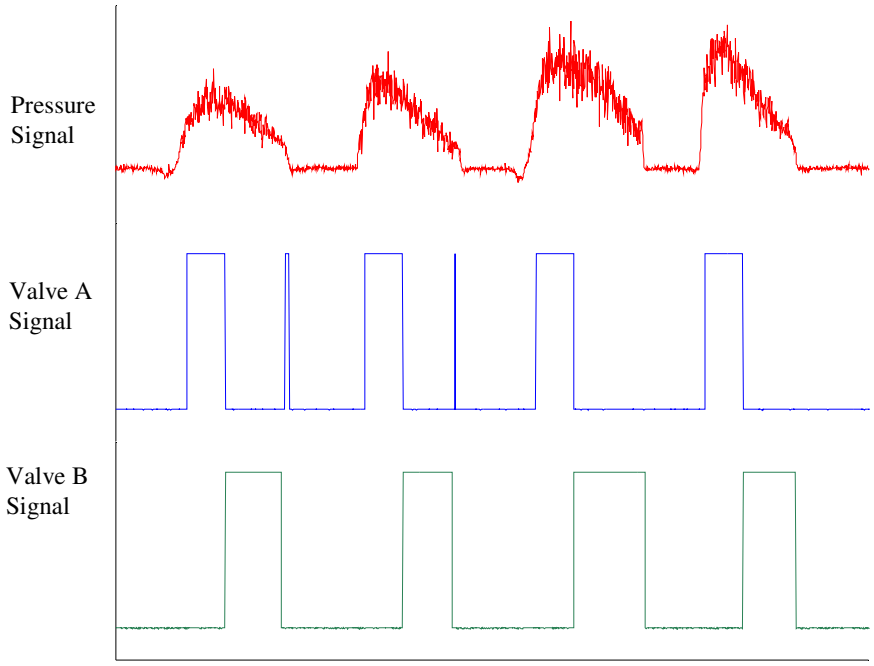


Figure 8.5 - Slow VC using Anubar with 7 mm exhaust diameter

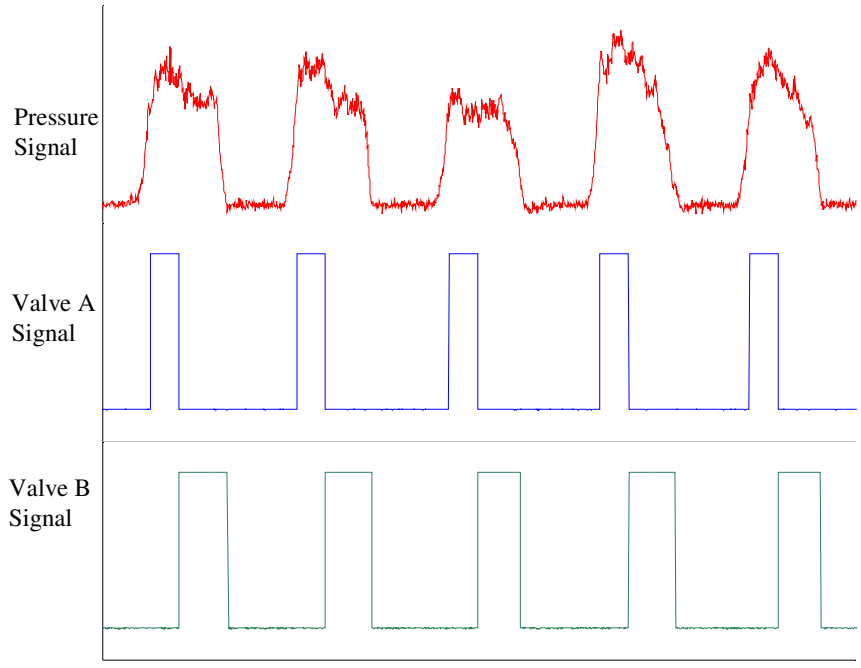


Figure 8.6 - Tidal Volumes using Anubar with 7 mm exhaust diameter

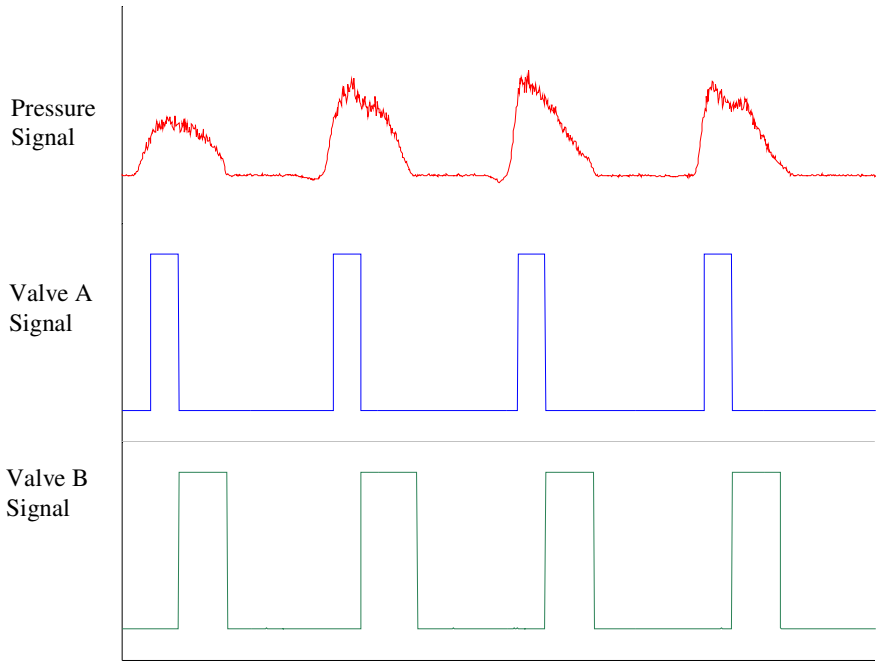


Figure 8.7 - Forced VC using Pitot tube with 17 mm exhaust diameter

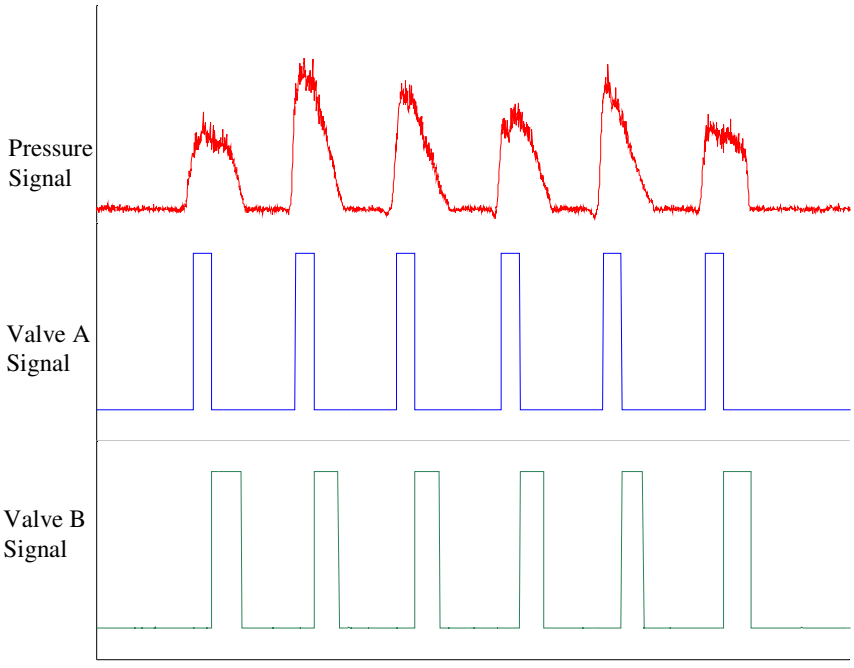


Figure 8.8 - Slow VC using Pitot tube with 17 mm exhaust diameter

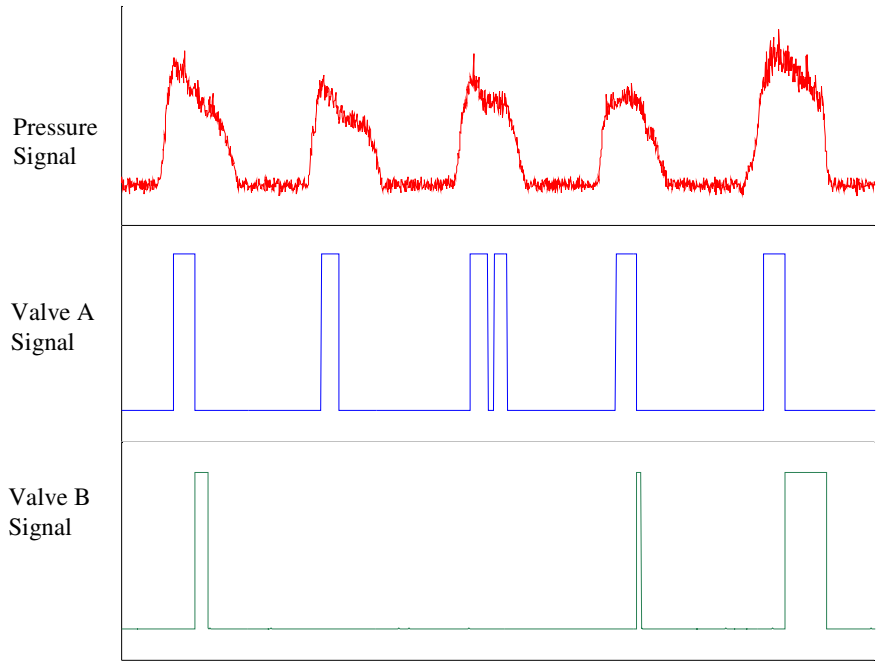


Figure 8.9 - Tidal Volumes using Pitot tube with 17 mm exhaust diameter

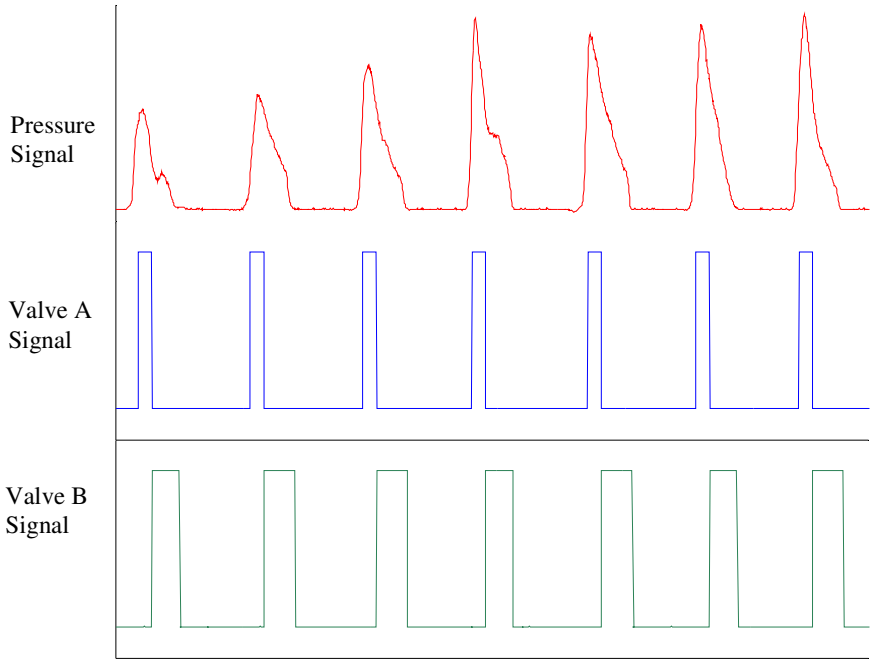


Figure 8.10 - Forced VC using Pitot tube with 7 mm exhaust diameter

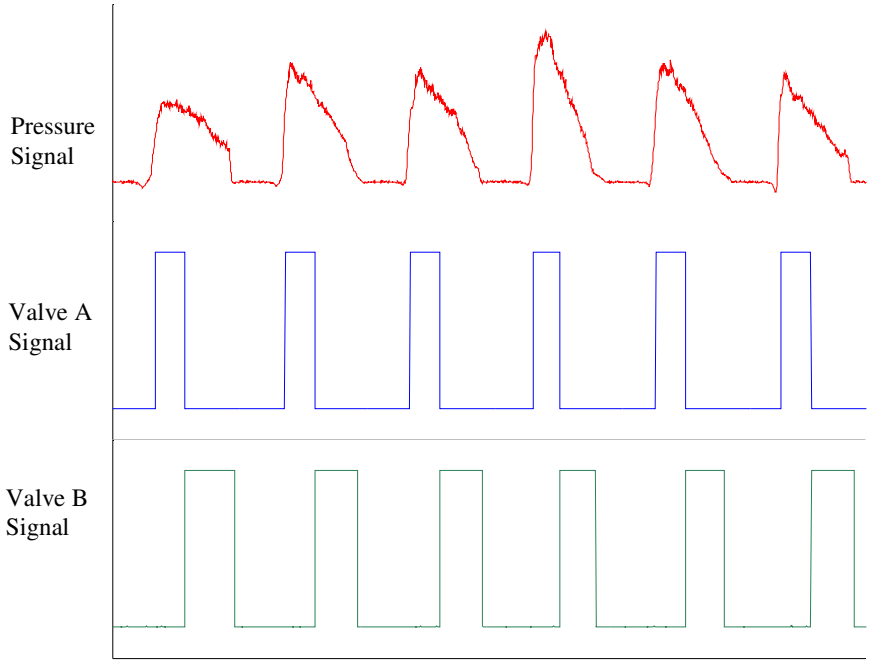


Figure 8.11 - Slow VC using Pitot tube with 7 mm exhaust diameter

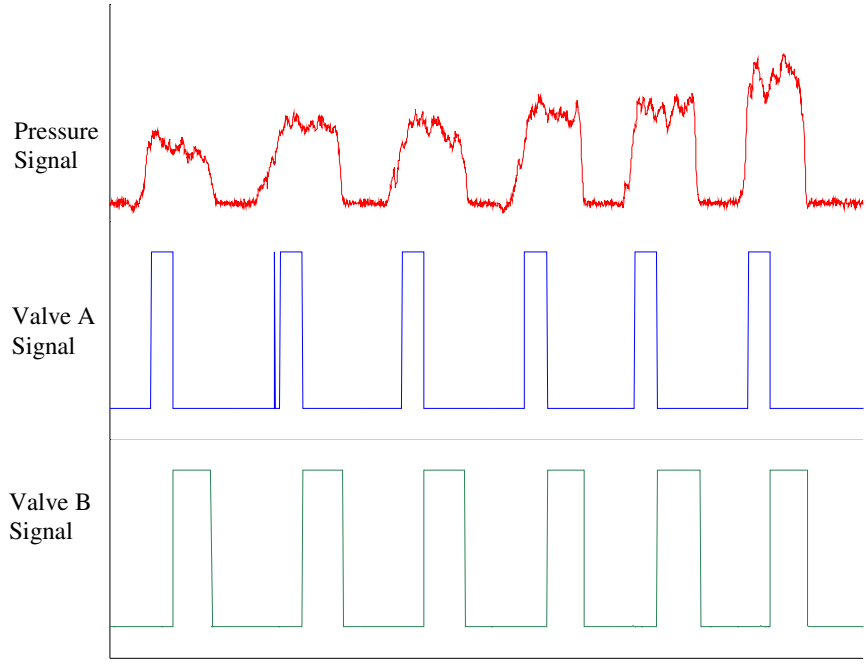


Figure 8.12 - Tidal Volumes using Pitot tube with 7 mm exhaust diameter

8.2 Alveolar Sample Collection

This series of tests examines the ability of the device to fraction the later, alveolar portions of breath exhalation. These fractions were set to be 0 – 20% (Fraction 1) and 60 – 100% (Fraction 2) of the breath exhalation profile. Fraction 1 is a collection of breath originating from the upper airways, essentially ambient air and breath that has been partially mixed with alveolar gasses. Due to mixing of deadspace and alveolar breath, a higher level of endogenously produced VOCs than ambient air is expected in Fraction 1. However, Fraction 1 is expected to have a significantly lower concentration of VOCs than Fraction 2, which is purely an alveolar sample.

These two fractions were collected and analysed using the SIM scan mode on a SIFT-MS instrument at the University of Otago, Christchurch School of Medicine and Health Sciences. The two analytes monitored for this validation study were acetone and isoprene. Acetone and isoprene were chosen as analytes to monitor because they are commonly found in breath at significantly higher levels than that of ambient air. As well as being common analytes found in breath, isoprene and acetone are easy to monitor with very few analytical complications, and are found at levels well within the detection limits of the instrument. Fractionated breath samples were compared to acetone and isoprene levels of un-fractioned breath, the whole air sample, collected in a Tedlar bag. Figure 8.13 and 8.14 show the levels of acetone and isoprene in each of the breath fractions, as well as, for both, an un-fractioned sample and an ambient air level sample.

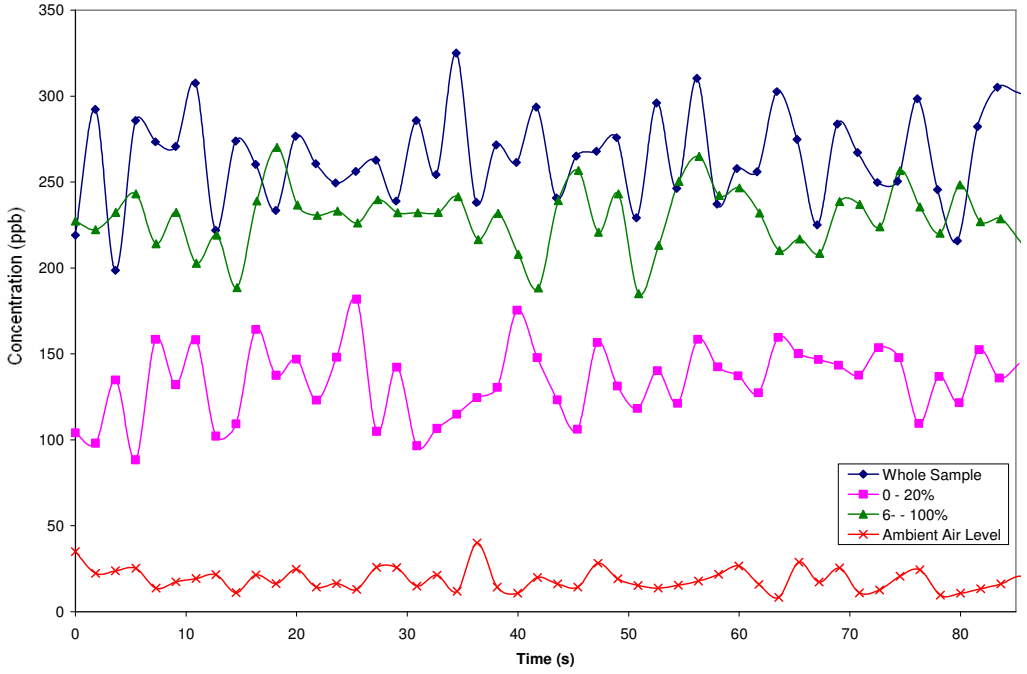


Figure 8.13 - Acetone levels Observed in Fractioned and Un-fractioned Breath

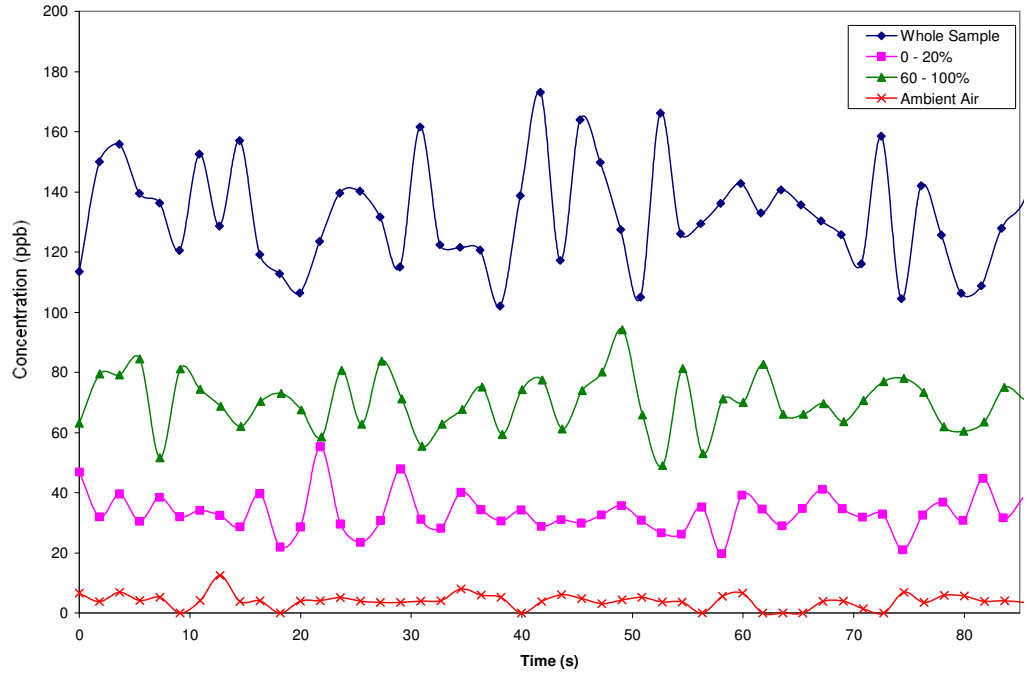


Figure 8.14 - Isoprene Levels in Fractioned and Un-Fractioned Breath

SIM scans on SIFT-MS indicated that breath fractions had successfully been collected. In particular, Fraction 1 contained significantly lower levels of acetone and isoprene than Fraction 2. However, the levels of analyte found in Fraction 1 were higher than that of the ambient air in the laboratory. These higher levels were caused by the mixing of alveolar gasses with dead space gasses in the airways. Hence, Fraction 1 had a higher VOC concentration than the ambient air, but was significantly lower than Fraction 2.

Figure 8.13 shows a level of acetone in Fraction 2 similar to that found in the whole breath sample, indicating that alveolar breath was collected into the Tedlar bag. However, levels of isoprene were significantly lower in the fractionated sample than the whole breath sample. Studies have examined breath isoprene levels during various stages of exercise and found that the levels of this analyte are directly related to breathing rate (Karl & Prazeller, 2001). As heart rate increases isoprene levels increase as more blood exchanges VOCs with air in the lungs. However, as breathing rate increases, the level of isoprene decreases. Therefore, since the breathing rate increased during collection of the fractioned breath samples, the isoprene levels dropped significantly lower than the whole breath sample.

The decrease in VOC concentrations due to increased breathing rate may also apply to other compounds. Hyperventilation may result in less time for blood to exchange gasses in the lungs. To overcome the dilution in analyte collected in the Tedlar bags, investigation into collecting a higher volume of sample per breath is required and controlling the rate of patient exhalations may be important.

8.3 Chapter Summary

To validate the operation of the breath fractionation device, two sets of tests were undertaken. The first test investigated differences between the flow measurement method and restriction of the exhaust on the sample tube. Results indicated that the averaging Pitot tube resulted in clearer signals that did not contain excessive noise. Clearer signals from the averaging Pitot tube were expected as it measures the average velocity profile across the sample tube, buffering fluctuations in fluid flow that would affect the Anubar measurements. Geometrically, a 7 mm exhaust diameter was required to create higher pressures inside the sample tube and to minimise the effect of excessive pressure transducer signal noise that resulted in unwanted firing of the solenoid valves.

To validate the fractionation of alveolar breath, acetone and isoprene were analysed in fractioned breath and compared with levels found in a whole breath sample. Analysis of the fractionated breath showed that the upper airway portion collected had a significantly lower concentration of acetone and isoprene than the alveolar fraction. The lower concentration of VOCs in Fraction 1 compared with Fraction 2 was expected as Fraction 1 consisted of air from the airways deadspace region.

Fraction 2 contained a slightly lower acetone concentration when compared with the whole breath sample as expected, due to an increased breathing rate. Isoprene concentrations in the alveolar sample were approximately 50% of the concentration of the whole breath sample. The lower isoprene concentration can be explained as studies have shown that isoprene levels decrease with an increased breathing rate (Karl & Prazeller, 2001).

To overcome the issue of VOC dilution in alveolar sampling the device needs to collect more sample per exhaled breath. The result being, that fewer breaths are required to collect the fractionated samples. This can be achieved by increasing the vacuum pressure inside the device case by including either a second pump or sourcing one with a higher pumping capacity. In addition to collecting more sample per breath exhalation, the subject could regulate their breathing rate so that the device obtains a true level of VOC concentration.

9

Conclusions and Future Work

This Chapter reviews the final designs of both breath testing devices designed for SIFT-MS instruments. Many design aspects relating to the direct breath collection device also apply to the remote breath collection device and have been implemented into their designs. These design considerations include materials, geometry and patient safety. Included in this chapter are recommendations for future work and modifications that may be implemented into the 2nd generation of breath collection equipment for SIFT-MS instruments.

9.1 Direct Device

Conclusions and Summary:

A direct breath collection device was designed and built to interface with a SIFT-MS instrument. The findings indicate that all materials emit some level of VOCs and the best compromise was to try to minimise this and thus minimise sample contamination. Therefore, materials involved in the direct flow of breath leading to the SIFT-MS flow tube should be chosen specifically to minimise external VOC emission. Therefore, all components in direct contact with the breath sample are made from stainless steel. All other components should be made from materials that would limit

VOC emission from external sources as much as possible. To satisfy the range of user anthropometrics, a rotating joint is used to attach the direct breath collection device to the SIFT-MS instrument, as flexible tubes are not suitable because of sample contamination from VOC retention or external VOC emission.

To minimise flow resistance experienced by the patient, the capillary from which SIFT-MS samples is placed as close as possible to the user's mouth. However, to optimise sample collection, the inlet and exhaust had to be reduced to 6.35 mm O.D stainless steel tube, and straws are used as disposable mouthpieces. The small diameter inlet is chosen to optimise the sample collection and expose the capillary to breath for a longer period of time.

The optimal operating temperature of the device was found to be 100 - 120°C, which ensures that water vapour will not condense inside the capillary causing blockage. This temperature is also limited by the materials and current heating method of the SIFT-MS instrument lines. However, to ensure patient safety from the elevated device operating temperature, an air gap and FEP coated device case are used as an insulator. Using an air gap as an insulator also minimises external VOC contamination of samples from any insulation methods.

Future Work and Recommendations:

The following recommendations are made for any following designs:

- Integrating temperature control of the device via Syft Technologies instrument software

- Simplify the heating system to be easily disassembled for sterilisation of device internals and mouthpiece
- Incorporate a filter system for mouthpieces to prevent patients coming into contact with infection risk from previous patients.

9.2 Remote Breath Collection Device

Conclusions and Summary:

Currently designed and commercially available alveolar breath collection devices were considered as possible methods of remote breath collection for SIFT-MS instruments. However, the majority of previously designed alveolar breath sampling devices employed the Haldane-Priestly tube principle. The Haldane-Priestly tube collects an alveolar breath sample but does not allow the flexibility of collecting breath from any specified region of an exhalation, nor does it allow for the separate collection of the deadspace volume.

Currently designed devices are designed to collect breath samples on to adsorbent traps, which is not a suitable method for analysis using the SIFT-MS technique. Currently the optimal storage medium for the collected breath samples is Tedlar bags or a similar system/approach. By using Tedlar bags, breath could be stored as a whole air sample, the ideal form for analysis using the SIFT-MS method.

The chosen breath fractionation method for the device involved measurement of breath through the device and sample collection based on the exhaled profile. Collection of breath samples is triggered by the exhaled breath profile, as measured

via an averaging pitot tube connected to a pressure transducer. The averaging Pitot tube was chosen over the Anubar because of clearer signals measured by the pressure transducer. A 7 mm restrictor was required on the exhaust of the sample tube to create the larger pressure profiles required for measurement with the pressure transducer.

To prevent exposure to potential infection to patients using the device, the inlet incorporates the use of disposable mouthpieces. Minimising contamination of collected samples from external VOC sources, requires the parts of the device that are in direct contact with the exhaled breath to be heated above 37°C and are made from either stainless steel or FEP.

The remote breath collection apparatus for the SIFT-MS technique has been designed to collect samples into 1 litre Tedlar bags. Two Tedlar bags are inserted into the device and a Perspex case placed over them. A partial vacuum pressure is created inside the Perspex case by evacuating the air inside with a diaphragm pump. The breath sample is drawn through the tubing in the device via the pressure difference inside the case and the atmosphere. This method of collecting samples into Tedlar bags is recommended by the standard air sampling guidelines (United States, Environmental Agency, 2003). A diaphragm pump was chosen to create the vacuum pressure inside the case and was selected for its high pumping capacity and low environmental noise.

The initial stages of the software and electronic hardware design involved selection and testing of appropriate pressure transducers. Three pressure transducers were tested and the Suresense™ was selected because it was thermally stable and self amplified.

The controls implemented on the breath collection device use capacitive touch sense buttons because they are robust and can be easily wiped down if required.

A software proof of concept was created using Simulink and D-Space, which showed that breath could be fractioned on percentages of an example exhaled breath profile. The microcontroller used for the final breath collection prototype device was a CY8C27443-24PXI PSoC from Cypress Microsystems. The PSoC was used because it can integrate analog and digital components in a simple low-cost embedded system.

The software architecture implemented on the breath collection device involved five individual sequences, each operating with its own finite state system. The five sequences in the breath collection device software are the purging, data entry, average breath, breath fractionation and breath collection summary sequences.

To validate the fractionation of alveolar breath, acetone and isoprene were analysed in fractioned breath and compared with levels found in a whole breath sample. Samples were analysed using the SIM scan mode on a SIFT-MS instrument. The analysis of the fractionated breath showed that the upper airways portion collected had a significantly lower concentration than the alveolar fraction indicating that the device had successfully fractionated breath samples. The alveolar fraction contained a slightly lower acetone concentration, compared with the whole breath sample. Isoprene concentrations in the alveolar sample were approximately half the concentration of the whole breath sample that was probably caused by an increase in breathing rate. The increased breathing rate restricted the time for blood to exchange VOCs with air in the lungs. To eliminate the dilution of alveolar samples collected,

the device would need to increase the volume of sample collected per exhalation, and regulate breathing rate from the subject.

Future Work and Recommendations:

The following recommendations are made for any successive designs:

- Faster sample collection could be achieved by implementing a larger pump into the device. This would minimise hyperventilation by the subject when providing a breath sample, thus reducing analyte dilution effects
- More user feedback could be integrated, including providing information on vacuum pressure inside the case to inform the device operator when the case is ready to collect breath samples
- Integration of pump and case, as the pump is currently too large to fit inside the breath collection device and is a separate unit. A pump could be designed and integrated into the device to create one single unit
- Assembly optimisation is required as the current remote breath collection device is difficult to assemble and could be designed to be easily assembled/disassembled for sterilisation.

References

BAUMBACH, J. I., VAUTZ, W. & RUZSANYI, V. (2005) Metabolites in Human Breath: Ion Mobility Spectrometers as Diagnostic Tools for Lung Diseases. IN SMITH, A. A. D. (Ed.) *Breath Analysis for Clinical Diagnosis and Therapeutic Monitoring.* Singapore, World Scientific Publishing Co. Pte. Ltd.

DWEIK, R. A. (2005) Nitric Oxide in Exhaled Breath: A Window on Lung Physiology and Pulmonary Disease. IN SMITH, A. A. D. (Ed.) *Breath Analysis for Clinical Diagnosis and Therapeutic Monitoring.* Singapore.

DYNE, D., COCKER, J. & WILSON, H. K. (1997) A novel device for capturing breath samples for solvent analysis. . *The Science of the Total Environment,* 199**,** 83-89.

GUSTAFSSON, L. E. (2005) Exhaled Nitric Oxide: How and Why we know it is Important. IN SMITH, A. A. D. (Ed.) *Breath Analysis for Clinical Diagnosis and Therapeutic Monitoring.* Singapore, World Scientific Publishing Co. Pte. Ltd.

HAMILTON, L. H. (1998) *Breath Tests & Gastroenterology,* Milwaukee, QuinTron Instrument Company.

KARL, T. & PRAZELLER, P. (2001) Human breath isoprene and its relation to blood cholesterol levels: new measurements and modeling. *Journal of Applied Physiology,* 91**,** 762 - 770.

NIOX (2004) NIOX gets asthma insights. Solna, Sweden.

OMEGA ENGINEERING, I. Transactions in Measurement and Control. *Flow & Level Measurement.* OMEGA Engineering, Inc.

PHILLIPS, M. (1999) Variation in volatile organic compounds in the breath of normal humans. . *Journal of Chromatography,* 729**,** 75-88.

PHILLIPS, M. (1997) Method for the Collection and Assay of Volatile Organic Compounds in Breath. *Analytical Biochemistry,* 247**,** 272-278.

PHILLIPS, M., GREENBERG, J. & AWAD, J. (1994) Metabolic and environmental origins of volatile organic compounds in breath. . *F Clin Pathol,* 47**,** 1052-1053.

RAYMER, J. H., THOMAS, K. W., COOPER, S. D., WHITAKER, K. A. & PELLIZZARI, E. D. (1990) A device for samplin of human alveolar breath for the measurement of expired volatile organic compounds. *Journal of Analytical Toxicology,* 14**,** 337-344.

RISBY, T. H. (2005) Current Status of Clinical Breath Analysis. IN SMITH, A. A. D. (Ed.) *Breath Analysis for Clinical Diagnosis and Therapeutic Monitoring.* Singapore, World Scientific Publishing Co. Pte. Ltd.

SCHOOL OF CHEMISTRY, UNIVERSITY OF BRISTOL. (2005) Mass Spectrometry Resource.

SCHUBERT, J. K., SPITTLER, K.-H., BRAUN, G., GEIGER, K. & GUTTMANN, J. (2001) CO2-Controlled sampling of alveolar gas in mechanically ventilated patients. . *Journal of Applied Physiology,* 90**,** 486-492.

SILKOFF, P. E. (1999) Recommendations for a Standardised Procedure for the Online Measurement of Exhaled Nitric Oxide in Adults. *Reccommendations for Standardised procedures for the online and offline measurement of exhaled lower respiratory nitric oxide and nasal nitric oxide in adults and children.* . American Journal of Respiratory and Critical Care Medicine.

SOUBANI, A. O. (2001) Noninvasive Monitoring of Oxygen and Carbon Dioxide. *American Journal of Emergency Medicine,* 19**,** 141-146.

SMITH, D. & SPANEL, P. (2004) Selected Ion Flow Tube Mass Spectrometry (SIFT-MS) for On-line Trace Gas Analysis. *Mass Spectrometry Reviews,* 24**,** 661-700.

SMITH, D. & SPANEL, P. (2005) Selected Ion Flow Tube Mass Spectrometry, SIFT-MS, for On-Line Trace Gas Analysis of Breath. IN SMITH, A. A. D. (Ed.) *Breath Analysis for Clinical Diagnosis and Therapeutic Monitoring.* Singapore, World Scientific Publishing Co. Pte. Ltd.

SYFT (2004a) SIFT-MS: Absolute Concentrations in Real Time. Christchurch, New Zealand, Syft Technologies Ltd.

SYFT (2004b) Technology Comparison: GC-MS and SIFT-MS. Christchurch, New Zealand, Syft Technologies Ltd.

SYFT (2006) Application Examples. Christchurch, New Zealand, Syft Technologies Ltd.

TAMARKIN, D. A. (2006) Anatomy & Physiology. *Respiratory System (Unit 22).* Springfield, Massachusetts.

UNITED STATES ENVIRONMENTAL PROTECTION AGENCY (2003) General Air Sampling Guidelines, Standard Operating Procedure Eleven.

YEUNG, C. Y., MA, Y. P., WONG, F. H., KWAN, H. C., FUNG, K. W. & TAM, A. Y. C. (1991) Automatic end-expiratory air sampling device for breath hydrogen test in infants. *The Lancet,* 337**,** 90-93.

Appendices

A1 Capillary Calibration

To calibrate a capillary tube it is connected in series to a mass flow controller (MFC) during normal operation. One end of the MFC is open to the atmosphere and the other is connected to the capillary. As air is drawn through the MFC it returns a voltage representing the flowrate measured. The flow rate of air through the MFC is measured, based on the heat transfer to the sensor from the air travelling through it. To calibrate the capillary on the direct breath collection device prototype an argon MFC was used that had a 0 – 5 V scale and measured a flow of 0 – 1000 Standard Cubic Centimetres per Minute (SCCM). Therefore, calibration of the capillary uses the following series of calculations to state the flow through it in Torr litres per second (TL/s).

The MFC returned 0.96 V calibrated for Argon.

To convert from Argon to air:

$$
\begin{aligned}
0.96\text{V} &= 96 \times 2 \text{ SCCM Argon} \\
&= \frac{192}{1.39} \text{ SCCM Air} \\
&= 138.13 \text{ SCCM @ 273.15K}
\end{aligned}
$$

1.39 is a correction factor for Nitrogen (N_2) to Argon and is used as air is mostly N_2. After converting the flow rate, in SCCM, to air it must be adjusted to account for temperature in the lab using the universal gas law.

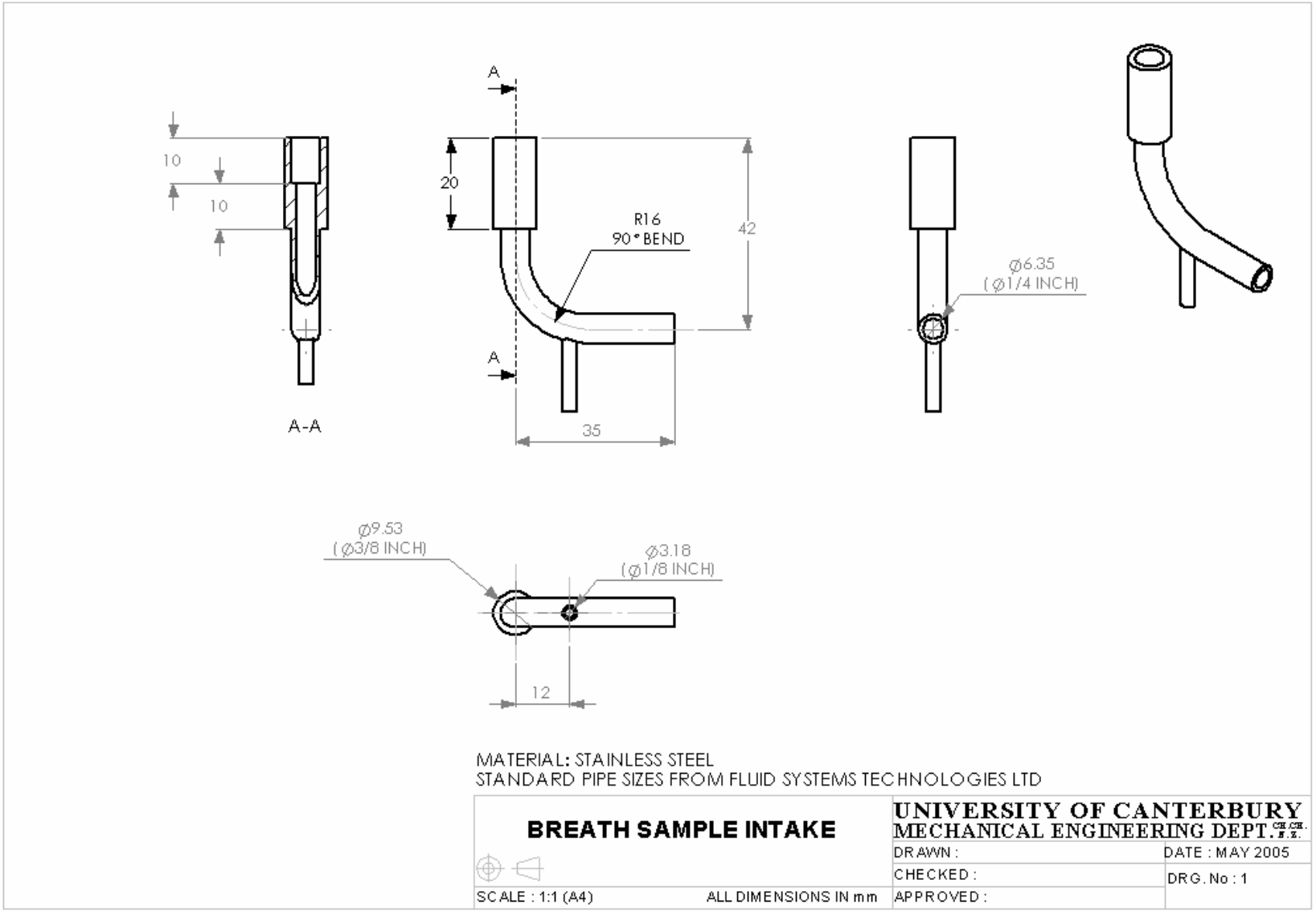
$$\frac{V_1}{T_1} = \frac{V_2}{T_2}$$
$$\frac{138.13}{273.15} = \frac{V_2}{293}$$
$$V_2 = 148.17 \text{ SCCM @ } 293\text{K}$$

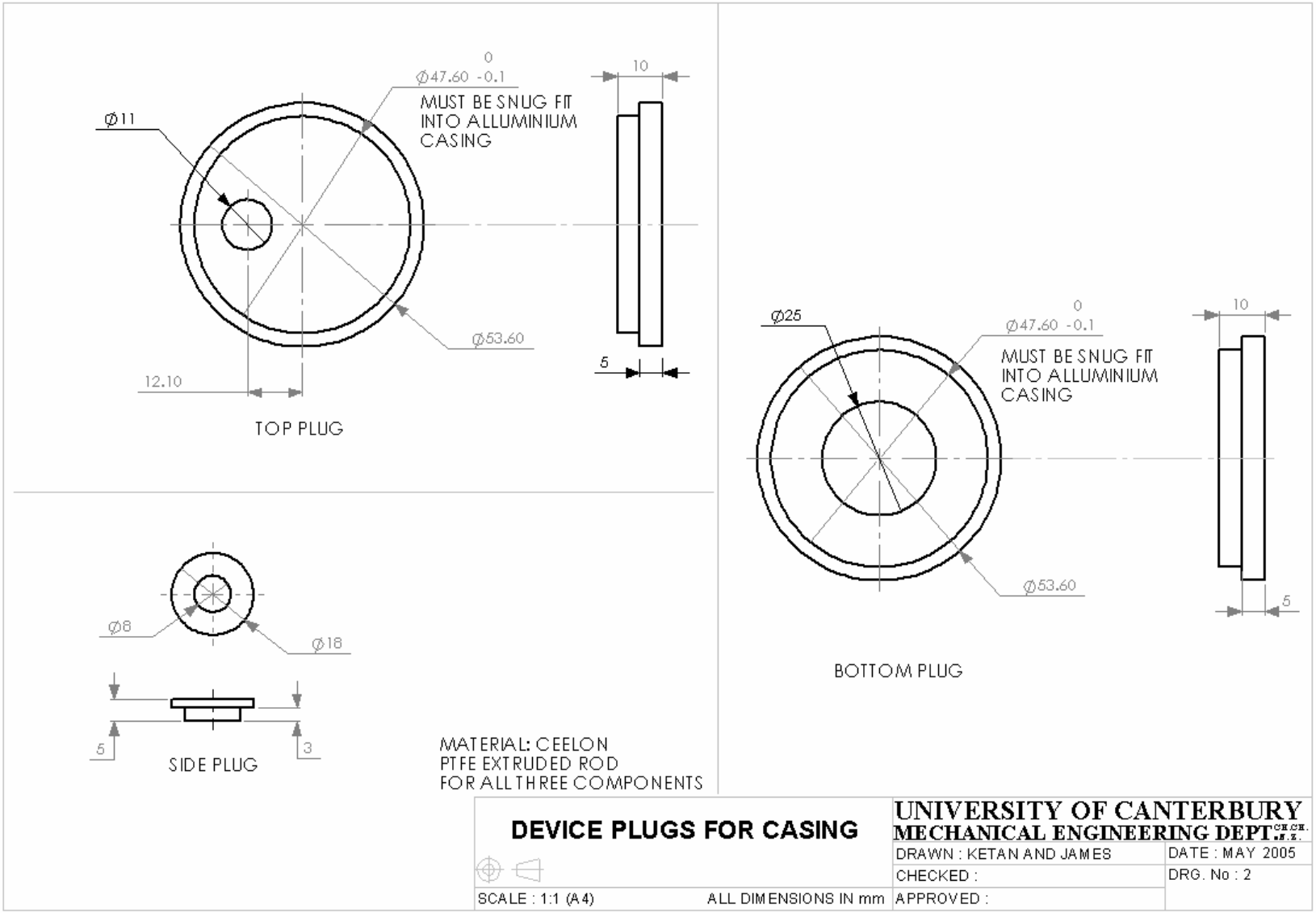
To convert to TL/s the units are converted from SCCM to Litres per second (L/s) then to TL/s.

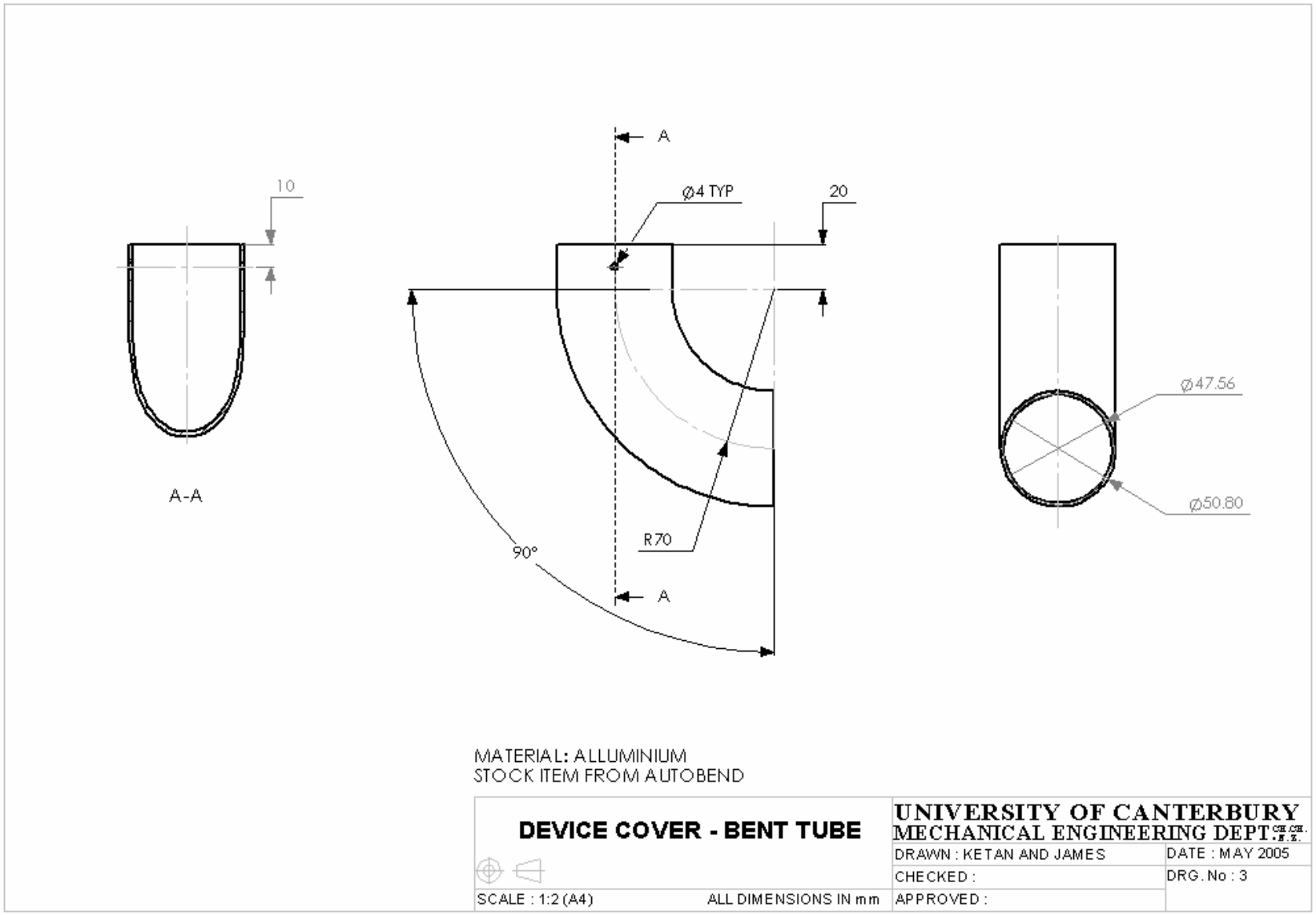
$$148.17 \text{ SCCM} = \frac{148.17}{1000 \times 60} \text{Ls}^{-1}$$
$$= \frac{148.17}{1000 \times 60} \times 760 \text{ TLs}^{-1}$$
$$= 1.877 \text{ TLs}^{-1}$$

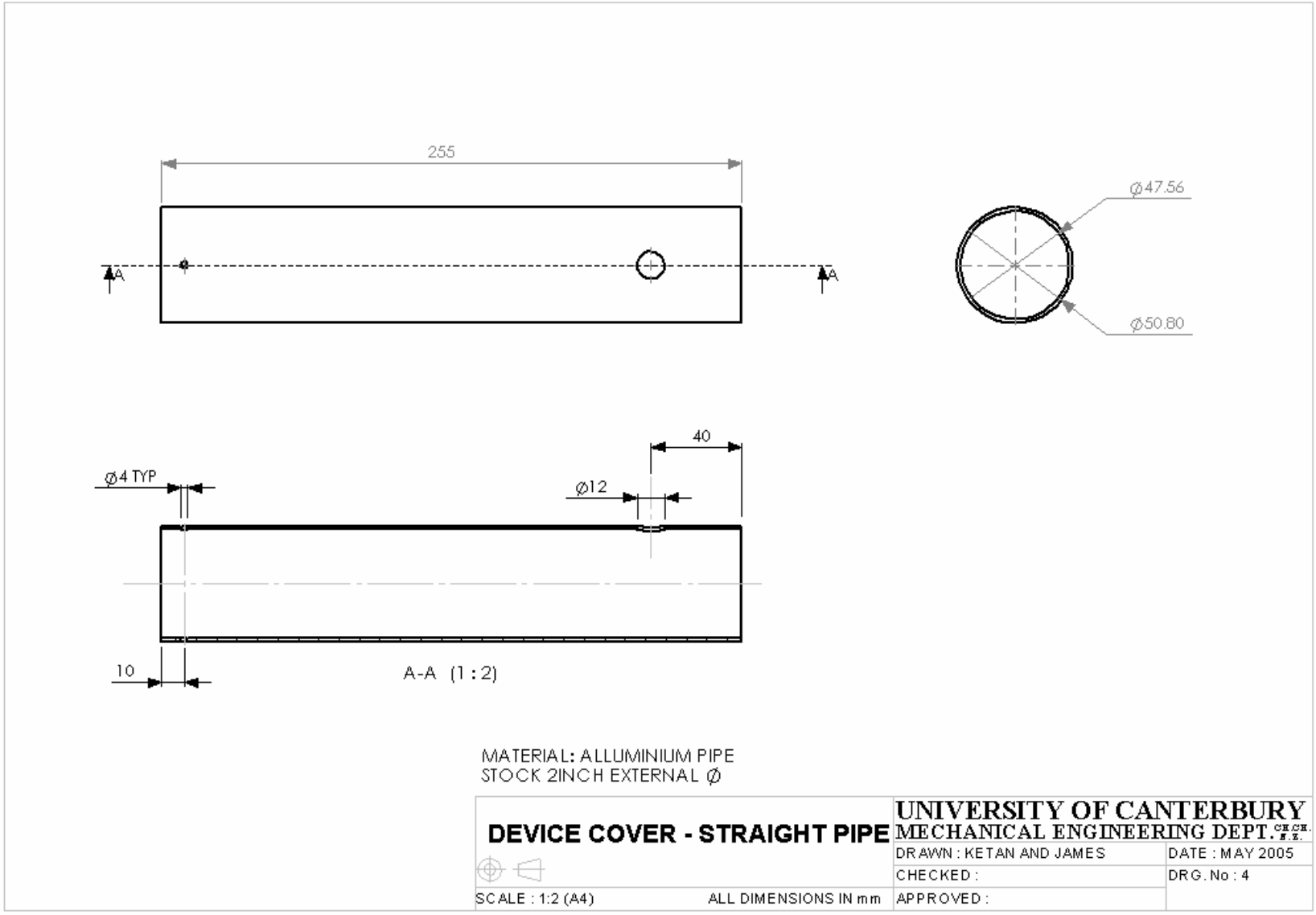
The capillary flow rate is entered into the SYFT Technologies instrument software and is used to calculate the concentration of analytes in ppb from cps.

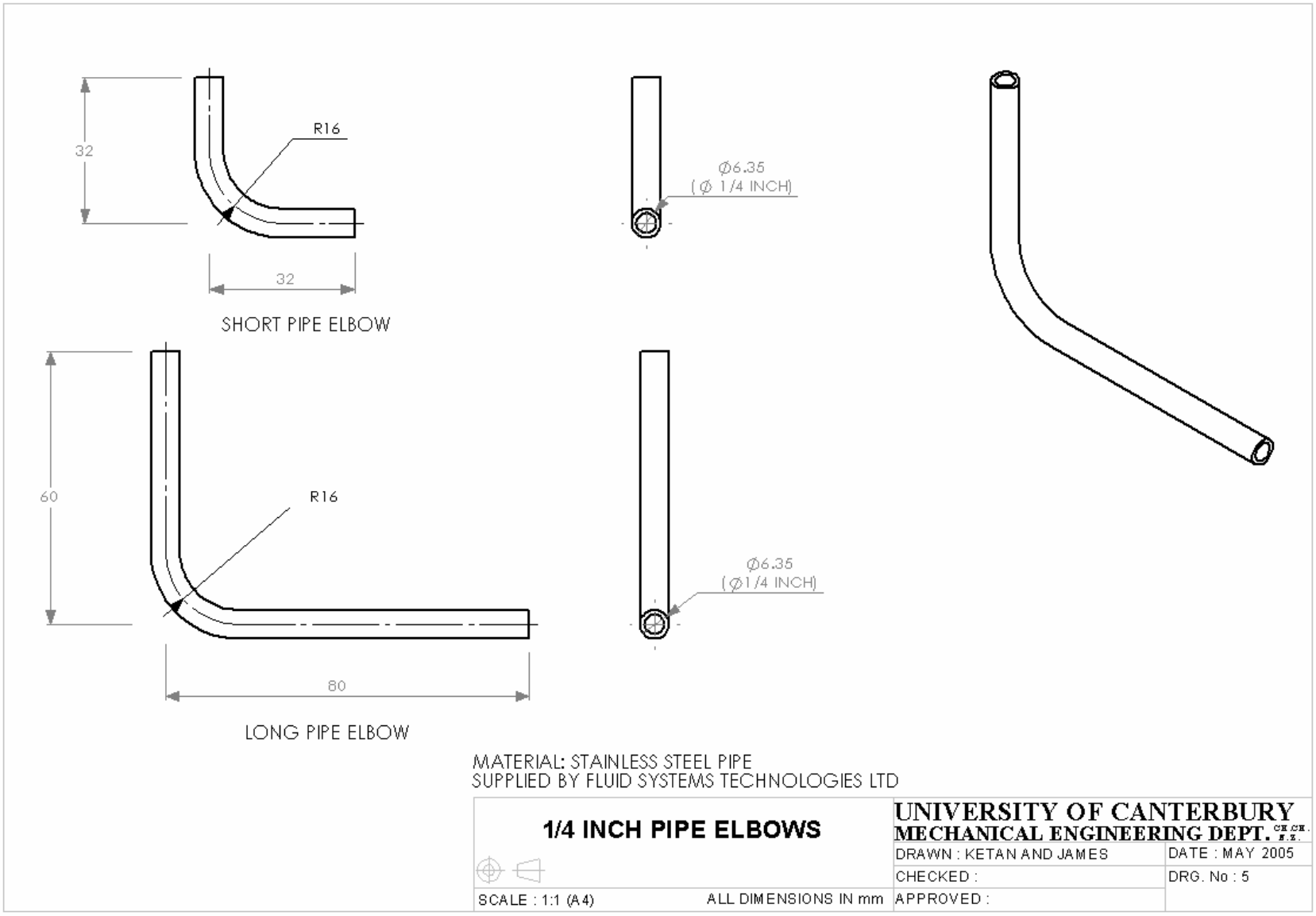
A2 Direct Breath Collection Device Drawings

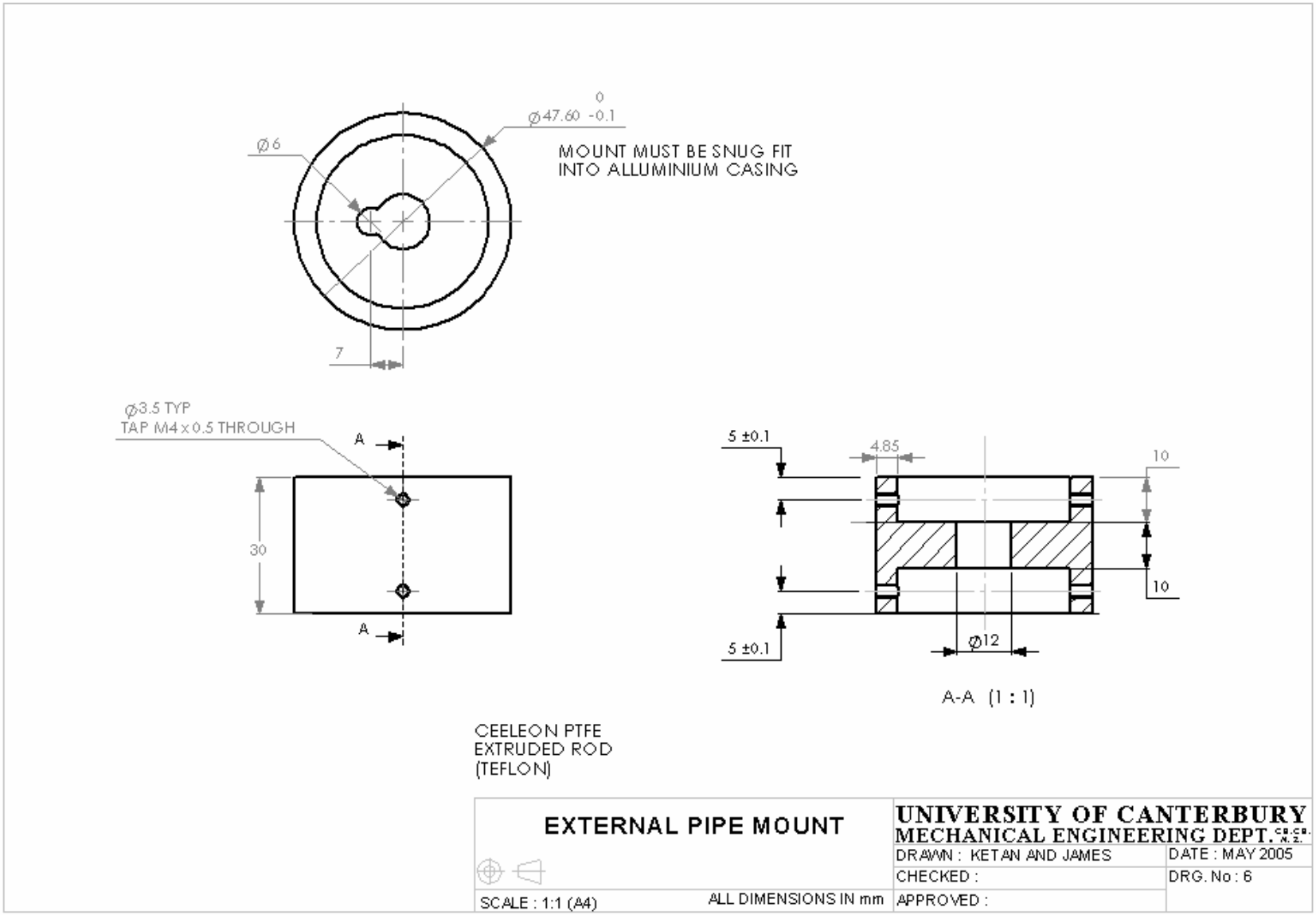

10
10
A-A
A
20
R16
90° BEND
42
A
35
Ø6.35
(Ø1/4 INCH)
Ø9.53
(Ø3/8 INCH)
Ø3.18
(Ø1/8 INCH)
12
MATERIAL: STAINLESS STEEL
STANDARD PIPE SIZES FROM FLUID SYSTEMS TECHNOLOGIES LTD
BREATH SAMPLE INTAKE
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.
DRAWN :
DATE : MAY 2005
CHECKED :
DRG. No : 1
SCALE : 1:1 (A4)
ALL DIMENSIONS IN mm
APPROVED :

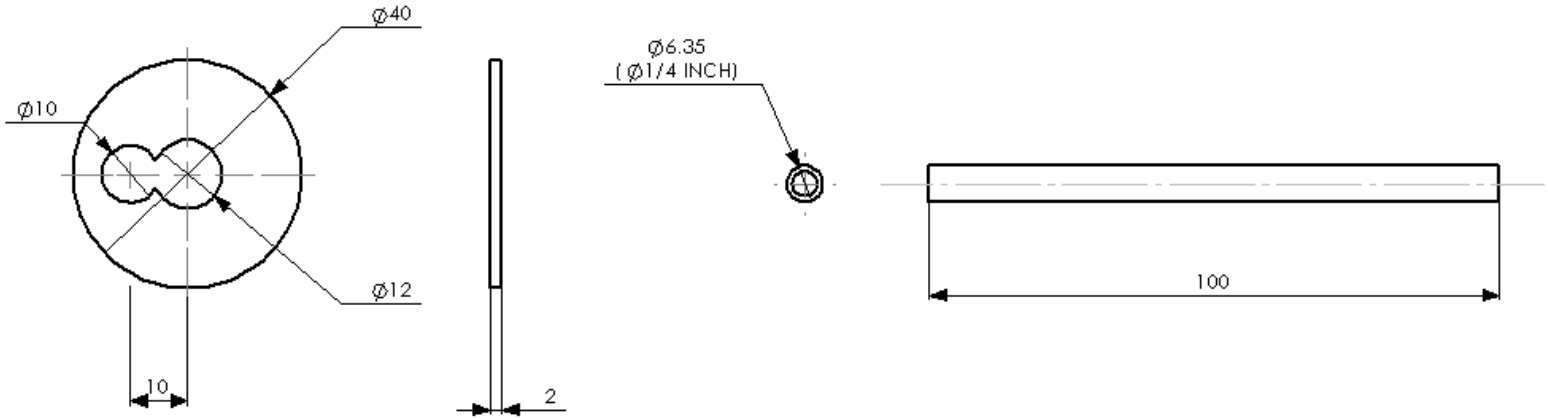

Ø47.60 0 -0.1
MUST BE SNUG FIT INTO ALLUMINIUM CASING
10
Ø11
Ø53.60
5
12.10
TOP PLUG
Ø25
Ø47.60 0 -0.1
MUST BE SNUG FIT INTO ALLUMINIUM CASING
10
Ø53.60
5
BOTTOM PLUG
Ø8
Ø18
5
3
SIDE PLUG
MATERIAL: CEELON
PTFE EXTRUDED ROD
FOR ALL THREE COMPONENTS
DEVICE PLUGS FOR CASING
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT. CHCH. N.Z.
DRAWN : KETAN AND JAMES
DATE : MAY 2005
CHECKED :
DRG. No : 2
SCALE : 1:1 (A4)
ALL DIMENSIONS IN mm
APPROVED :


10
A-A
A
Ø4 TYP
20
R70
90°
A
Ø47.56
Ø50.80
MATERIAL: ALLUMINIUM
STOCK ITEM FROM AUTOBEND
DEVICE COVER - BENT TUBE
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT
DRAWN : KETAN AND JAMES
DATE : MAY 2005
CHECKED :
DRG. No : 3
SCALE : 1:2 (A4)
ALL DIMENSIONS IN mm
APPROVED :


255
A
A
Ø47.56
Ø50.80
40
Ø4 TYP
Ø12
10
A-A (1 : 2)
MATERIAL: ALLUMINIUM PIPE
STOCK 2INCH EXTERNAL Ø
DEVICE COVER - STRAIGHT PIPE
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.
DRAWN : KETAN AND JAMES
DATE : MAY 2005
CHECKED :
DRG. No : 4
SCALE : 1:2 (A4)
ALL DIMENSIONS IN mm
APPROVED :


32
R16
32
SHORT PIPE ELBOW
Ø6.35
(Ø 1/4 INCH)
60
R16
80
LONG PIPE ELBOW
Ø6.35
(Ø1/4 INCH)
MATERIAL: STAINLESS STEEL PIPE
SUPPLIED BY FLUID SYSTEMS TECHNOLOGIES LTD
1/4 INCH PIPE ELBOWS
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT. CH CH.
DRAWN : KETAN AND JAMES
DATE : MAY 2005
CHECKED :
DRG. No : 5
SCALE : 1:1 (A4)
ALL DIMENSIONS IN mm
APPROVED :

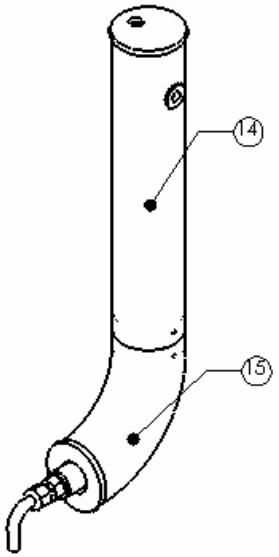

Ø47.60 0 -0.1
Ø6
MOUNT MUST BE SNUG FIT
INTO ALLUMINIUM CASING
7
Ø3.5 TYP
TAP M4 x 0.5 THROUGH
A
30
A
5 ±0.1
4.85
10
10
5 ±0.1
Ø12
A-A (1 : 1)
CEELEON PTFE
EXTRUDED ROD
(TEFLON)
EXTERNAL PIPE MOUNT
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.
DRAWN : KETAN AND JAMES
DATE : MAY 2005
CHECKED :
DRG. No : 6
SCALE : 1:1 (A4)
ALL DIMENSIONS IN mm
APPROVED :



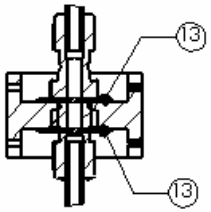
MATERIAL: STAINLESS STEEL
PIPE SECTION STOCK 1/4 INCH FROM FLUID SYSTEMS LTD

DEVICE MOUNT WASHER AND STRAIGHT PIPE SECTION	UNIVERSITY OF CANTERBURY MECHANICAL ENGINEERING DEPT. CH CH. N.Z.	
	DRAWN : KETAN AND JAMES	DATE : MAY 2005
	CHECKED :	DRG. No : 7
SCALE : 1:2 (A4) ALL DIMENSIONS IN mm	APPROVED :	

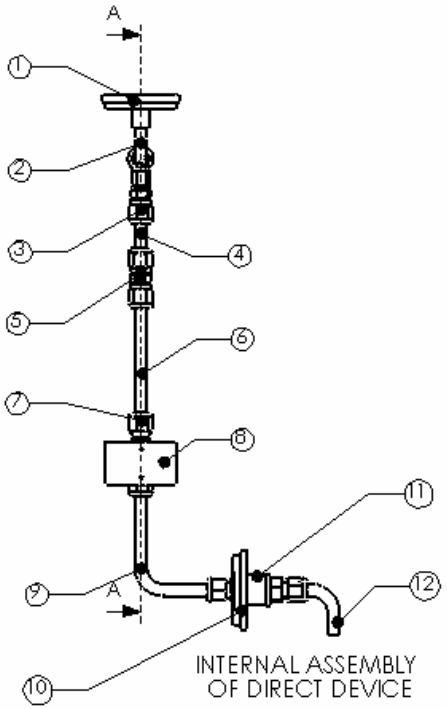
NUMBER	QNTY	NAME	MATL	STANDARD PART	SUPPLIER	DRW REF
1	1	TOP END CAP	PTFE	NO	LUDWICI	2
2	1	SAMPLE INTAKE	SS	NO	FLUID SYS	1
3	1	REDUCING UNION	SS	YES	FLUID SYS	N/A
4	1	CAPILARY	SS	YES	SYFT	N/A
5	1	UNION	SS	YES	FLUID SYS	N/A
6	1	PIPE	SS	NO	FLUID SYS	7
7	1	BULKHEAD	SS	YES	FLUID SYS	N/A
8	1	DEVICE MOUNT	PTFE	NO	LUDWICI	6
9	1	LONG PIPE ELBOW	SS	NO	FLUID SYS	5
10	1	BOTTOM END CAP	PTFE	NO	LUDWICI	2
11	1	QUICK CONNECT	SS	YES	FLUID SYS	N/A
12	1	SHORT PIPE ELBOW	SS	NO	FLUID SYS	5
13	2	DEVICE MOUNT WASHER	SS	NO	MICO METALS	7
14	1	STRAIGHT DEVICE CASE	AL	NO	MICO METALS	4
15	1	BENT DEVICE CASE	AL	NO	MICO METALS	3



DIRECT DEVICE
WITH ALLUMINIUM CASING



DETAILED SECTION VIEW
AA (1:2)



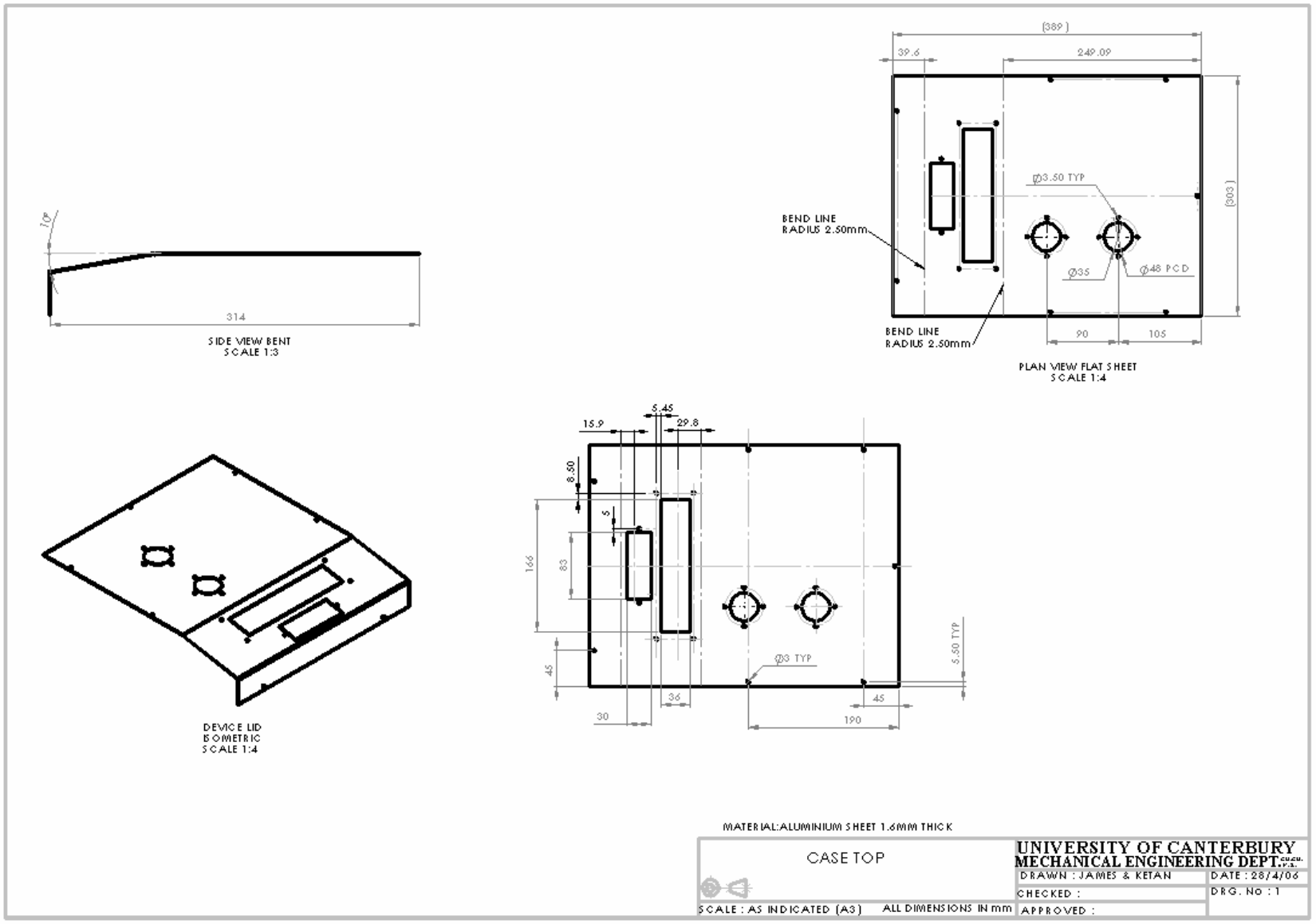
INTERNAL ASSEMBLY
OF DIRECT DEVICE

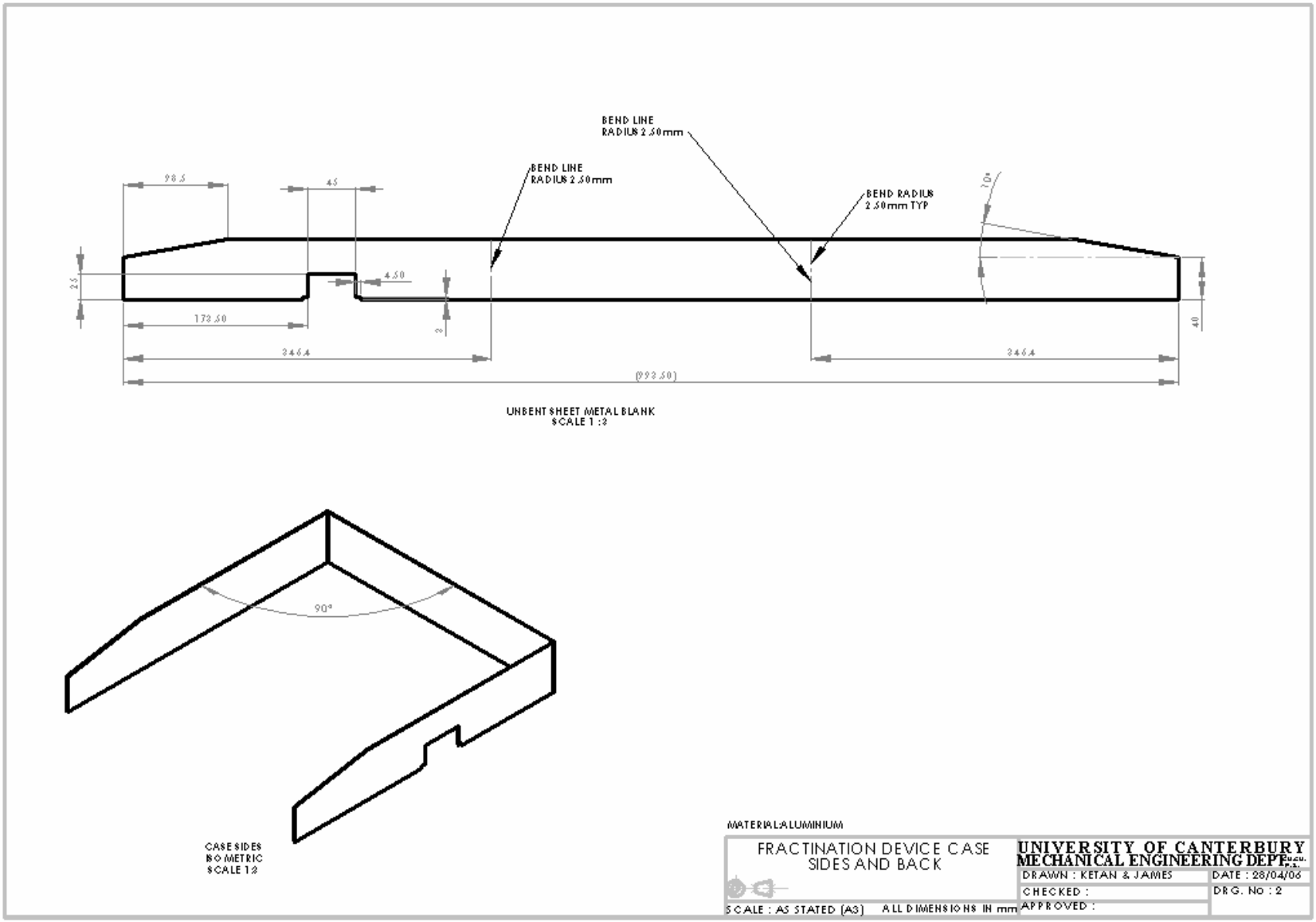
DEVICE ASSEMBLY AND PARTS

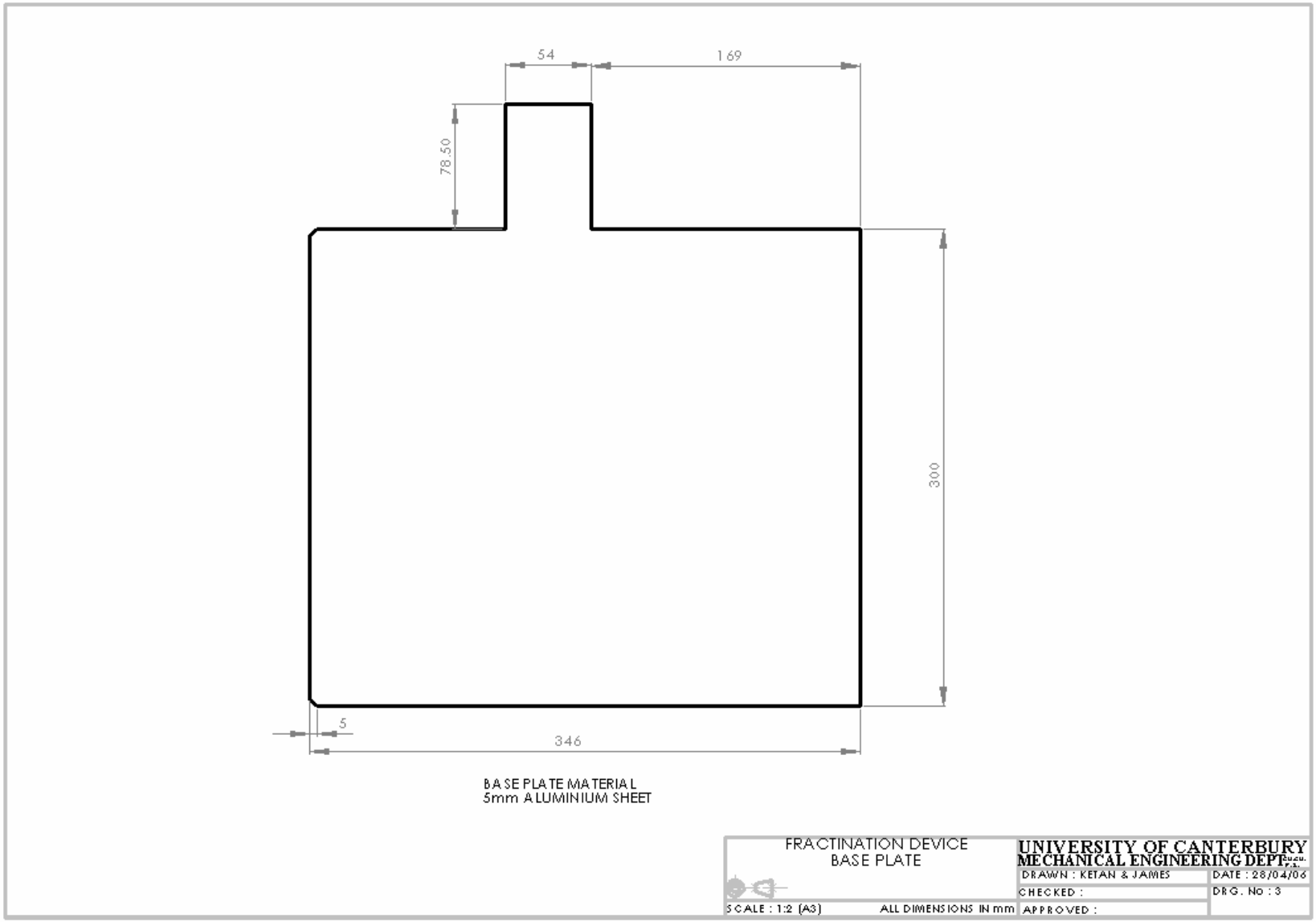
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.

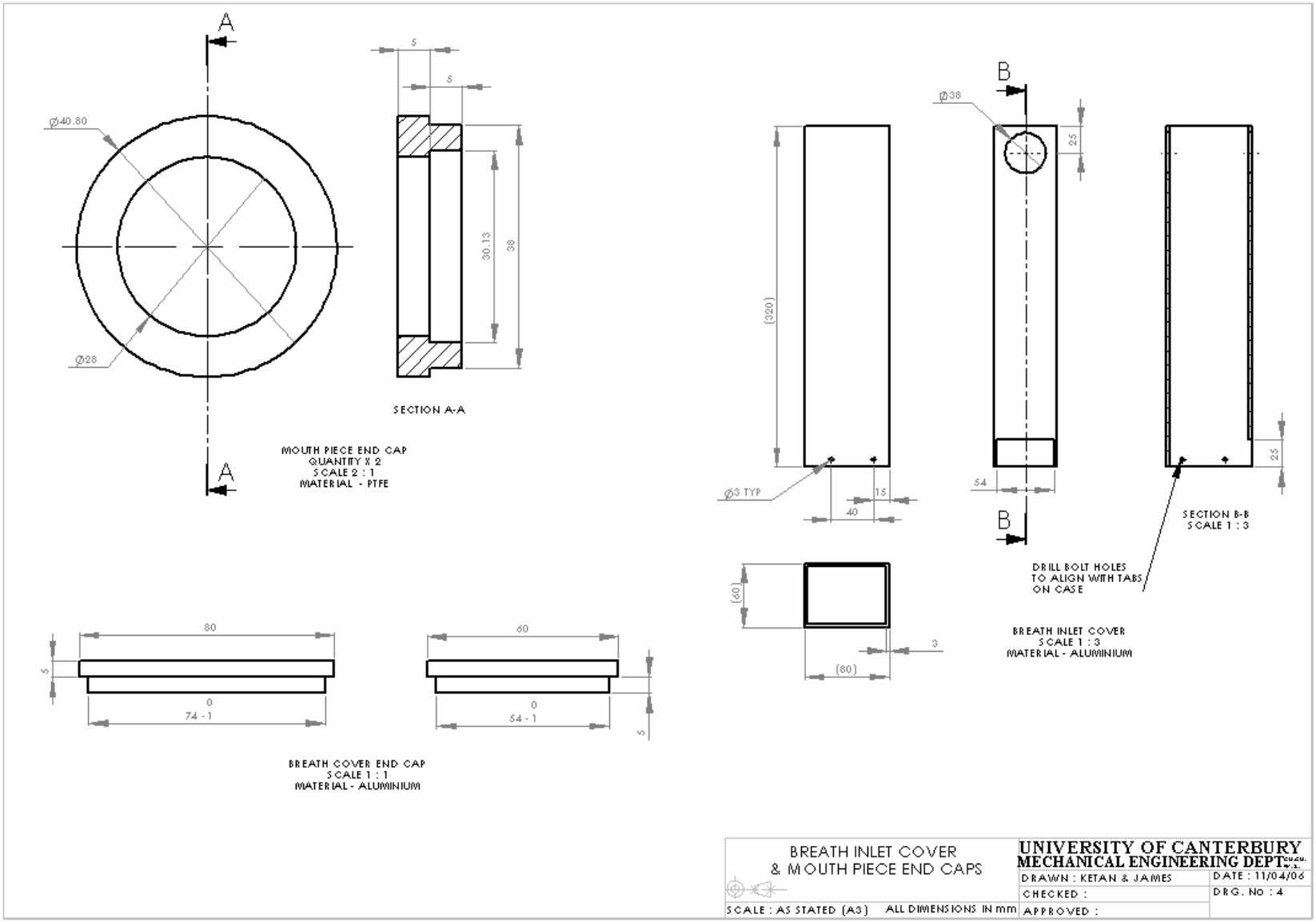
DRAWN : KETAN AND JAMES	DATE : MAY 2005
CHECKED :	DRG.No : 8
SCALE : 1:4 (A4) ALL DIMENSIONS IN mm	APPROVED :

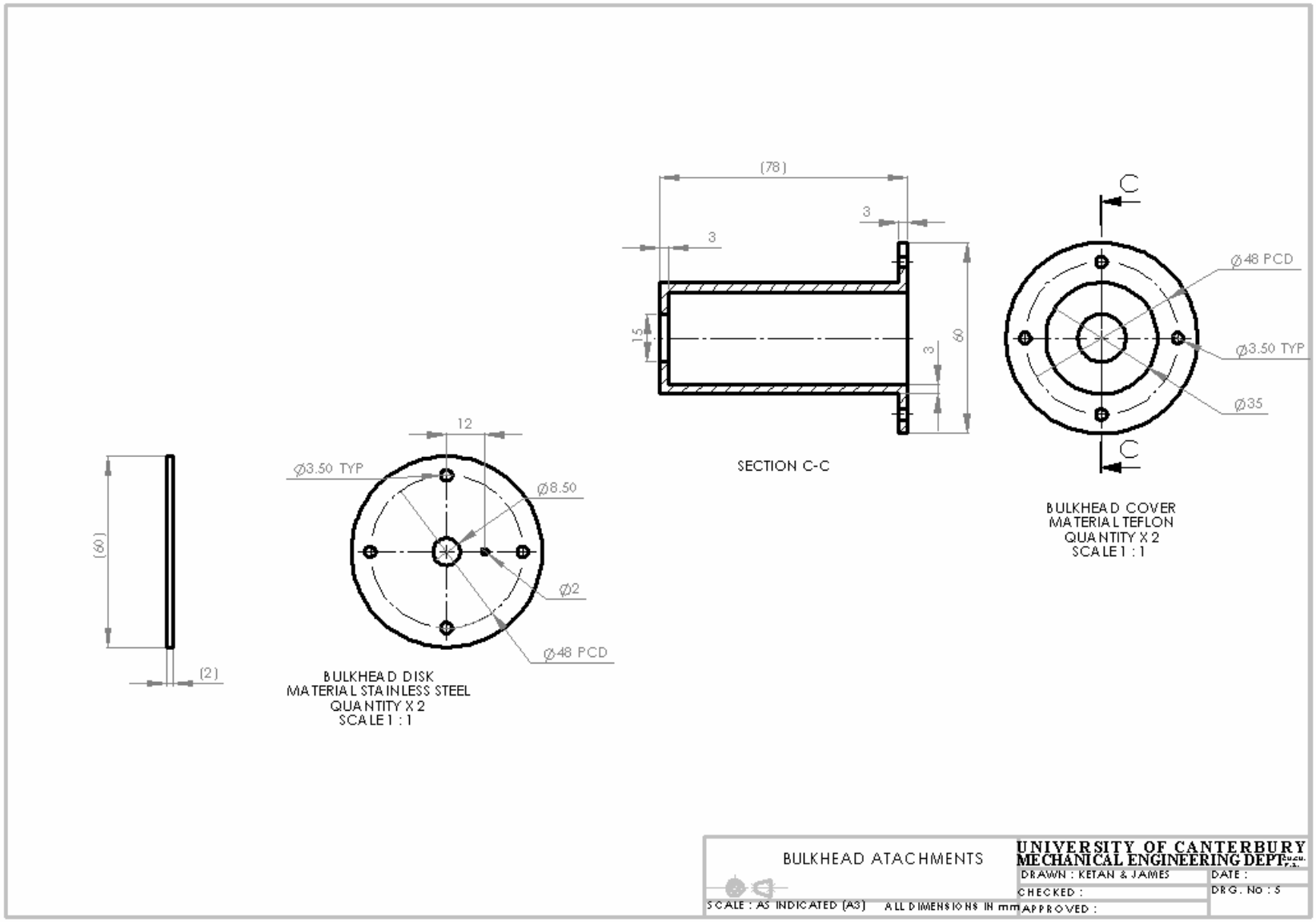
B1 Remote Breath Collection Device Drawings

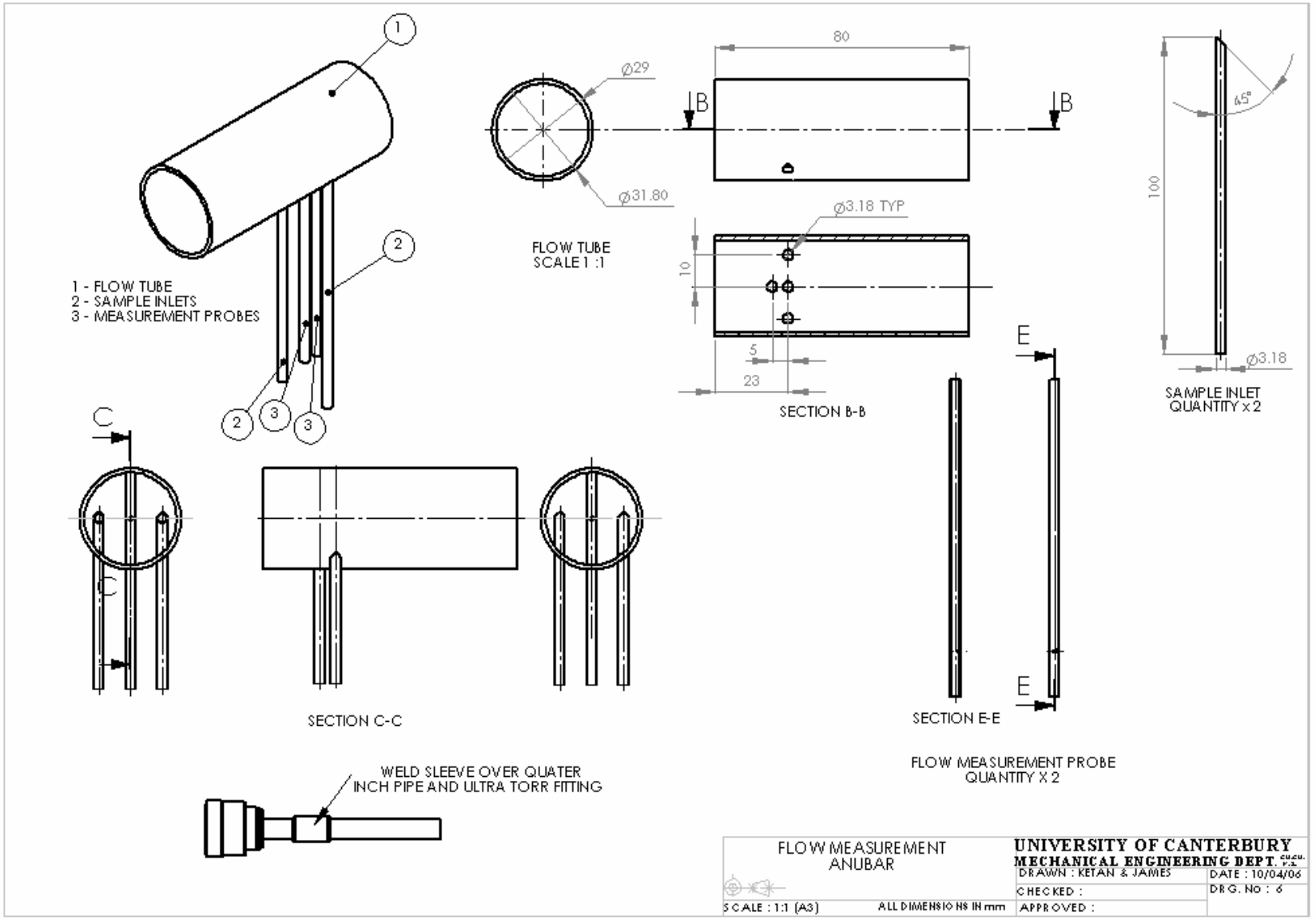

10°
314
SIDE VIEW BENT
SCALE 1:3
(389)
39.6
249.09
(303)
Ø3.50 TYP
BEND LINE
RADIUS 2.50mm
Ø35
Ø48 PCD
BEND LINE
RADIUS 2.50mm
90
105
PLAN VIEW FLAT SHEET
SCALE 1:4
DEVICE LID
ISOMETRIC
SCALE 1:4
5.45
15.9
29.8
8.50
5
166
83
45
Ø3 TYP
5.50 TYP
36
30
45
190
MATERIAL:ALUMINIUM SHEET 1.6MM THICK
CASE TOP
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.
DRAWN : JAMES & KETAN
DATE : 28/4/06
CHECKED :
DRG. No : 1
SCALE : AS INDICATED (A3)
ALL DIMENSIONS IN mm
APPROVED :

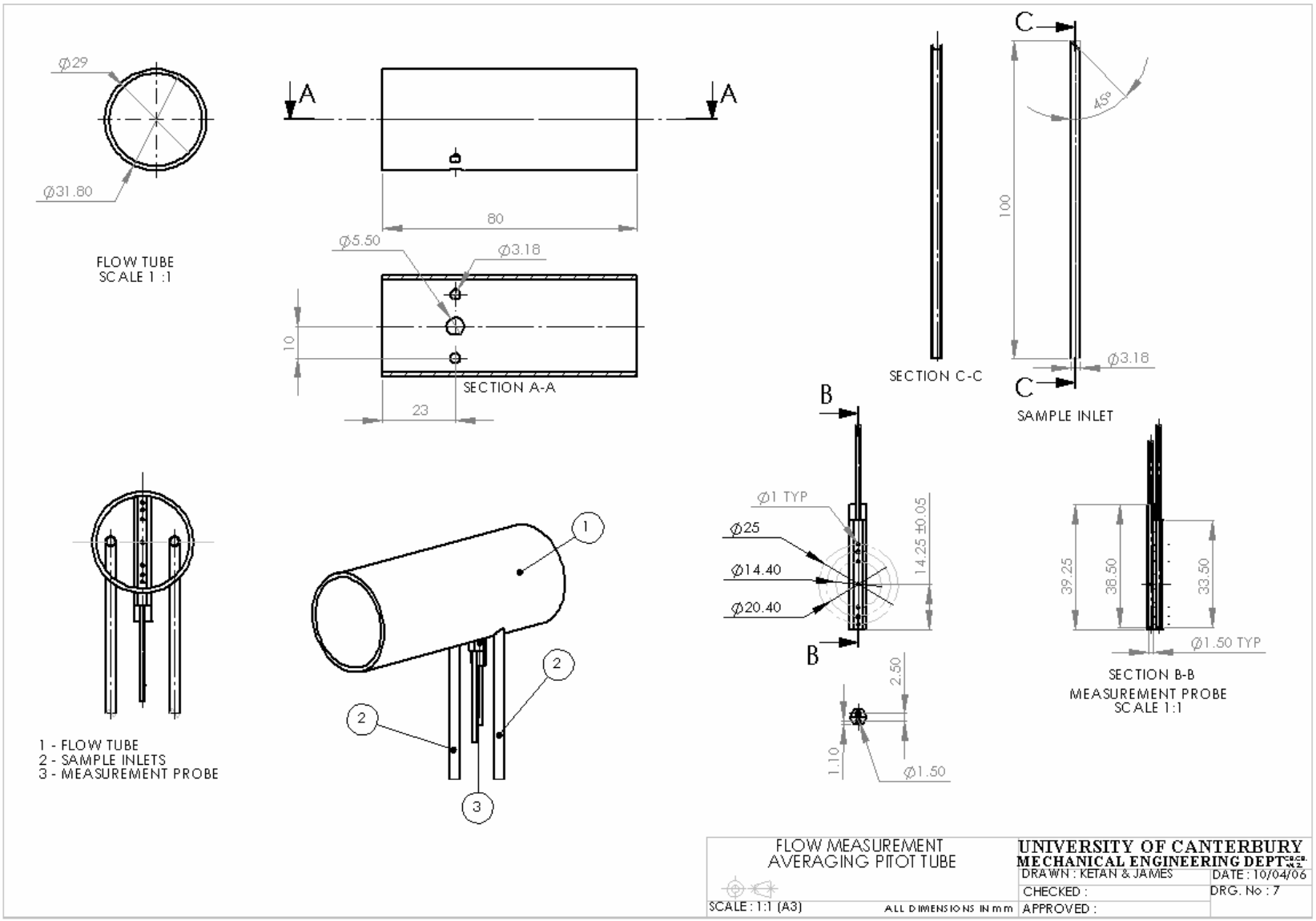

BEND LINE
RADIUS 2.50mm
BEND LINE
RADIUS 2.50mm
BEND RADIUS
2.50mm TYP
98.5
45
4.50
2
172.50
346.4
346.4
(992.50)
10°
25
40
UNBENT SHEET METAL BLANK
SCALE 1 : 2
90°
CASE SIDES
NO METRIC
SCALE 1:2
MATERIAL: ALUMINIUM
FRACTINATION DEVICE CASE
SIDES AND BACK
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.
DRAWN : KETAN & JAMES
DATE : 28/04/06
CHECKED :
DRG. No : 2
APPROVED :
SCALE : AS STATED (A3)
ALL DIMENSIONS IN mm

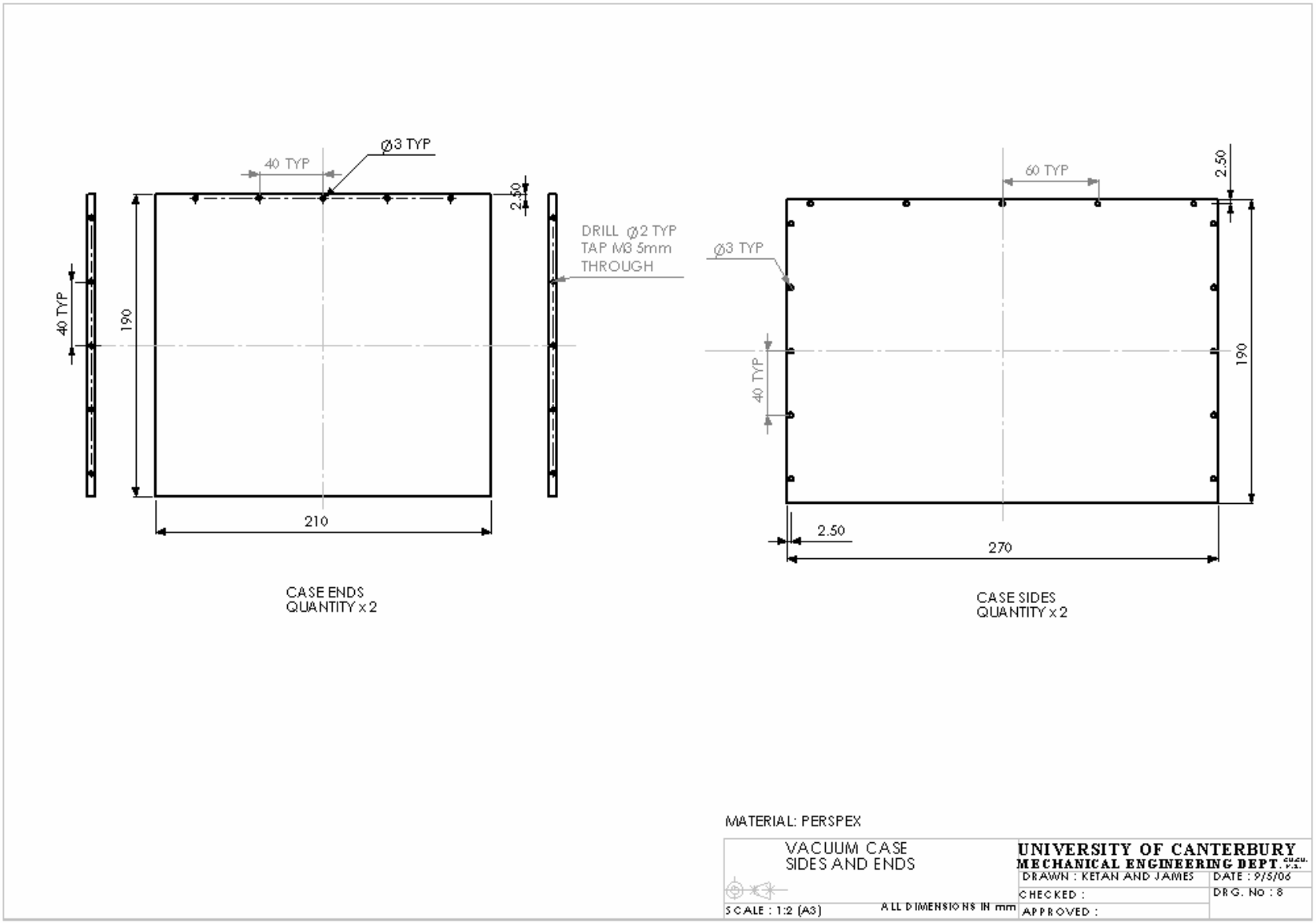

54
169
78.50
300
5
346
BASE PLATE MATERIAL
5mm ALUMINIUM SHEET
FRACTINATION DEVICE
BASE PLATE
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT
DRAWN : KETAN & JAMES
DATE : 28/04/06
CHECKED :
DRG. No : 3
SCALE : 1:2 (A3)
ALL DIMENSIONS IN mm
APPROVED :

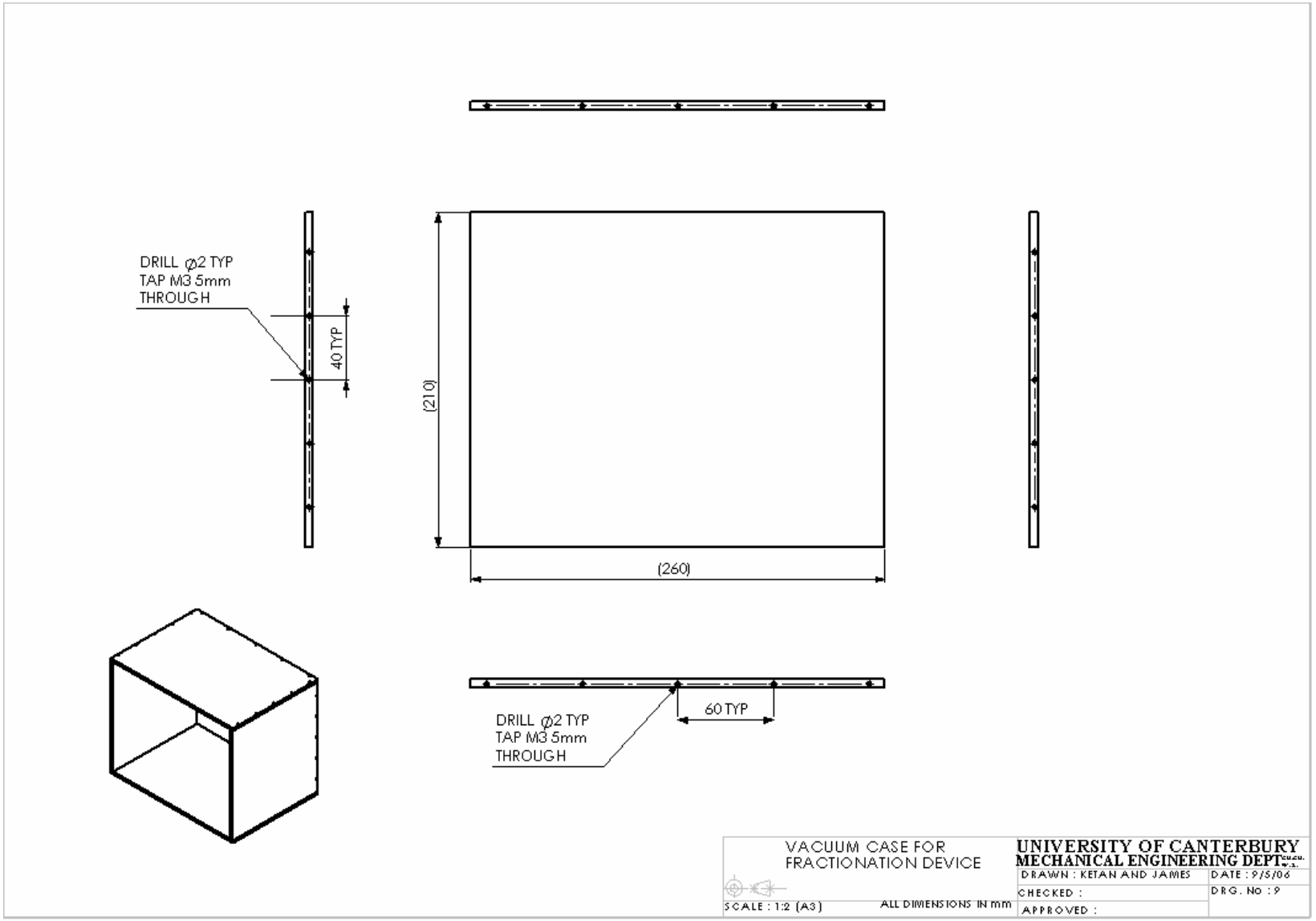

A
A
Ø40.80
Ø28
5
5
30.13
38
SECTION A-A
MOUTH PIECE END CAP
QUANTITY X 2
SCALE 2 : 1
MATERIAL - PTFE
B
B
Ø38
25
(320)
Ø3 TYP
15
40
54
25
SECTION B-B
SCALE 1 : 3
DRILL BOLT HOLES
TO ALIGN WITH TABS
ON CASE
(40)
3
(80)
BREATH INLET COVER
SCALE 1 : 3
MATERIAL - ALUMINIUM
80
5
0
74 - 1
60
0
54 - 1
5
BREATH COVER END CAP
SCALE 1 : 1
MATERIAL - ALUMINIUM
BREATH INLET COVER
& MOUTH PIECE END CAPS
SCALE : AS STATED (A3)
ALL DIMENSIONS IN mm
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT
DRAWN : KETAN & JAMES
DATE : 11/04/06
CHECKED :
DRG. No : 4
APPROVED :

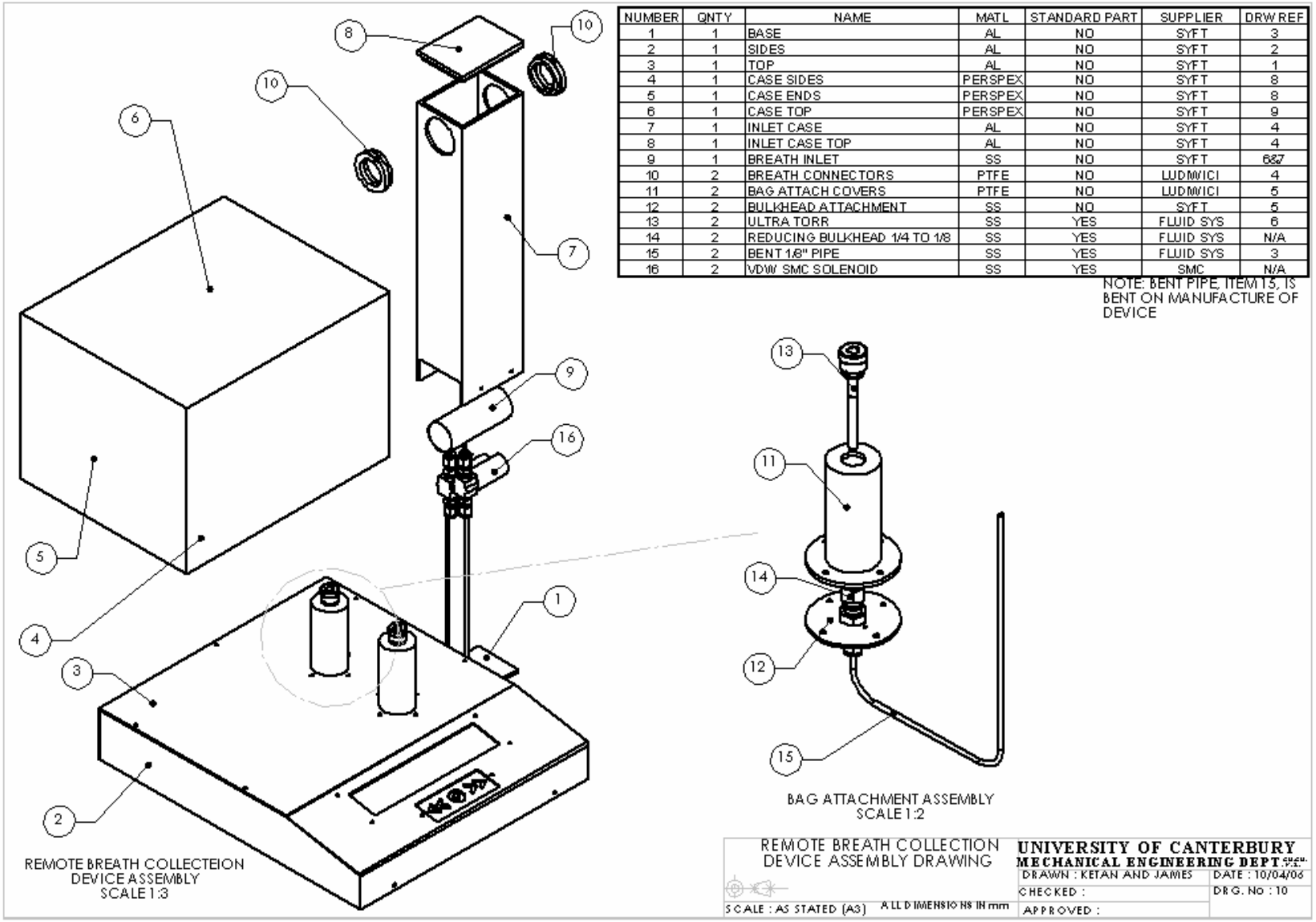

(78)
3
3
15
3
60
SECTION C-C
C
C
Ø48 PCD
Ø3.50 TYP
Ø35
BULKHEAD COVER
MATERIAL TEFLON
QUANTITY X 2
SCALE 1 : 1
12
Ø3.50 TYP
Ø8.50
Ø2
Ø48 PCD
BULKHEAD DISK
MATERIAL STAINLESS STEEL
QUANTITY X 2
SCALE 1 : 1
(60)
(2)
BULKHEAD ATACHMENTS
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT
DRAWN : KETAN & JAMES
DATE :
CHECKED :
DRG. No : 5
APPROVED :
SCALE : AS INDICATED (A3)
ALL DIMENSIONS IN mm


1 - FLOW TUBE
2 - SAMPLE INLETS
3 - MEASUREMENT PROBES
FLOW TUBE
SCALE 1 :1
Ø29
Ø31.80
80
B
B
Ø3.18 TYP
10
5
23
SECTION B-B
45°
100
Ø3.18
SAMPLE INLET
QUANTITY x 2
C
C
SECTION C-C
E
E
SECTION E-E
FLOW MEASUREMENT PROBE
QUANTITY X 2
WELD SLEEVE OVER QUATER
INCH PIPE AND ULTRA TORR FITTING
FLOW MEASUREMENT
ANUBAR
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.
DRAWN : KETAN & JAMES
DATE : 10/04/06
CHECKED :
DRG. No : 6
APPROVED :
SCALE : 1:1 (A3)
ALL DIMENSIONS IN mm


Ø29
Ø31.80
FLOW TUBE
SCALE 1 :1
A
A
80
Ø5.50
Ø3.18
10
23
SECTION A-A
SECTION C-C
C
C
100
45°
Ø3.18
SAMPLE INLET
B
B
Ø1 TYP
Ø25
Ø14.40
Ø20.40
14.25 ±0.05
2.50
1.10
Ø1.50
39.25
38.50
33.50
Ø1.50 TYP
SECTION B-B
MEASUREMENT PROBE
SCALE 1:1
1 - FLOW TUBE
2 - SAMPLE INLETS
3 - MEASUREMENT PROBE
1
2
2
3
FLOW MEASUREMENT
AVERAGING PITOT TUBE
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT
DRAWN : KETAN & JAMES
DATE : 10/04/06
CHECKED :
DRG. No : 7
SCALE : 1:1 (A3)
ALL DIMENSIONS IN mm
APPROVED :


Ø3 TYP
40 TYP
2.50
DRILL Ø2 TYP
TAP M3 5mm
THROUGH
40 TYP
190
210
CASE ENDS
QUANTITY x 2
60 TYP
2.50
Ø3 TYP
40 TYP
190
2.50
270
CASE SIDES
QUANTITY x 2
MATERIAL: PERSPEX
VACUUM CASE
SIDES AND ENDS
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT.
DRAWN : KETAN AND JAMES
DATE : 9/5/06
CHECKED :
DRG. No : 8
APPROVED :
SCALE : 1:2 (A3)
ALL DIMENSIONS IN mm


DRILL ⌀2 TYP
TAP M3 5mm
THROUGH
40 TYP
(210)
(260)
60 TYP
DRILL ⌀2 TYP
TAP M3 5mm
THROUGH
VACUUM CASE FOR
FRACTIONATION DEVICE
UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT
DRAWN : KETAN AND JAMES
DATE : 9/5/06
CHECKED :
DRG. No : 9
APPROVED :
SCALE : 1:2 (A3)
ALL DIMENSIONS IN mm



NUMBER	QNTY	NAME	MATL	STANDARD PART	SUPPLIER	DRW REF
1	1	BASE	AL	NO	SYFT	3
2	1	SIDES	AL	NO	SYFT	2
3	1	TOP	AL	NO	SYFT	1
4	1	CASE SIDES	PERSPEX	NO	SYFT	8
5	1	CASE ENDS	PERSPEX	NO	SYFT	8
6	1	CASE TOP	PERSPEX	NO	SYFT	9
7	1	INLET CASE	AL	NO	SYFT	4
8	1	INLET CASE TOP	AL	NO	SYFT	4
9	1	BREATH INLET	SS	NO	SYFT	6&7
10	2	BREATH CONNECTORS	PTFE	NO	LUDMVICI	4
11	2	BAG ATTACH COVERS	PTFE	NO	LUDMVICI	5
12	2	BULKHEAD ATTACHMENT	SS	NO	SYFT	5
13	2	ULTRA TORR	SS	YES	FLUID SYS	6
14	2	REDUCING BULKHEAD 1/4 TO 1/8	SS	YES	FLUID SYS	N/A
15	2	BENT 1/8" PIPE	SS	YES	FLUID SYS	3
16	2	VDW SMC SOLENOID	SS	YES	SMC	N/A

NOTE: BENT PIPE, ITEM 15, IS BENT ON MANUFACTURE OF DEVICE

REMOTE BREATH COLLECTION DEVICE ASSEMBLY DRAWING

UNIVERSITY OF CANTERBURY
MECHANICAL ENGINEERING DEPT

DRAWN : KETAN AND JAMES
DATE : 10/04/06
CHECKED :
DRG. No : 10
APPROVED :
SCALE : AS STATED (A3)
ALL DIMENSIONS IN mm

B2 Remote Breath Collection Device Code

```
//-------------------------------------------------------------------------
// C main line
// Written by Ketan Lad and James Neilson
// University of Canterbury, Department of Mechanical Engineering
// Syft Technologies Ltd
// May 2006
//-------------------------------------------------------------------------
#include "main.h"               // Include header file main.h required to run LCD display.
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

//=========================================================================================
// MASKS
#define QTSS 0x01
#define QTDRDY 0x40
#define TOUCHED 0x80
#define VALVE1 0x01
#define VALVE2 0x04
#define LED1 0x20
#define LED2 0x08
#define LED3 0x02
#define VOLTMASK 0x07FF
#define VOLTFLIP 0x0400
#define   CLICK 0x20
#define LEFT 0x10
#define RIGHT 0x01
#define ENTER 0x04
//=========================================================================================
// FUNCTIONS
void ReadHoneywell(void);
//void ReadWheel(void);
void DeleteScreenArrowsDataEntryStates(void);
void DeleteScreenArrowsPurgeStates(void);
//void DeleteScreenArrowsObtainAverageBreathStates(void);
void DeleteScreenArrowsCollectBreathStates(void);
void DataEntryStates(void);
void TimeDelay(void);
void PurgeStates(void);
void ObtainAverageBreath(void);
void BreathCollection(void);
void IndicateBusy(void);
void Summary(void);
void Sound(int pulseTime);
void ButtonPressCheck(void);
//void itoa();
//=========================================================================================
// Global Variables
int         volt = 0;
int         decimalPlace = 0;
char        placeVariable = 0;
//float     scratch = 0;
//float     strain =0;
//float     sniff = 0;
char        convertedVariable[10];
//int       FindDecimal(float sniff);
int         statusByte = 0;
float       timeInterval = 0.05;
char        measuredPoints[10];
int         masked = 0;
int         flipped = 0;
```

```
float       voltage = 0;
char        qWheelData = 0;
char        qWheelData2 = 0;
float       position = 0;
int         upperAirwayFractionStart = 0;
int         upperAirwayFractionEnd = 30;
int         lowerAirwayFractionStart = 40;
int         lowerAirwayFractionEnd = 100;
int         tolerance = 25;
//char      *pointVar;
int         soundPulse = 50;
int         variableNumber = 1;
int         i=0;
int         enterFlag = 0;
int         flashCounter = 0;
int         timeEnter = 3000;
int         timeState = 6000;
int         timeScroll = 7000;
char        changingVariable[3];
int                 purgeTime = 5;
unsigned int        j = 0;
int         backFlag = 0;
int         k = 0;
float       zeroVoltage = 0;
int         zeroVoltCount = 100;
float       AverageBreath = 0;
unsigned int        timeCounter = 0;
unsigned int        totalTime = 9;      //Clock rate for DELSIG11 is 24MHz/6*4 where 6 adn 4 are VC1 and
                                          VC2, DELSIG11 requires 1024 clock cycles to take one sample.
                                          Therefore (24e6/(6*4))/1024 = 976 samples a second. We read the
                                          deslsig 3 times to get a correct reading.
int                 button1 = 0;
int                 button2 = 0;
int         button3 = 0;                //we read a voltage at 976/3 = 325 readings per secound (Hz).
int                 breathsInTolerance = 0;
int                 breathsOutTolerance = 0;
int         counter = 0;
float       previousVoltage = 0;
int                 totalExhaledBreaths = 0;
float       runningTotalArea = 0;
float   integratedArea = 0;
float       time = 0.009;
int                 timeRead = 0;
int                 busyTime = 0;
float       sampleBreath1 = 0;
float       sampleBreath2 = 0;
float       sampleBreath3 = 0;
float       tolBreath1 = 0;
float       tolBreath2 = 0;
float       tolBreath3 = 0;
float       upperToleranceLimit = 0;
float       lowerToleranceLimit = 0;

void main()
{

    int statusByte = 0;
    M8C_EnableGInt;
            lcd_e = 0x90;
            LCD_1_Start();
    PGA_1_Start(PGA_1_HIGHPOWER);
    DELSIG11_1_Start(DELSIG11_1_HIGHPOWER);
            DELSIG11_1_StartAD();
                                        // Start DELSIG11 and AD of the deslsig

            lcd_e = 0x10;
//=========================================================================
```

```
================================================
//WELCOME MESSAGE
lcd_e = 0x10;
LCD_1_Position(0,6);
LCD_1_PrCString("Welcome to Syft Technologies");
LCD_1_Position(1,6);
LCD_1_PrCString("Breath Collection Apparatus");
lcd_e = 0x80;
LCD_1_Position(1,4);
LCD_1_PrCString("By James Neilson & Ketan Lad 2006");

for (i=0;i<=5;i++)
        {TimeDelay();}

// ================================================================
Purge:
        PurgeStates();
// Reset variables below before going to Data entry states.
totalExhaledBreaths = 0;
breathsOutTolerance = 0;
breathsInTolerance = 0;

DataEntry:
        DataEntryStates();

        if (backFlag == 1)          // Backflag is used by the program to go back to previous state.
                {
                backFlag = 0;
                goto Purge;
                }

ObtainAverageProfile:
        ObtainAverageBreath();

        if (backFlag == 1)
                {
                backFlag = 0;
                goto DataEntry;
                }


CollectBreath:
BreathCollection();
        if (backFlag == 1)
                {
                backFlag = 0;
                goto ObtainAverageProfile;
                }

Summary();
        if (backFlag == 1)
                {
                backFlag = 0;
                goto CollectBreath;
                }
goto Purge;
}       // end of main

//================================================================================
==============================
// Miscelenous Small Functions
void TimeDelay(void)
{
        for (j=0;j<33000;j++);
}

void DeleteScreenArrowsDataEntryStates(void)
```

```
{
        lcd_e = 0x10;
        LCD_1_Position(1,12);
        LCD_1_PrCString(" ");
        LCD_1_Position(1,19);
        LCD_1_PrCString(" ");
        LCD_1_Position(1,30);
        LCD_1_PrCString(" ");
        lcd_e = 0x80;
        LCD_1_Position(0,12);
        LCD_1_PrCString(" ");
        LCD_1_Position(0,19);
        LCD_1_PrCString(" ");
        LCD_1_Position(1,37);
        LCD_1_PrCString(" ");
        LCD_1_Position(1,2);
        LCD_1_PrCString(" ");
        return;
}

void DeleteScreenArrowsPurgeStates(void)
{
        lcd_e = 0x10;
        LCD_1_Position(1,24);
        LCD_1_PrCString(" ");
        LCD_1_Position(0,39);
        LCD_1_PrCString(" "); //Include the busy indication character
        lcd_e = 0x80;
        LCD_1_Position(0,24);
        LCD_1_PrCString(" ");
        LCD_1_Position(1,37);
        LCD_1_PrCString(" ");
                return;
}

void DeleteScreenArrowsCollectBreathStates(void)
{
        lcd_e = 0x80;
        LCD_1_Position(1,2);
        LCD_1_PrCString(" ");
        LCD_1_Position(1,37);
        LCD_1_PrCString(" ");
        return;
}

void ButtonPressCheck(void)
{
// Sub to read the port registers of each of the three buttons.
        button1 = PRT0DR & RIGHT;
        button2 = PRT0DR & LEFT;
        button3 = PRT0DR & ENTER;

}

void Wait(void)
{
        for(i=0;i<=timeState;i++);
}


//=============================================================================
==================
// PURGE SCREEN SETUP
void PurgeStates(void)
{
// Setup the Screen ========================================
lcd_e = 0x90;
```

```
LCD_1_Control(0x01);
lcd_e = 0x10;
LCD_1_Position(0,2);
LCD_1_PrCString("MODE: PURGE & CONTINOUS COLLECTION");
LCD_1_Position(1,2);
LCD_1_PrCString("Purge/Collection Time     secs");

itoa(changingVariable,purgeTime,10);
LCD_1_Position(1,25);
LCD_1_PrString(changingVariable);

lcd_e = 0x80;
LCD_1_Position(1,38);
LCD_1_PrCString(">>");
//============================================================
Timer16_1_WritePeriod(3906);

        State1:
                Timer16_1_DisableInt();
                Timer16_1_Stop();
                oneSec = 0;

                lcd_e = 0x80;
                LCD_1_Position(0,25);
                LCD_1_PrCString("START");
                DeleteScreenArrowsPurgeStates();
                lcd_e = 0x80;
                LCD_1_Position(0,24);
                LCD_1_WriteData(0x7E);
                Wait();
                //PWM8_WritePulseWidth(0);
                //ReadWheel();
                //if ((qWheelData & TOUCHED) == TOUCHED )
                //          {
                ButtonPressCheck();
                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State3;
                        }

                if (button2 == 0x10)          // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button3 == 0x04)                    // Select variable to change.
                        {
                        Sound(soundPulse);
                        timeRead = 0;
                        goto State4;
                        }
        goto State1;

        State2:

                DeleteScreenArrowsPurgeStates();
                lcd_e = 0x10;
                LCD_1_Position(1,24);
                LCD_1_WriteData(0x7E);
                Wait();
                ButtonPressCheck();
                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State1;
```

```
                }

        if (button2 == 0x10)        // Go back one state.
                {
                Sound(soundPulse);
                goto State3;
                }

        if (button3 == 0x04)                // Select variable to change.
                {
                Sound(soundPulse);
                goto State5;
                }
goto State2;

State3:

        DeleteScreenArrowsPurgeStates();
        lcd_e = 0x80;
        LCD_1_Position(1,37);
        LCD_1_WriteData(0x7E);
        Wait();
        ButtonPressCheck();
        if (button1 == 0x01)                // Go forward one state.
                {
                Sound(soundPulse);
                goto State2;
                }

        if (button2 == 0x10)        // Go back one state.
                {
                Sound(soundPulse);
                goto State1;
                }

        if (button3 == 0x04)                // Select variable to change.
                {
                Sound(soundPulse);
                return;
                }
goto State3;

State4:
        PRT2DR = PRT2DR | VALVE1;
PRT2DR = PRT2DR | VALVE2;
        DeleteScreenArrowsPurgeStates();
        lcd_e = 0x80;
        LCD_1_Position(0,24);
        LCD_1_WriteData(0x7E);
        Wait();
        lcd_e = 0x80;
        LCD_1_Position(0,25);
        LCD_1_PrCString("STOP ");

        Timer16_1_EnableInt();
Timer16_1_Start();
        oneSec = 0;
        while (!(timeRead == purgeTime))
        {
                if (oneSec == 1)
                        {
                        oneSec = 0;
                        timeRead = timeRead + 1;
                        busyTime = busyTime +1;
                        IndicateBusy();
                        }
```

```
                                        button3 = PRT0DR & ENTER;
                                        if (button3 == 0x04)                    // Select variable to change.
                                                {
                                                PRT2DR = PRT2DR & ~VALVE1;
                                        PRT2DR = PRT2DR & ~VALVE2;
                                        for(k=0;k<=timeState;k++);
                                                Sound(soundPulse);
                                                goto State1;
                                                }
                        }
                PRT2DR = PRT2DR & ~VALVE1;
        PRT2DR = PRT2DR & ~VALVE2;
                goto State1;

                State5:
                        DeleteScreenArrowsPurgeStates();
                        lcd_e = 0x10;
                        LCD_1_Position(1,24);
                        LCD_1_WriteData(0xA5);
                        Wait();
                        ButtonPressCheck();
                        if (button3 == 0x04)                                            // Enter button.
                                {
                                Sound(soundPulse);
                                goto State2;
                                }

                        if (button1 == 0x01)                            // Increment upperAirwayFractionEnd
                                {
                                Sound(soundPulse);
                                purgeTime = purgeTime +1;
                                if (purgeTime == 1000) //16.7 mins max time
                                        purgeTime = 0;
                                }
                        if (button2 == 0x10)                    //Decrement upperAirwayFractionEnd
                                {
                                Sound(soundPulse);
                                purgeTime = purgeTime -1;
                                if (purgeTime == -1)
                                        purgeTime = 0;
                                }

                                lcd_e = 0x10;
                                LCD_1_Position(1,25);
                                LCD_1_PrCString("   ");
                                itoa(changingVariable,purgeTime,10);
                                LCD_1_Position(1,25);
                                LCD_1_PrString(changingVariable);

                goto State5;

}
//=============================================================================
================================================
//DATA ENTRY SEQUENCE
void DataEntryStates(void)
{
// Set up the screen ============================
lcd_e = 0x90;
LCD_1_Control(0x01);

lcd_e = 0x10;
LCD_1_Position(0,2);
LCD_1_PrCString("MODE: DATA ENTRY");

LCD_1_Position(1,2);
LCD_1_PrCString("Fraction 1");
```

```
LCD_1_Position(1,16);
LCD_1_PrCString("% -");

LCD_1_Position(1,34);
LCD_1_PrCString("%");

LCD_1_Position(1,23);
LCD_1_PrCString("%   Tol");
//==================STATE 8
itoa(changingVariable,upperAirwayFractionStart,10);
if (upperAirwayFractionStart < 10)
        LCD_1_Position(1,15);
if (upperAirwayFractionStart < 100 && upperAirwayFractionStart >= 10)
        LCD_1_Position(1,14);
if (upperAirwayFractionStart == 100)
        LCD_1_Position(1,13);
LCD_1_PrString(changingVariable);
//==================STATE 9
itoa(changingVariable,upperAirwayFractionEnd,10);
if (upperAirwayFractionEnd < 10)
        LCD_1_Position(1,22);
if (upperAirwayFractionEnd < 100  && upperAirwayFractionEnd >= 10)
        LCD_1_Position(1,21);
if (upperAirwayFractionEnd == 100)
        LCD_1_Position(1,20);
LCD_1_PrString(changingVariable);
//==================STATE 12
itoa(changingVariable,tolerance,10);
if (tolerance < 10)
        LCD_1_Position(1,33);
if (tolerance < 100  && tolerance >= 10)
        LCD_1_Position(1,32);
if (tolerance == 100)
        LCD_1_Position(1,31);
LCD_1_PrString(changingVariable);

lcd_e = 0x80;
LCD_1_Position(0,11);
LCD_1_PrCString("2");

LCD_1_Position(0,16);
LCD_1_PrCString("% -");

LCD_1_Position(0,23);
LCD_1_PrCString("%");

LCD_1_Position(1,0);
LCD_1_PrCString("<<");

LCD_1_Position(1,38);
LCD_1_PrCString(">>");
//==================STATE 10
itoa(changingVariable,lowerAirwayFractionStart,10);
if (lowerAirwayFractionStart < 10)
        LCD_1_Position(0,15);
if (lowerAirwayFractionStart < 100  && lowerAirwayFractionStart >= 10)
        LCD_1_Position(0,14);
if (lowerAirwayFractionStart == 100)
        LCD_1_Position(0,13);
LCD_1_PrString(changingVariable);
//==================STATE 11
itoa(changingVariable,lowerAirwayFractionEnd,10);
if (lowerAirwayFractionEnd < 10)
        LCD_1_Position(0,22);
if (lowerAirwayFractionEnd < 100  && lowerAirwayFractionEnd >= 10)
        LCD_1_Position(0,21);
```

```
if (lowerAirwayFractionEnd == 100)
        LCD_1_Position(0,20);
LCD_1_PrString(changingVariable);



//=====================================
        State1:

                DeleteScreenArrowsDataEntryStates();
                lcd_e = 0x10;
                LCD_1_Position(1,12);
                LCD_1_WriteData(0x7E);
                Wait();
                ButtonPressCheck();

                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button2 == 0x10)            // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State7;
                        }

                if (button3 == 0x04)                    // Select variable to change.
                        {
                        Sound(soundPulse);
                        goto State8;
                        }

        goto State1;

        State2:
                DeleteScreenArrowsDataEntryStates();
                lcd_e = 0x10;
                LCD_1_Position(1,19);
                LCD_1_WriteData(0x7E);
                Wait();
                ButtonPressCheck();
                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State3;
                        }

                if (button2 == 0x10)            // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State1;
                        }

                if (button3 == 0x04)                    // Select variable to change.
                        {
                        Sound(soundPulse);
                        goto State9;
                        }
        goto State2;


        State3:
                DeleteScreenArrowsDataEntryStates();
                lcd_e = 0x80;
```

```
		LCD_1_Position(0,12);
		LCD_1_WriteData(0x7E);
		Wait();
		ButtonPressCheck();
		if (button1 == 0x01)			// Go forward one state.
			{
			Sound(soundPulse);
			goto State4;
			}

		if (button2 == 0x10)		// Go back one state.
			{
			Sound(soundPulse);
			goto State2;
			}

		if (button3 == 0x04)			// Select variable to change.
			{
			Sound(soundPulse);
			goto State10;
			}
goto State3;

State4:
		DeleteScreenArrowsDataEntryStates();
		lcd_e = 0x80;
		LCD_1_Position(0,19);
		LCD_1_WriteData(0x7E);
		Wait();
		ButtonPressCheck();
		if (button1 == 0x01)			// Go forward one state.
			{
			Sound(soundPulse);
			goto State5;
			}

		if (button2 == 0x10)		// Go back one state.
			{
			Sound(soundPulse);
			goto State3;
			}

		if (button3 == 0x04)			// Select variable to change.
			{
			Sound(soundPulse);
			goto State11;
			}
goto State4;

State5:
		DeleteScreenArrowsDataEntryStates();
		lcd_e = 0x10;
		LCD_1_Position(1,30);
		LCD_1_WriteData(0x7E);
		Wait();
		ButtonPressCheck();
		if (button1 == 0x01)			// Go forward one state.
			{
			Sound(soundPulse);
			goto State6;
			}

		if (button2 == 0x10)		// Go back one state.
			{
			Sound(soundPulse);
			goto State4;
			}
```

```
		if (button3 == 0x04)			// Select variable to change.
			{
			Sound(soundPulse);
			goto State12;
			}
goto State5;

State6:
		DeleteScreenArrowsDataEntryStates();
		lcd_e = 0x80;
		LCD_1_Position(1,37);
		LCD_1_WriteData(0x7E);
		Wait();
		ButtonPressCheck();
		if (button1 == 0x01)			// Go forward one state.
			{
			Sound(soundPulse);
			goto State7;
			}

		if (button2 == 0x10)		// Go back one state.
			{
			Sound(soundPulse);
			goto State5;
			}

		if (button3 == 0x04)			// Select variable to change.
			{
			Sound(soundPulse);
			return;
			}
goto State6;

State7:
		DeleteScreenArrowsDataEntryStates();
		lcd_e = 0x80;
		LCD_1_Position(1,2);
		LCD_1_WriteData(0x7F);
		Wait();
		ButtonPressCheck();
		if (button1 == 0x01)			// Go forward one state.
			{
			Sound(soundPulse);
			goto State1;
			}
		if (button2 == 0x10)		// Go back one state.
			{
			Sound(soundPulse);
			goto State6;
			}

		if (button3 == 0x04)			// Return to Main.
			{
			Sound(soundPulse);
			backFlag = 1;
			return;
			}
goto State7;

State8:
		DeleteScreenArrowsDataEntryStates();
		lcd_e = 0x10;
		LCD_1_Position(1,12);
		LCD_1_WriteData(0xA5);
		Wait();
		ButtonPressCheck();
```

```
                if (button3 == 0x04)                            // Enter button.
                        {
                        if (upperAirwayFractionEnd < upperAirwayFractionStart)
                                {
                                upperAirwayFractionEnd = upperAirwayFractionStart;
                                lcd_e = 0x10;
                                LCD_1_Position(1,20);
                                LCD_1_PrCString("   ");
                                itoa(changingVariable,upperAirwayFractionEnd,10);

                                if (upperAirwayFractionEnd < 10)
                                        LCD_1_Position(1,22);

                                if (upperAirwayFractionEnd < 100   && upperAirwayFractionEnd >=
10)
                                        LCD_1_Position(1,21);

                                if (upperAirwayFractionEnd == 100)
                                        LCD_1_Position(1,20);

                                LCD_1_PrString(changingVariable);
                                }
                        Sound(soundPulse);
                        goto State1;
                        }
                if (button1 == 0x01)                            // Increment upperAirwayFractionStart
                        {
                        upperAirwayFractionStart = upperAirwayFractionStart +1;
                        Sound(soundPulse);
                        if (upperAirwayFractionStart == 101)
                                upperAirwayFractionStart = 0;
                        }
                if (button2 == 0x10)                            // Decrement upperAirwayFractionStart
                        {
                        upperAirwayFractionStart = upperAirwayFractionStart -1;
                        Sound(soundPulse);
                        if (upperAirwayFractionStart == - 1)
                                upperAirwayFractionStart = 100;
                        }

                        lcd_e = 0x10;
                        LCD_1_Position(1,13);
                        LCD_1_PrCString("   ");

                        if (upperAirwayFractionStart < 10)
                                LCD_1_Position(1,15);

                        if (upperAirwayFractionStart < 100   && upperAirwayFractionStart >= 10)
                                LCD_1_Position(1,14);

                        if (upperAirwayFractionStart == 100)
                                LCD_1_Position(1,13);

                        itoa(changingVariable,upperAirwayFractionStart,10);
                        LCD_1_PrString(changingVariable);

        goto State8;

        State9:
                DeleteScreenArrowsDataEntryStates();
                lcd_e = 0x10;
                LCD_1_Position(1,19);
                LCD_1_WriteData(0xA5);
                Wait();
                ButtonPressCheck();

                if (button3 == 0x04)                                    // Enter button.
```

```
                    {
                    Sound(soundPulse);
                    goto State2;
                    }

            if (button1 == 0x01)                    // Increment upperAirwayFractionEnd
                    {
                    upperAirwayFractionEnd = upperAirwayFractionEnd +1;
                    if (upperAirwayFractionEnd == 101)
                            upperAirwayFractionEnd = upperAirwayFractionStart;
                            Sound(soundPulse);
                    }

            if (button2 == 0x10)                    // Decrement upperAirwayFractionEnd
                    {
                    upperAirwayFractionEnd = upperAirwayFractionEnd -1;
                    if (upperAirwayFractionEnd == (upperAirwayFractionStart - 1))
                            upperAirwayFractionEnd = 100;
                    Sound(soundPulse);
                    }

                    lcd_e = 0x10;
                    LCD_1_Position(1,20);
                    LCD_1_PrCString("   ");
                    itoa(changingVariable,upperAirwayFractionEnd,10);

                    if (upperAirwayFractionEnd < 10)
                            LCD_1_Position(1,22);

                    if (upperAirwayFractionEnd < 100  && upperAirwayFractionEnd >= 10)
                            LCD_1_Position(1,21);

                    if (upperAirwayFractionEnd == 100)
                            LCD_1_Position(1,20);

                    LCD_1_PrString(changingVariable);
    goto State9;

    State10:
            DeleteScreenArrowsDataEntryStates();
            lcd_e = 0x80;
            LCD_1_Position(0,12);
            LCD_1_WriteData(0xA5);
            Wait();
            ButtonPressCheck();

            if (button3 == 0x04)                            // Enter button.
                    {
                    if (lowerAirwayFractionEnd < lowerAirwayFractionStart)
                            {
                            lowerAirwayFractionEnd = lowerAirwayFractionStart;
                            lcd_e = 0x80;
                            LCD_1_Position(0,20);
                            LCD_1_PrCString("   ");
                            itoa(changingVariable,lowerAirwayFractionEnd,10);

                            if (lowerAirwayFractionEnd < 10)
                                    LCD_1_Position(0,22);

                            if (lowerAirwayFractionEnd < 100  && lowerAirwayFractionEnd >=
10)
                                    LCD_1_Position(0,21);

                            if (lowerAirwayFractionEnd == 100)
                                    LCD_1_Position(0,20);

                            LCD_1_PrString(changingVariable);
```

```
                        }
                Sound(soundPulse);
                goto State3;
                }

        if (button1 == 0x01)                    // Increment upperAirwayFractionStart
                {
                lowerAirwayFractionStart = lowerAirwayFractionStart +1;
                if (lowerAirwayFractionStart == 101)
                        lowerAirwayFractionStart = 0;
                Sound(soundPulse);
                }

        if (button2 == 0x10)                    // Decrement upperAirwayFractionStart
                {
                lowerAirwayFractionStart = lowerAirwayFractionStart -1;
                if (lowerAirwayFractionStart == - 1)
                        lowerAirwayFractionStart= 100;
                Sound(soundPulse);
                }

        lcd_e = 0x80;
        LCD_1_Position(0,13);
        LCD_1_PrCString("   ");

        if (lowerAirwayFractionStart < 10)
                LCD_1_Position(0,15);

        if (lowerAirwayFractionStart < 100   && lowerAirwayFractionStart >= 10)
                LCD_1_Position(0,14);

        if (lowerAirwayFractionStart == 100)
                LCD_1_Position(0,13);

        itoa(changingVariable,lowerAirwayFractionStart,10);
        LCD_1_PrString(changingVariable);
goto State10;

State11:
        DeleteScreenArrowsDataEntryStates();
        lcd_e = 0x80;
        LCD_1_Position(0,19);
        LCD_1_WriteData(0xA5);
        Wait();
        ButtonPressCheck();

        if (button3 == 0x04)                            // Enter button.
                {
                Sound(soundPulse);
                goto State4;
                }

        if (button1 == 0x01)                    // Increment lowerAirwayFractionEnd
                {
                lowerAirwayFractionEnd = lowerAirwayFractionEnd +1;
                if (lowerAirwayFractionEnd == 101)
                        lowerAirwayFractionEnd = lowerAirwayFractionStart;
                Sound(soundPulse);
                }

        if (button2 == 0x10)                    // Decrement lowerAirwayFractionEnd
                {
                lowerAirwayFractionEnd = lowerAirwayFractionEnd -1;
                if (lowerAirwayFractionEnd == (lowerAirwayFractionStart - 1))
                        lowerAirwayFractionEnd = 100;
                Sound(soundPulse);
                }
```

```
		lcd_e = 0x80;
		LCD_1_Position(0,20);
		LCD_1_PrCString("   ");
		itoa(changingVariable,lowerAirwayFractionEnd,10);

		if (lowerAirwayFractionEnd < 10)
			LCD_1_Position(0,22);

		if (lowerAirwayFractionEnd < 100  && lowerAirwayFractionEnd >= 10)
			LCD_1_Position(0,21);

		if (lowerAirwayFractionEnd == 100)
			LCD_1_Position(0,20);

		LCD_1_PrString(changingVariable);
	goto State11;

	State12:
		DeleteScreenArrowsDataEntryStates();
		lcd_e = 0x10;
		LCD_1_Position(1,30);
		LCD_1_WriteData(0xA5);
		Wait();
		ButtonPressCheck();

		if (button3 == 0x04)                          // Enter button.
			{
			Sound(soundPulse);
			goto State5;
			}

		if (button1 == 0x01)                          // Increment tolerance
			{
			Sound(soundPulse);
			tolerance = tolerance +1;
			if (tolerance == 101)
				tolerance = 0;
			}

		if (button2 == 0x10)                          // Decrement tolerance
			{
			Sound(soundPulse);
			tolerance = tolerance -1;
			if (tolerance == - 1)
				tolerance = 100;
			}

		lcd_e = 0x10;
		LCD_1_Position(1,31);
		LCD_1_PrCString("   ");
		itoa(changingVariable,tolerance,10);

		if (tolerance < 10)
			LCD_1_Position(1,33);

		if (tolerance < 100  && tolerance >= 10)
			LCD_1_Position(1,32);

		if (tolerance == 100)
			LCD_1_Position(1,31);

		LCD_1_PrString(changingVariable);

	goto State12;
}
```

```
void ObtainAverageBreath(void)
{
//*********************************************************************************************
// Setup Screen
lcd_e = 0x90;
LCD_1_Control(0x01);

lcd_e = 0x10;
LCD_1_Position(0,2);
LCD_1_PrCString("MODE: OBTAIN AVERAGE BREATH PROFILE");

LCD_1_Position(1,10);
LCD_1_PrCString("Breath Into Device When");

lcd_e = 0x80;
LCD_1_Position(0,15);
LCD_1_PrCString("Ready to Begin");

LCD_1_Position(1,0);
LCD_1_PrCString("<<");

LCD_1_Position(1,38);
LCD_1_PrCString(">>");

// ***************************************************************************
Timer16_1_WritePeriod(390);

RecordZeroVoltage:
        for (j=1;j<=zeroVoltCount;j++)
                        {
                        ReadHoneywell();
                        zeroVoltage = zeroVoltage + voltage ;
                        }
        zeroVoltage = zeroVoltage / zeroVoltCount;
        voltage = 0;
//***************************************************************************
        State1:
                DeleteScreenArrowsCollectBreathStates();
                lcd_e = 0x80;
                LCD_1_Position(1,37);
                LCD_1_WriteData(0x7E);
                if (!(upperToleranceLimit > tolBreath1 && upperToleranceLimit > tolBreath2 &&
upperToleranceLimit > tolBreath3 && lowerToleranceLimit < tolBreath1 && lowerToleranceLimit <
tolBreath2 && lowerToleranceLimit < tolBreath3))
                        {
                        ReadHoneywell();
                        if ((voltage - zeroVoltage) > 0.01)
                                {
                                lcd_e = 0x90;
                                LCD_1_Control(0x01);
                                lcd_e = 0x10;
                                LCD_1_Position(0,2);
                                LCD_1_PrCString("MODE:  OBTAIN  AVERAGE  BREATH
PROFILE");
                                lcd_e = 0x80;
                                LCD_1_Position(1,0);
                                LCD_1_PrCString("<<");
                                LCD_1_Position(1,38);
                                LCD_1_PrCString(">>");
                                oneSec = 0;
                                Timer16_1_Start();
                                Timer16_1_EnableInt();
                                integratedArea = 0;
                                goto State4;
                                }
                        }
                Wait();
```

```
                ButtonPressCheck();

                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button2 == 0x10)            // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button3 == 0x04)                    // Select variable to change.
                        {
                        Sound(soundPulse);
                        goto State3;
                        }

        goto State1;

        State2:
                DeleteScreenArrowsCollectBreathStates();
                lcd_e = 0x80;
                LCD_1_Position(1,2);
                LCD_1_WriteData(0x7F);
                if (!(upperToleranceLimit > tolBreath1 && upperToleranceLimit > tolBreath2 &&
upperToleranceLimit > tolBreath3 && lowerToleranceLimit < tolBreath1 && lowerToleranceLimit <
tolBreath2 && lowerToleranceLimit < tolBreath3))
                        {
                        ReadHoneywell();
                        if ((voltage - zeroVoltage) > 0.01)
                                {
                                //lcd_e = 0x90;
                                //LCD_1_Control(0x01);
                                lcd_e = 0x10;
                                LCD_1_Position(0,2);
                                LCD_1_PrCString("MODE:    OBTAIN    AVERAGE    BREATH
PROFILE");
                                lcd_e = 0x80;
                                LCD_1_Position(1,0);
                                LCD_1_PrCString("<<");
                                LCD_1_Position(1,38);
                                LCD_1_PrCString(">>");
                                oneSec = 0;
                                Timer16_1_Start();
                                Timer16_1_EnableInt();
                                integratedArea = 0;
                                goto State4;
                                }
                        }
                Wait();
                ButtonPressCheck();

                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State1;
                        }

                if (button2 == 0x10)            // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State1;
                        }
```

```
                if (button3 == 0x04)                    // Select variable to change.
                        {
                        backFlag = 1;
                        tolBreath1 = 0;
                        tolBreath2 = 0;
                        tolBreath3 = 0;
                        sampleBreath1 = 0;
                        sampleBreath2 = 0;
                        sampleBreath3 = 0;
                        Sound(soundPulse);
                        return;
                        }
        goto State2;

        State3:
                if (!(upperToleranceLimit > tolBreath1 && upperToleranceLimit > tolBreath2 &&
upperToleranceLimit > tolBreath3 && lowerToleranceLimit < tolBreath1 && lowerToleranceLimit <
tolBreath2 && lowerToleranceLimit < tolBreath3))
                        {
                        lcd_e = 0x80;
                        LCD_1_Position(1,2);
                        LCD_1_PrCString("No Avg Breaths or Out of Tolerance");
                        for (j=0;j<=40000;j++);
                        LCD_1_Position(1,2);
                        LCD_1_PrCString("                              ");
                //lcd_e = 0x90;
                //LCD_1_Control(0x01);
                        goto State1;
                        }
                ReadHoneywell();
                while ((voltage - zeroVoltage) >= 0.01)
                        ReadHoneywell();
                tolBreath1 = 0;
                tolBreath2 = 0;
                tolBreath3 = 0;
                sampleBreath1 = 0;
                sampleBreath2 = 0;
                sampleBreath3 = 0;
                Sound(soundPulse);
                return;

        State4:
                DeleteScreenArrowsCollectBreathStates();

                while ((voltage - zeroVoltage) >= 0.01)
                        {
                        if (oneSec ==1)
                                {
                                oneSec = 0;
                                ReadHoneywell();
                                }
                        }//Breath given now
                Timer16_1_DisableInt();
                Timer16_1_Stop();

                if (breathsInTolerance % 3 == 1)
                        sampleBreath1 = integratedArea;
                if (breathsInTolerance % 3 == 2)
                        sampleBreath2 = integratedArea;
                if (breathsInTolerance % 3 == 0)
                        sampleBreath3 = integratedArea;

                tolBreath1 = sampleBreath1/sampleBreath2;
                tolBreath2 = sampleBreath1/sampleBreath3;
                tolBreath3 = sampleBreath3/sampleBreath2;
                upperToleranceLimit = 1+((float)tolerance/100);
                lowerToleranceLimit = 1-((float)tolerance/100);
```

```
                runningTotalArea = runningTotalArea + integratedArea;
                integratedArea = 0;
                breathsInTolerance = breathsInTolerance + 1;
        //lcd_e = 0x80;
        //itoa(changingVariable,totalExhaledBreaths,10);
        //LCD_1_Position(1,20);
        //LCD_1_PrString(changingVariable);

                if (upperToleranceLimit > tolBreath1 && upperToleranceLimit > tolBreath2 &&
upperToleranceLimit > tolBreath3 && lowerToleranceLimit < tolBreath1 && lowerToleranceLimit <
tolBreath2 && lowerToleranceLimit < tolBreath3)
                        {
                        lcd_e = 0x90;
                        LCD_1_Control(0x01);
                        lcd_e = 0x10;
                        LCD_1_Position(0,2);
                        LCD_1_PrCString("MODE: OBTAIN AVERAGE BREATH PROFILE");
                        LCD_1_Position(1,2);
                        LCD_1_PrCString("Avg of 3 Breaths in    % Tolerance");
                        LCD_1_Position(1,22);
                        itoa(changingVariable,tolerance,10);
                        LCD_1_PrString(changingVariable);
                        lcd_e = 0x80;
                        LCD_1_Position(0,7);
                        LCD_1_PrCString("Select >> To Continue");
                        LCD_1_Position(1,0);
                        LCD_1_PrCString("<<");
                        LCD_1_Position(1,38);
                        LCD_1_PrCString(">>");
                        AverageBreath = (sampleBreath1 + sampleBreath2 + sampleBreath3)/3;
                        }
        goto State1;

} // End of Function


void BreathCollection(void)
{
//*************************************************************************************************
// Setup Screen
lcd_e = 0x90;
LCD_1_Control(0x01);
//for (i=0;i<=30000;i++);

lcd_e = 0x10;
LCD_1_Position(0,2);
LCD_1_PrCString("MODE: BREATH FRACTINATION");

LCD_1_Position(1,2);
LCD_1_PrCString("Breath Into Device To Collect Sample");

lcd_e = 0x80;
LCD_1_Position(0,8);
LCD_1_PrCString("Select >> When Finished");

LCD_1_Position(1,0);
LCD_1_PrCString("<<");

LCD_1_Position(1,38);
LCD_1_PrCString(">>");

// ************************************************************************
        State1:
                DeleteScreenArrowsCollectBreathStates();
                lcd_e = 0x80;
```

```
                LCD_1_Position(1,37);
                LCD_1_WriteData(0x7E);
                ReadHoneywell();
                if ((voltage - zeroVoltage) > 0.01)
                        {
                        oneSec = 0;
                        Timer16_1_Start();
                        Timer16_1_EnableInt();
                        integratedArea = 0;
                        goto State3;
                        }
                Wait();
                ButtonPressCheck();
                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button2 == 0x10)            // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button3 == 0x04)                    // Select variable to change.
                        {
                        Sound(soundPulse);
                        return;
                        }
        goto State1;

State2:
                DeleteScreenArrowsCollectBreathStates();
                lcd_e = 0x80;
                LCD_1_Position(1,2);
                LCD_1_WriteData(0x7F);
                ReadHoneywell();

                if ((voltage - zeroVoltage) > 0.01)
                        {
                        oneSec = 0;
                        Timer16_1_Start();
                        Timer16_1_EnableInt();
                        integratedArea = 0;
                        goto State3;
                        }
                Wait();
                ButtonPressCheck();

                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State1;
                        }

                if (button2 == 0x10)            // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State1;
                        }

                if (button3 == 0x04)                    // Select variable to change.
                        {
                        Sound(soundPulse);
                        backFlag = 1;
                        return;
```

```
                    }
goto State2;



State3:
        DeleteScreenArrowsCollectBreathStates();
        while ((voltage - zeroVoltage) >= 0.01)
                {
                if (oneSec ==1)
                        {
                        oneSec = 0;
                        ReadHoneywell();
                        }
                if ((integratedArea/AverageBreath) > ((float)upperAirwayFractionStart/100) &&
(integratedArea/AverageBreath) < ((float)upperAirwayFractionEnd/100))
                        {
                        PRT2DR = PRT2DR | VALVE1;
                        }
                else
                        {
                        PRT2DR = PRT2DR & ~VALVE1;
                        }
                if ((integratedArea/AverageBreath) > ((float)lowerAirwayFractionStart/100) &&
(integratedArea/AverageBreath) < ((float)lowerAirwayFractionEnd/100))
                        {
                        PRT2DR = PRT2DR | VALVE2;
                        }
                else
                        {
                        PRT2DR = PRT2DR & ~VALVE2;
                        }

                if ((integratedArea/AverageBreath) > 0.2)
                        PRT2DR = PRT2DR | LED1;
                if ((integratedArea/AverageBreath) > 0.5)
                        PRT2DR = PRT2DR | LED2;
                if ((integratedArea/AverageBreath) >= 0.8)
                        PRT2DR = PRT2DR | LED3;

                }//Breath given now

        Timer16_1_DisableInt();
        Timer16_1_Stop();
        PRT2DR = PRT2DR & ~VALVE1;
        PRT2DR = PRT2DR & ~VALVE2;
        for (i=0;i>=20000;i++);
        PRT2DR = PRT2DR & ~LED1;
        PRT2DR = PRT2DR & ~LED2;
        PRT2DR = PRT2DR & ~LED3;
        totalExhaledBreaths = totalExhaledBreaths +1;

        if ((integratedArea/AverageBreath) < upperToleranceLimit && (integratedArea/AverageBreath)
> lowerToleranceLimit);
                //breathsInTolerance = breathsInTolerance + 1;
        else
                breathsOutTolerance = breathsOutTolerance + 1;

        integratedArea = 0;
goto State1;
}// END OF FUNCTION

void Summary(void)
{
//*********************************************************************************************
// Setup Screen
lcd_e = 0x90;
```

```
LCD_1_Control(0x01);

lcd_e = 0x10;
LCD_1_Position(0,2);
LCD_1_PrCString("MODE: COLLECTION SUMMARY");

LCD_1_Position(1,2);
LCD_1_PrCString("Breaths Collected ");
LCD_1_Position(1,21);
itoa(changingVariable,totalExhaledBreaths,10);
LCD_1_PrString(changingVariable);

LCD_1_Position(1,28);
LCD_1_PrCString("Tol    %");
LCD_1_Position(1,33);
itoa(changingVariable,tolerance,10);
LCD_1_PrString(changingVariable);

lcd_e = 0x80;
LCD_1_Position(0,2);
LCD_1_PrCString("Breaths out of Tol ");
LCD_1_Position(0,22);
itoa(changingVariable,breathsOutTolerance,10);
LCD_1_PrString(changingVariable);

LCD_1_Position(1,16);
LCD_1_PrCString("Return to Purge Mode");


LCD_1_Position(1,0);
LCD_1_PrCString("<<");

//LCD_1_Position(1,38);
//LCD_1_PrCString(">>");
//*************************************************************************************************
        State1:
                DeleteScreenArrowsCollectBreathStates();
                lcd_e = 0x80;
                LCD_1_Position(1,37);
                LCD_1_WriteData(0x7E);
                Wait();
                ButtonPressCheck();

                if (button1 == 0x01)                    // Go forward one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button2 == 0x10)            // Go back one state.
                        {
                        Sound(soundPulse);
                        goto State2;
                        }

                if (button3 == 0x04)                    // Select variable to change.
                        {
                        Sound(soundPulse);
                        return;
                        }

        goto State1;

        State2:
                DeleteScreenArrowsCollectBreathStates();
                lcd_e = 0x80;
                LCD_1_Position(1,2);
```

```
                    LCD_1_WriteData(0x7F);
                    Wait();
                    ButtonPressCheck();

                    if (button1 == 0x01)                    // Go forward one state.
                            {
                            Sound(soundPulse);
                            goto State1;
                            }

                    if (button2 == 0x10)            // Go back one state.
                            {
                            Sound(soundPulse);
                            goto State1;
                            }

                    if (button3 == 0x04)                    // Select variable to change.
                            {
                            Sound(soundPulse);
                            backFlag = 1;
                            return;
                            }

            goto State2;

}

void ReadHoneywell(void)
{
            previousVoltage = voltage;
            for(i=1;i<=3;i++)
            //Read the data three times and take the last read as the value wanted
                    {
                    while (!DELSIG11_1_fIsDataAvailable());                 // is data available
                    volt=DELSIG11_1_iGetDataClearFlag();                            //data is available so
get it and store it
                    }
            masked = volt & VOLTMASK;
            flipped = masked ^ VOLTFLIP;
            voltage = (float)5*((float)flipped/(float)0x0800);
            integratedArea = integratedArea + time * (0.5 * voltage + 0.5 * previousVoltage);
            return;
}

void IndicateBusy(void)
{
if (busyTime % 4 == 1)
                    {
                    lcd_e = 0x10;
                    LCD_1_Position(0,39);
                    LCD_1_WriteData(0xFF);
                    }
            if (busyTime % 4 == 2)
                    {
                    lcd_e = 0x10;
                    LCD_1_Position(0,39);
                    LCD_1_WriteData(0xA0);
                    //LCD_1_PrCString("G");
                    }
            if (busyTime % 4 == 3)
                    {
                    lcd_e = 0x10;
                    LCD_1_Position(0,39);
                    LCD_1_WriteData(0xFF);
                    }
            if (busyTime % 4 == 0)
                    {
```

```
                        lcd_e = 0x10;
                        LCD_1_Position(0,39);
                        LCD_1_WriteData(0xA0);
                        }
            return;
} // End of Function

//=========================================================================
====
void Sound(int pulseTime)
{
PWM16_1_Start();
PWM16_1_WritePulseWidth(pulseTime);
for (i=0;i<1200;i++);
PWM16_1_Stop();
return;
}
```

B3 Modified LCD Code

```
;;*****************************************************************************
;;  FILENAME:   LCD_1.asm
;;  Version: 1.4, Updated on 2005/09/30 at 10:52:22
;;  Generated by PSoC Designer ver 4.2  b1013 : 02 September, 2004
;;
;;  DESCRIPTION: LCD User Module software implementation file
;;               for 22/24/25/26/27xxx PSoC family of devices.
;;
;; This set of functions is written for the common 2 and 4 line
;; LCDs that use the Hitachi HD44780A controller.
;;
;;  LCD connections to PSoC port
;;
;;    PX.0 ==> LCD D4
;;    PX.1 ==> LCD D5
;;    PX.2 ==> LCD D6
;;    PX.3 ==> LCD D7
;;    PX.4 ==> LCD E
;;    PX.5 ==> LCD RS
;;    PX.6 ==> LCD R/W
;;
;;  NOTE: User Module APIs conform to the fastcall16 convention for marshalling
;;        arguments and observe the associated "Registers are volatile" policy.
;;        This means it is the caller's responsibility to preserve any values
;;        in the X and A registers that are still needed after the API functions
;;        returns. For Large Memory Model devices it is also the caller's
;;        responsibility to perserve any value in the CUR_PP, IDX_PP, MVR_PP and
;;        MVW_PP registers. Even though some of these registers may not be modified
;;        now, there is no guarantee that will remain the case in future releases.
;;-----------------------------------------------------------------------------
;;  Copyright (c) Cypress MicroSystems 2001-2003. All Rights Reserved.
;;*****************************************************************************
;;
;;*****************************************************************************
;;

include "m8c.inc"
include "memory.inc"
include "LCD_1.inc"

;-------------------------------------------------
;  Global Symbols
;-------------------------------------------------

export   LCD_1_Start
export  _LCD_1_Start
export   LCD_1_Init
export  _LCD_1_Init

export   LCD_1_WriteData
export  _LCD_1_WriteData

export   LCD_1_Control
export  _LCD_1_Control

export  LCD_1_PrString
export _LCD_1_PrString

export  LCD_1_PrCString
export _LCD_1_PrCString

export  LCD_1_Position
export _LCD_1_Position

export  LCD_1_PrHexByte
```

```
export _LCD_1_PrHexByte

export  LCD_1_PrHexInt
export _LCD_1_PrHexInt

export  LCD_1_Delay50uTimes
export _LCD_1_Delay50uTimes

export  LCD_1_Delay50u
export _LCD_1_Delay50u

;-----------------------------------------------
; If bargraph functions not required, don't
; export the function names.
;-----------------------------------------------

IF (LCD_1_BARGRAPH_ENABLE)
export  LCD_1_InitBG
export _LCD_1_InitBG

export  LCD_1_InitVBG
export _LCD_1_InitVBG

; NOTE: The two functions,
;
;    LCD_1_DrawVBG and
;    LCD_1_DrawBG
;
; are implemented using both fastcall16 and legacy fastcall16 because they
; fall into a special and rare case where the calling sequences specified
; by the two disciplines are incompatible. The fastcall16 versions are
; provided for both C and Assembly users in all memory models. The legacy
; fastcall16 versions are provided only to support existing small memory
; model assembly language code---they do not work in the large memory
; model.
;
; ** The legacy fastcall16 versions are provided on a temporary basis to
; ** ease the transition to the 4.2 release of PSoC Designer. Their use is
; ** deprecated and thier status is "No Further Maintenance".
;
; The fastcall16 versions of these functions are distinguished by a
; leading underscore in the name. The legacy fastcall16 names (which appear
; in this comment) do not have the leading underscore. Details on the
; calling sequence to be used for fastcall16 are given in the user module
; datasheet.
;
; Fastcall16 versions:
export _LCD_1_DrawVBG
export _LCD_1_DrawBG

IF SYSTEM_SMALL_MEMORY_MODEL
; Legacy Fastcall versions:
export  LCD_1_DrawVBG
export  LCD_1_DrawBG
ENDIF  ; SYSTEM_SMALL_MEMORY_MODEL

ENDIF  ; BARGRAPH_ENABLE

;
; The following functions are deprecated and will be eliminated in a future
; version of PSoC Designer.
;
export   LCD_1_Write_Data
export  _LCD_1_Write_Data


;-----------------------------------------------
```

```
;  EQUATES
;---------------------------------------------
;LCD_E:             equ    10h
;Replaced by C variable _lcd_e declared in main.h
LCD_RW:             equ    40h
LCD_RS:             equ    20h

LCD_DATA_MASK:      equ    0Fh
LCD_READY_BIT:      equ    08h

;LCD_DATA_READ:      equ    ( LCD_E | LCD_RW | LCD_RS )
;LCD_CNTL_READ:      equ    ( LCD_E | LCD_RW )
LCD_DATA_READ:                  equ        (LCD_RW | LCD_RS)
LCD_CNTL_READ:      equ             LCD_RW
;LCD_PORT_WRITE:     equ    7Fh
;LCD_PORT_MASK:      equ    7Fh
LCD_PORT_WRITE:     equ    FFh
LCD_PORT_MASK:      equ    FFh

LCD_Port:           equ    PRT1DR
LCD_PortMode0:      equ    PRT1DM0
LCD_PortMode1:      equ    PRT1DM1

DISP_INC:           equ    03h
DISP_OFF:           equ    08h
DISP_ON:            equ    0Ch
LCD_4BIT_2LINE:     equ    2Ch


;---------------------------------------------
;      Bargraph definitions
;---------------------------------------------

LCD_BG_CHAR_WIDTH:  equ    16     ; 16 characters in width
LCD_BG_SEG_WIDTH:   equ    80     ; 16 * 5 = 80
LCD_BG_COL_START:   equ    0      ; Always start in the left most column

                            ; Offsets for 2x16, 2x20, 4x20
                            ; Change these values for a custome LCD

LCD_ROW1_OFFSET:    equ    80h    ; Address/command offset for row 1
LCD_ROW2_OFFSET:    equ    C0h    ; Address/command offset for row 2
LCD_ROW3_OFFSET:    equ    94h    ; Address/command offset for row 1
LCD_ROW4_OFFSET:    equ    D4h    ; Address/command offset for row 2

LCD_BG_ROW1_OFFSET: equ    80h    ; Address/command offset for row 1
LCD_BG_ROW2_OFFSET: equ    C0h    ; Address/command offset for row 2

CG_RAM_OFFSET:      equ    40h    ; Offset to character RAM

AREA UserModules (ROM, REL)

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_PrCString
;
;  DESCRIPTION:
;    Print constant (ROM) string to LCD
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;     A:X  Pointer to String
;          A contains MSB of string address
;          X contains LSB of string address
;
;  RETURNS:  none
;
```

```
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
 LCD_1_PrCString:
_LCD_1_PrCString:
    RAM_PROLOGUE RAM_USE_CLASS_1
 Loop_PrCString:
    push  A                         ; Store ROM pointer
    push  X
    romx                            ; Get character from ROM
    jnz   LCD_PrCString_WR              ; print character and advance pointer
    pop   X                         ; Restore the stack
    pop   A
    RAM_EPILOGUE RAM_USE_CLASS_1
    ret                             ; Return

LCD_PrCString_WR:
    call  LCD_1_WriteData               ; Write data to LCD
    pop   X                         ; Get ROM pointer
    pop   A
    inc   X                         ; Inc LSB of pointer
    jnc   Loop_PrCString
    inc   A                         ; Inc MSB of pointer if LSB overflow
    jmp   Loop_PrCString

.ENDSECTION

;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_PrHexByte
;
;  DESCRIPTION:
;     Print a byte in Hex (two characters) to current LCD position
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;     A  => (BYTE) Data/char to be printed
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
.LITERAL
LCD_HEX_STR::
     DS    "0123456789ABCDEF"
.ENDLITERAL
.SECTION

 LCD_1_PrHexByte:
_LCD_1_PrHexByte:
    RAM_PROLOGUE RAM_USE_CLASS_1
    push  A                         ; Save lower nibble
```

```
    asr   A                         ; Shift high nibble to right
    asr   A
    asr   A
    asr   A
    and   A,0Fh                     ; Mask off nibble
    index LCD_HEX_STR               ; Get Hex value
    call  LCD_1_WriteData           ; Write data to screen
    pop   A                         ; Restore value
    and   A,0Fh                     ; Mask off lower nibble
    index LCD_HEX_STR               ; Get Hex value
    call  LCD_1_WriteData           ; Write data to screen
    RAM_EPILOGUE RAM_USE_CLASS_1
    ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_PrHexInt
;
;  DESCRIPTION:
;     Print an Int in Hex (four characters) to current LCD position
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;     A:X Integer value
;         A  contains LSB of Int
;         X  contains MSB of Int
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
 LCD_1_PrHexInt:
_LCD_1_PrHexInt:
    RAM_PROLOGUE RAM_USE_CLASS_1
    swap  A,X
    call  LCD_1_PrHexByte           ; Print MSB
    mov   A,X                       ; Move LSB into position
    call  LCD_1_PrHexByte           ; Print LSB
    RAM_EPILOGUE RAM_USE_CLASS_1
    ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_PrString
;
;  DESCRIPTION:
;     Print (RAM) ASCII string to LCD
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;     A:X contains pointer to string
;         X  contains LSB of string pointer
;         A  contains MSB or page of string pointer (not used at this time)
;
;  RETURNS:
```

```
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;         IDX_PP
;
;
 LCD_1_PrString:
_LCD_1_PrString:
    RAM_PROLOGUE RAM_USE_CLASS_3
    RAM_SETPAGE_IDX A
 Loop_PrString:
    mov   A,[X]                     ; Get value pointed to by X
    jz    End_LCD_PrString          ; Check for end of string
    ;LCD_1_writeData is known not to modify X so no need to perserve
    call  LCD_1_WriteData           ; Write data to screen
    inc   X                         ; Advance pointer to next character
    push  X
    pop   X
    jmp   Loop_PrString             ; Go get next character
End_LCD_PrString:
    RAM_EPILOGUE RAM_USE_CLASS_3
    ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_WriteData
;
;  DESCRIPTION:
;    Write a byte to the LCD's data register.
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;    A contains byte to be written to LCD data register
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
 LCD_1_WriteData:
_LCD_1_WriteData:
 LCD_1_Write_Data:   ; Do not use
_LCD_1_Write_Data:   ; Do not use
    RAM_PROLOGUE RAM_USE_CLASS_1
    call  LCD_Check_Ready           ; Make sure controller is ready
                                    ; A is preserved in LCD_Check_Ready
    push  A                         ; Save copy of character
    asr   A                         ; Shift high nibble to right
    asr   A
    asr   A
    asr   A
```

```
   and   A,0Fh                    ; Mask off high nibble
   call  LCD_WDATA_Nibble         ; Write Upper nibble
   pop   A                        ; Retrieve copy of character
   and   A,0Fh                    ; Mask off high nibble
   nop
   nop
   nop
   call  LCD_WDATA_Nibble         ; Write Lower nibble
   RAM_EPILOGUE RAM_USE_CLASS_1
   ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_Control
;
;  DESCRIPTION:
;    Write a byte to the LCD's control register.
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;    A contains data to be written to LCD control register.
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;   The A and X registers may be modified by this or future implementations
;   of this function. The same is true for all RAM page pointer registers in
;   the Large Memory Model. When necessary, it is the calling function's
;   responsibility to perserve their values across calls to fastcall16
;   functions.
;
;   Currently only the page pointer registers listed below are modified:
;        CUR_PP
;
 LCD_1_Control:
_LCD_1_Control:
   RAM_PROLOGUE RAM_USE_CLASS_1
   call  LCD_Check_Ready          ; Make sure controller is ready
                                  ; A is preserved in LCD_Check_Ready
   push  A                        ; Save copy of byte
   asr   A                        ; Shift Upper Nibble to right
   asr   A
   asr   A
   asr   A
   and   A,0Fh                    ; Mask off, just in case
   call  LCD_WCNTL_Nibble         ; Write high nibble
   pop   A                        ; Restore copy of byte
   and   A,0Fh                    ; Mask off high nibble
   nop
   nop
   nop
   call  LCD_WCNTL_Nibble         ; Write Lower nibble
   RAM_EPILOGUE RAM_USE_CLASS_1
   ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_WCNTL_Nibble
;
;  DESCRIPTION:
;    Write a single nibble to the LCD's command register
;
;-----------------------------------------------------------------------------
;
```

```
;  ARGUMENTS:
;     A[3:0]   Contains Nibble to be written to command register
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
 LCD_WCNTL_Nibble:
    RAM_PROLOGUE RAM_USE_CLASS_4
    push  A
    RAM_SETPAGE_CUR >Port_1_Data_SHADE          ; Set CUR_PP to LCD variable address
    and   [Port_1_Data_SHADE],~LCD_PORT_MASK
    mov   A,[Port_1_Data_SHADE]
    mov   reg[LCD_Port],A                       ; Reset control lines

    pop   A
    and   A,LCD_DATA_MASK                       ; Make sure no bogus data in MSN
    ;or    A,LCD_E                            ; Bring "E" Enable line high
    or        A,[_lcd_e]
    or    A,[Port_1_Data_SHADE]                 ; OR in bit 7 just
    mov   reg[LCD_Port], A                      ; Write data
    mov   [Port_1_Data_SHADE],A                 ; Keep shadow register in sync
    nop
    and   A,(~LCD_PORT_MASK|LCD_DATA_MASK)      ; Disable E signal and leave data on bus.
    mov   [Port_1_Data_SHADE],A                 ; Keep shadow register in sync
    mov   reg[LCD_Port],A
    RAM_EPILOGUE RAM_USE_CLASS_4
    ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_WDATA_Nibble
;
;  DESCRIPTION:
;    Write a single nibble to the LCD's DATA register
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;     A[3:0]   Contains Nibble to be written to data register
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
LCD_WDATA_Nibble:
    RAM_PROLOGUE RAM_USE_CLASS_4
    push  A
    RAM_SETPAGE_CUR >Port_1_Data_SHADE          ; Set CUR_PP to LCD variable address
    and   [Port_1_Data_SHADE],~LCD_PORT_MASK
```

```
   or    [Port_1_Data_SHADE],LCD_RS              ; Raise RS to signify a Data Write
   mov   A,[Port_1_Data_SHADE]
   mov   reg[LCD_Port],A

   pop   A
   and   A,LCD_DATA_MASK                         ; Make sure no bogus data in A[7:4]
   ;or    A,(LCD_E | LCD_RS)                     ; Bring "E" Enable line high
   or        A,[_lcd_e]
   or        A,LCD_RS
   or    A,[Port_1_Data_SHADE]                   ; Keep shadow in sync
   mov   reg[LCD_Port], A                        ; Write data
   mov   [Port_1_Data_SHADE],A                   ; Keep shadow in sync
   NOP
   and   A,(~LCD_PORT_MASK|LCD_DATA_MASK|LCD_RS)     ; Disable E signal and leave Data on
bus
   mov   [Port_1_Data_SHADE],A                   ; keep shadow in sync
   mov   reg[LCD_Port],A
   RAM_EPILOGUE RAM_USE_CLASS_4
   ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_Check_Ready
;
;  DESCRIPTION:
;    Wait until LCD has completed last command.
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS: none
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;   The A and X registers may be modified by this or future implementations
;   of this function. The same is true for all RAM page pointer registers in
;   the Large Memory Model. When necessary, it is the calling function's
;   responsibility to perserve their values across calls to fastcall16
;   functions.
;
;   Currently only the page pointer registers listed below are modified:
;       CUR_PP
;
;    If LCD is not present, this routine may never return.
;
LCD_Check_Ready:
   RAM_PROLOGUE RAM_USE_CLASS_4
   push  A                                       ; Save Accumulator
   RAM_SETPAGE_CUR >Port_1_Data_SHADE            ; Set CUR_PP to LCD variable address
   and   [Port_1_Data_SHADE],~LCD_PORT_MASK      ; Mask of all LCD bits
   mov   A,[Port_1_Data_SHADE]
   mov   reg[LCD_Port],A                         ; Zero LCD port bits

   and   [Port_1_DriveMode_0_SHADE],~LCD_DATA_MASK     ; Clear out LCD mode bits.
   mov   A,[Port_1_DriveMode_0_SHADE]
   M8C_SetBank1                                  ; Change port mode to read status
   mov   reg[LCD_PortMode0],A                    ; Setup LCD Port for reading
   M8C_SetBank0

   or    [Port_1_Data_SHADE],LCD_RW              ; Raise RW to signify Read operation
   mov   A,[Port_1_Data_SHADE]
   mov   reg[LCD_Port],A
   NOP

LCD_RDY_LOOP:
   ;or    [Port_1_Data_SHADE], LCD_CNTL_READ      ; Raise E to start cycle
```

```
    mov   A,[Port_1_Data_SHADE]
    or      A,LCD_CNTL_READ
    or      A,[_lcd_e]
    mov   reg[LCD_Port],A

    nop                                         ; Wait 2 nops to make sure data is ready
    nop
    mov   A,reg[LCD_Port]

; The code below is used to work around the async read issue with the ICE with the
; 25/26xxx family of devices. It will help to eliminate "Invalid memory reference"
; errors. It is not required when running without the ICE or when using any other
; family besides the 25/26xxx family. If not using the ICE or with any other family
; the ICE_PORT_SYNC flag should be set to 0.
IF(ICE_PORT_SYNC)
    mov   reg[ 0xfa], A
    mov   A, reg[0xfa]
ENDIF

    push  A
    and   [Port_1_Data_SHADE],(~LCD_PORT_MASK | LCD_RW)       ; Lower E signal
    mov   A,[Port_1_Data_SHADE]
    mov   reg[LCD_Port],A

    nop                             ; Add delay for the slowest part and the
    nop                             ; fastest PSoC
    nop
                                            ; Get the LSBs
    ;or    [Port_1_Data_SHADE],LCD_CNTL_READ                ; Raise E to start cycle
    mov     A,[Port_1_Data_SHADE]
    or    A,LCD_CNTL_READ
    or    A,[_lcd_e]
    ;mov   A,[Port_1_Data_SHADE]
    mov   reg[LCD_Port],A

    nop
    nop

    and   [Port_1_Data_SHADE],(~LCD_PORT_MASK | LCD_RW)       ; Lower E signal
    mov   A,[Port_1_Data_SHADE]
    mov   reg[LCD_Port],A

    pop   A
    and   A,LCD_READY_BIT                           ; Check busy
    jnz   LCD_RDY_LOOP                              ; If LCD still busy, read again

    or    [Port_1_DriveMode_0_SHADE],LCD_PORT_WRITE          ; Revert Data bit to Write mode
    mov   A,[Port_1_DriveMode_0_SHADE]
    M8C_SetBank1
    mov   reg[LCD_PortMode0],A                      ; Setup LCD Port for writing
    M8C_SetBank0
    pop   A
    RAM_EPILOGUE RAM_USE_CLASS_4                            ; Restore Accumulator
    ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_Start
;  FUNCTION NAME: LCD_1_Init
;
;  DESCRIPTION:
;     Initialize LCD
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS: none
```

```
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
;  THEORY of OPERATION or PROCEDURE:
;    REGISTERS ARE VOLATILE: THE A AND X REGISTERS MAY BE MODIFIED!
;    This initialization is a bit long, but it should work for
;    most 2 and 4 line LCDs.
;
 LCD_1_Start:
_LCD_1_Start:
 LCD_1_Init:
_LCD_1_Init:
    RAM_PROLOGUE RAM_USE_CLASS_4
    RAM_SETPAGE_CUR >Port_1_Data_SHADE          ; Set CUR_PP to LCD variable address

    and   [Port_1_DriveMode_0_SHADE],~LCD_PORT_MASK            ; Mask off LCD bits
    or    [Port_1_DriveMode_0_SHADE],LCD_PORT_WRITE            ; Set LCD port for writing
    and   [Port_1_DriveMode_1_SHADE],~LCD_PORT_MASK            ; Mask off LCD bits

    mov   A,[Port_1_DriveMode_0_SHADE]
    M8C_SetBank1
    mov   reg[LCD_PortMode0],A                          ; Setup LCD Port for writing
    mov   A,[Port_1_DriveMode_1_SHADE]
    mov   reg[LCD_PortMode1],A
    M8C_SetBank0

    mov   A,250                         ; Delay for 12.5 mSec (250 * 50uSec)
    call  LCD_1_Delay50uTimes
    mov   A,250                         ; Delay for 12.5 mSec (250 * 50uSec)
    call  LCD_1_Delay50uTimes

    mov   A,03h
    call  LCD_WCNTL_Nibble

    mov   A,82                        ; Delay for 4.1 mSec (82 * 50uSec)
    call  LCD_1_Delay50uTimes

    mov   A,03h
    call  LCD_WCNTL_Nibble

    call  LCD_1_Delay50u
    call  LCD_1_Delay50u
    call  LCD_1_Delay50u

    mov   A,03h
    call  LCD_WCNTL_Nibble

    mov   A,90                        ; Delay for 4.5 mSec (90 * 50uSec)
    call  LCD_1_Delay50uTimes

    mov   A,02h
    call  LCD_WCNTL_Nibble

    mov   A,90                        ; Delay for 4.5 mSec (90 * 50uSec)
    call  LCD_1_Delay50uTimes

    mov   A,08h
```

```
   call  LCD_1_Control
   mov   A,90                    ; Delay for 4.5 mSec (90 * 50uSec)
   call  LCD_1_Delay50uTimes

   mov   A,01h
   call  LCD_1_Control
   mov   A,90                    ; Delay for 4.5 mSec (90 * 50uSec)
   call  LCD_1_Delay50uTimes

   mov   A,06h
   call  LCD_1_Control

   mov   A,0Eh
   call  LCD_1_Control

   mov   A,LCD_4BIT_2LINE        ; Setup for 4 bit interface, 2 line
   call  LCD_1_Control

   mov   A,DISP_OFF
   call  LCD_1_Control

   mov   A,DISP_ON
   call  LCD_1_Control

   mov   A,DISP_INC
   call  LCD_1_Control

   mov   A,90                    ; Delay for 4.5 mSec (90 * 50uSec)
   call  LCD_1_Delay50uTimes
   RAM_EPILOGUE RAM_USE_CLASS_4
   ret
.ENDSECTION

;---------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_Position
;
;  DESCRIPTION:
;    Position Cursor at Row and Col location
;
;---------------------------------------------------------------------------
;
;  ARGUMENTS:
;    A => Row  0 to 3
;    X => Col  0 to 39+
;
;  RETURNS:  none
;
;  SIDE EFFECTS:
;   The A and X registers may be modified by this or future implementations
;   of this function. The same is true for all RAM page pointer registers in
;   the Large Memory Model. When necessary, it is the calling function's
;   responsibility to perserve their values across calls to fastcall16
;   functions.
;
.LITERAL
LCD_ROW_OFFSET::
    DB    LCD_ROW1_OFFSET, LCD_ROW2_OFFSET, LCD_ROW3_OFFSET, LCD_ROW4_OFFSET
.ENDLITERAL

.SECTION
 LCD_1_Position:
_LCD_1_Position:
   RAM_PROLOGUE RAM_USE_CLASS_2
   and   A,03h                   ; Mask off 2 bits for row address 0 to 3
   push  X                       ; Store COL
   index LCD_ROW_OFFSET          ; Get ROW memory offset from table
   mov   X,SP                    ; Get Stack pointer
```

```
    add   A,[X+(-1)]                ; Add the COL to the display pointer
    pop   X

LCD_POS_IT:
    call  LCD_1_Control             ; Write control byte
    RAM_EPILOGUE RAM_USE_CLASS_2
    ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_Delay50uTimes
;
;  DESCRIPTION:
;     Delay increments of 50uSeconds
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;     A contains the delay multiplier
;
;  RETURNS:
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;
 LCD_1_Delay50uTimes:
_LCD_1_Delay50uTimes:
    RAM_PROLOGUE RAM_USE_CLASS_1
    call  LCD_1_Delay50u
    dec   A
    jnz   LCD_1_Delay50uTimes
    RAM_EPILOGUE RAM_USE_CLASS_1
    ret

.ENDSECTION

;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_Delay50u
;
;  DESCRIPTION:
;     Delay 50uSec for any clock frequency from 1.5MHz to 24MHz
;     Slower clock frequencies the delay will be;
;          1.5
;        ------------- * 50uSec
;        clock_freq(MHz)
;
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS: none
;
;  RETURNS: none
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;  THEORY of OPERATION or PROCEDURE:
```

```
;
.LITERAL
 Delay50u_Table::
    DB   08h,  19h,   3Ah,   7Ch,   01h,    01h,    01h,   01h
;       3MHz, 6MHz, 12MHz, 24MHz, 1.5MHz, 750kHz, 188kHz, 94kHz
.ENDLITERAL
.SECTION

  LCD_1_Delay50u:
 _LCD_1_Delay50u:                    ; [11]  Call
   RAM_PROLOGUE RAM_USE_CLASS_1
   push  A
   M8C_SetBank1                      ; [4]
   mov   A, reg[OSC_CR0]               ; [6] Get delay value
   M8C_SetBank0                      ; [4]
   and   A,07h                      ; [4] Mask off only the clock bits
   cmp   A,05h
   jnc   Delay50u_End
   index Delay50u_Table                ; [13] Get delay value
Delay50u_Loop:                      ;
   dec   A                          ; [4]
   jnz   Delay50u_Loop                 ; [5]
Delay50u_End:
   pop   A
   RAM_EPILOGUE RAM_USE_CLASS_1
   ret
.ENDSECTION


;-----------------------------------------------------------------------------
;     If bargraph is not enabled, the following functions are not required.
;-----------------------------------------------------------------------------

IF (LCD_1_BARGRAPH_ENABLE)

IF SYSTEM_SMALL_MEMORY_MODEL
.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_DrawBG
;
;  DESCRIPTION:
;  This legacy fastcall version are provided only to support existing small
;  memory model assembly language code---it does not work in the large memory
;  model.
;
;  ** This legacy fastcall version is provided on a temporary basis to
;  ** ease the transition to the 4.2 release of PSoC Designer. Its use is
;  ** deprecated and its status is "No Further Maintenance". If you call this
;  ** function in assembly you should convert to _LCD_1_DrawVBG
;  ** (with a leading underscore) and the fastcall16 interface
;
;  Draw a horizontal bargraph on the LCD with the given parameters. This
;  is a legacy function that is intended to support existing Assembly
;  language programs that call this function. This should not be used for
;  new code or with Large Memory Model programs.
;-----------------------------------------------------------------------------
;
;  LEGACY FASTCALL ARGUMENTS:
;    A    => Starting row for bargraph 0 to 3
;    [X]   => Starting Column for bargraph 0 to 39+
;    [x-1] => Length of bargraph in chars 1 to 40+
;    [X-2] => Position of pointer in segments 5 times Length
;
;
;  RETURNS:  none
;
;  SIDE EFFECTS:
```

```
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    If LCD_1_Init is not called before this function, the
;    bargraph will not be drawn properly.
;
; Stack offset constants
BG_COLX:       equ  0                ; Stack position of Column
BG_CHAR_LENX:  equ -1                ; Stack position of Length
BG_LENGTHX:    equ -2                ; Stack postion of bargraph pointer position


LCD_1_DrawBG:
   push  X
   mov   X,[X+BG_COLX]               ; Row in A, Col in X
   call  LCD_1_Position              ; Set cursor position
   pop   X                           ; Restore pointer

LCD_BG_LOOP1X:
   cmp   [X+BG_LENGTHX],00h          ; Check for past end of BG
   jnz   LCD_CHECK1X
   mov   A,00h                       ; Load empty character
   jmp   LCD_BG_DOITX                ;

LCD_CHECK1X:
   cmp   [X+BG_LENGTHX],06h          ; Check if BG pointer is at this character
   jnc   LCD_CHECK2X                 ; Note yet, use full character
   mov   A,[X+BG_LENGTHX]
   sub   [X+BG_LENGTHX],A
   jmp   LCD_BG_DOITX

LCD_CHECK2X:                         ; Put index to full character
   mov   A, 06h
   sub   [X+BG_LENGTHX],05h          ; Subtract another 5 positions

LCD_BG_DOITX:
   call  LCD_1_WriteData             ; Display BG character

   dec   [X+BG_CHAR_LENX]            ; Dec Char count
   jnz   LCD_BG_LOOP1X               ; Do it all over again
   ret
.ENDSECTION
ENDIF ; SYSTEM_SMALL_MEMORY_MODEL

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_DrawBG
;
;  DESCRIPTION:
;     Draw a horizontal bargraph on the LCD with the given parameters.
;
;
;-----------------------------------------------------------------------------
;
;  FASTCALL16 ARGUMENTS:
;   [SP-3] => Starting row for bargraph 0 to 3
;   [SP-4] => Starting Column for bargraph 0 to 39+
;   [SP-5] => Length of bargraph in chars 1 to 40+
;   [SP-6] => Position of pointer in segments 5 times Length
;
;
;  RETURNS:  none
;
;  SIDE EFFECTS:
```

```
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
;    If LCD_1_Init is not called before this function, the
;    bargraph will not be drawn properly.
;
; Stack offset constants
BG_ROW:       equ -3
BG_COL:       equ -4                ; Stack position of Column
BG_CHAR_LEN:  equ -5                ; Stack position of Length
BG_LENGTH:    equ -6                ; Stack postion of bargraph pointer position


_LCD_1_DrawBG:
   RAM_PROLOGUE RAM_USE_CLASS_2
   mov   X, SP
   push  X
   mov   A,[X+BG_ROW]               ; Row in A
   mov   X,[X+BG_COL]               ; Col in X
   call  LCD_1_Position             ; Set cursor position
   pop  X

LCD_BG_LOOP1:
   cmp   [X+BG_LENGTH],00h          ; Check for past end of BG
   jnz   LCD_CHECK1
   mov   A,00h                   ; Load empty character
   jmp   LCD_BG_DOIT                ;

LCD_CHECK1:
   cmp   [X+BG_LENGTH],06h          ; Check if BG pointer is at this character
   jnc   LCD_CHECK2                 ; Note yet, use full character
   mov   A,[X+BG_LENGTH]
   sub   [X+BG_LENGTH],A
   jmp   LCD_BG_DOIT

LCD_CHECK2:                         ; Put index to full character
   mov   A, 06h
   sub   [X+BG_LENGTH],05h          ; Subtract another 5 positions

LCD_BG_DOIT:
   call  LCD_1_WriteData            ; Display BG character

   dec   [X+BG_CHAR_LEN]            ; Dec Char count
   jnz   LCD_BG_LOOP1               ; Do it all over again
   RAM_EPILOGUE RAM_USE_CLASS_2
   ret
.ENDSECTION

IF SYSTEM_SMALL_MEMORY_MODEL
.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_DrawVBG
;
;  DESCRIPTION:
;  This legacy fastcall version are provided only to support existing small
;  memory model assembly language code---it does not work in the large memory
;  model.
;
;  ** This legacy fastcall version is provided on a temporary basis to
;  ** ease the transition to the 4.2 release of PSoC Designer. Its use is
;  ** deprecated and its status is "No Further Maintenance". If you call this
```

```
;  ** function in assembly you should convert to _LCD_1_DrawVBG
;  ** (with a leading underscore) and the fastcall16 interface
;
;  Draw a vertical bargraph on the LCD with the given parameters. This
;  is a legacy function that is intended to support existing Assembly
;  language programs that call this function. This should not be used for
;  new code or with Large Memory Model programs.
;------------------------------------------------------------------------------
;
;  LEGACY FASTCALL ARGUMENTS:
;    A    => Starting row for bargraph 0 to 3
;   [X]   => Starting Column for bargraph 0 to 40+
;   [x-1] => Height of bargraph in chars 1 - 4
;   [X-2] => Position of pointer in segments 8 times height
;  RETURNS:
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    If LCD_1_Init is not called before this function, the
;    bargraph will not be drawn properly.
;
; Stack offset constants
VBG_COLX:           equ  0
VBG_CHAR_HEIGHTX:   equ -1
VBG_SEG_HEIGHTX:    equ -2

LCD_1_DrawVBG:

   and   A,03h                    ; Make sure only rows 0 - 3 are valid
VBG_LOOPX:
   push  A
   index LCD_ROW_OFFSET           ; Get row offset
   add   A,[X+VBG_COLX]           ; Add column offset to position
   call  LCD_1_Control            ; Position Cursor
   cmp   [X+VBG_SEG_HEIGHTX],00h  ; Check for zero segs
   jnz   VBG_NZ_SEGX
   mov   A,' '                    ; Load space character
   jmp   VBG_WRITE_CHARX
VBG_NZ_SEGX:
   cmp   [X+VBG_SEG_HEIGHTX],09h  ; Check for full segment
   jnc   VBG_FULL_SEGX
                                  ; Partial segment between 1 and 8
   mov   A,[X+VBG_SEG_HEIGHTX]
   dec   A
   mov   [X+VBG_SEG_HEIGHTX],00h  ; Zero segment height
   jmp   VBG_WRITE_CHARX

VBG_FULL_SEGX:                    ; Bargaph
   sub   [X+VBG_SEG_HEIGHTX],08h  ; Subtract full segment
   mov   A,07h                    ; Load full segment

VBG_WRITE_CHARX:                  ; Write character to display
   call  LCD_1_WriteData          ; Write value
   pop   A
   dec   A
   dec   [X+VBG_CHAR_HEIGHTX]
   jnz   VBG_LOOPX
   ret
.ENDSECTION
ENDIF ; SYSTEM_SMALL_MEMORY_MODEL

.SECTION
```

```
;-------------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_DrawVBG
;
;  DESCRIPTION:
;    Draw a vertical bargraph on the LCD with the given parameters.
;
;
;-------------------------------------------------------------------------------
;
;  FASTCALL16 ARGUMENTS:
;
;   [SP-3] => Starting row for bargraph 0 to 3
;   [SP-4] => Starting Column for bargraph 0 to 40+
;   [SP-5] => Height of bargraph in chars 1 - 4
;   [SP-6] => Position of pointer in segments 8 times height
;  RETURNS:
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
;    If LCD_1_Init is not called before this function, the
;    bargraph will not be drawn properly.
;
; Stack offset constants
VBG_ROW:        equ -3
VBG_COL:        equ -4
VBG_CHAR_HEIGHT:    equ -5
VBG_SEG_HEIGHT:     equ -6

_LCD_1_DrawVBG:
   RAM_PROLOGUE RAM_USE_CLASS_2
   mov   X, SP
   mov   A, [X+VBG_ROW]
   and   A,03h                  ; Make sure only rows 0 - 3 are valid
VBG_LOOP:
   push  A
   index LCD_ROW_OFFSET         ; Get row offset
   add   A,[X+VBG_COL]          ; Add column offset to position
   call  LCD_1_Control          ; Position Cursor
   cmp   [X+VBG_SEG_HEIGHT],00h     ; Check for zero segs
   jnz   VBG_NZ_SEG
   mov   A,' '                  ; Load space character
   jmp   VBG_WRITE_CHAR
VBG_NZ_SEG:
   cmp   [X+VBG_SEG_HEIGHT],09h     ; Check for full segment
   jnc   VBG_FULL_SEG
                                ; Partial segment between 1 and 8
   mov   A,[X+VBG_SEG_HEIGHT]
   dec   A
   mov   [X+VBG_SEG_HEIGHT],00h     ; Zero segment height
   jmp   VBG_WRITE_CHAR

VBG_FULL_SEG:                   ; Bargaph
   sub   [X+VBG_SEG_HEIGHT],08h     ; Subtract full segment
   mov   A,07h                  ; Load full segment

VBG_WRITE_CHAR:                 ; Write character to display
   call  LCD_1_WriteData        ; Write value
   pop   A
   dec   A
```

```
    dec   [X+VBG_CHAR_HEIGHT]
    jnz   VBG_LOOP
    RAM_EPILOGUE RAM_USE_CLASS_2
    ret
.ENDSECTION

.SECTION
;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_InitVBG
;
;  DESCRIPTION:
;     Initialize the vertical bargraph characters.
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:  none
;
;  RETURNS:  none
;
;  SIDE EFFECTS:
;    REGISTERS ARE VOLATILE: THE A AND X REGISTERS MAY BE MODIFIED!
;    Only one type of bargraph (horizontal or vertical) may be used
;    at a time since they each require their own set of characters.
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
; Stack offset constants
VBGDATA_CTR:      equ    00h          ; Char data count stack offset
VBG_BYTES:        equ    01h          ; Byte counter stack offset

 LCD_1_InitVBG:
_LCD_1_InitVBG:
    RAM_PROLOGUE RAM_USE_CLASS_2
    mov   X,SP                        ; Get location of stack
    push  A                           ; Create 2 locations
    push  A

    mov   A,CG_RAM_OFFSET             ; Setup pointer
    call  LCD_1_Control               ; Position the CG pointer
    mov   [X+VBGDATA_CTR],01h         ; Reset data counter

 VBG_Loop1:                           ; loop once for each 8 characters
    mov   [X+VBG_BYTES],08h           ; Load cycle pointer
 VBG_Loop2:                           ; Loop once for each line in character (8 times)
    mov   A,[X+VBGDATA_CTR]
    cmp   A,[X+VBG_BYTES]
    jnc   VBG_SOLID
    mov   A,00h                       ; Empty line
    jmp   VBG_Load                    ; Jump to load the bargraph
VBG_SOLID:
    mov   A,FFh                       ; Load solid line
VBG_Load:
    call  LCD_1_WriteData             ; character data
    dec   [X+VBG_BYTES]               ; Dec byte counter
    jnz   VBG_Loop2                   ; End Loop 2
    inc   [X+VBGDATA_CTR]
    cmp   [X+VBGDATA_CTR],09h
    jnz   VBG_Loop1                   ; End Loop1
```

```
    pop  A
    pop  A
    mov  A,DISP_ON                  ; Turn on display, don't really
    call LCD_1_Control              ; need this.
    RAM_EPILOGUE RAM_USE_CLASS_2
    ret
.ENDSECTION

;-----------------------------------------------------------------------------
;  FUNCTION NAME: LCD_1_InitBG
;
;  DESCRIPTION:
;     Initialize horizontal bargraph characters
;
;-----------------------------------------------------------------------------
;
;  ARGUMENTS:
;     A = type  0 = full                |||||||||.........
;               1 = single vertical line   ..........|......
;
;  RETURNS:
;
;  SIDE EFFECTS:
;    The A and X registers may be modified by this or future implementations
;    of this function. The same is true for all RAM page pointer registers in
;    the Large Memory Model. When necessary, it is the calling function's
;    responsibility to perserve their values across calls to fastcall16
;    functions.
;
;    Currently only the page pointer registers listed below are modified:
;         CUR_PP
;
;    Only one type of bargraph (horizontal or vertical) may be used
;    at a time since they each require their own set of characters.
;
;  THEORY of OPERATION or PROCEDURE:
;    This function writes to the LCD character RAM to generate 8 custom
;    characters used to generated one of two horizontal bargraphs.
;
.LITERAL
 BG_TYPE1:: ; ....., |...., ||..., |||.., ||||., |||||, |||||
     DB    00h, 10h, 18h, 1Ch, 1Eh, 1Fh, 1Fh
 BG_TYPE2:: ; ....., |...., .|..., ..|.., ...|., ....|, .....
    DB  00h, 10h, 08h, 04h, 02h, 01h, 00h
.ENDLITERAL

.SECTION
; Stack offset constants
BGDATA_PTR:   equ    00h              ; Stack offsets
BGCHARS:      equ    01h
BGTYPE:       equ    02h

 LCD_1_InitBG:
_LCD_1_InitBG:
    RAM_PROLOGUE RAM_USE_CLASS_2
    mov   X,SP                      ; Get location of stack
    add   SP,3
    mov   [X+BGTYPE],A              ; Store the bargraph type

    mov   A,CG_RAM_OFFSET             ; Setup pointer
    call  LCD_1_Control             ; Position the CG pointer
    mov   [X+BGDATA_PTR],00h          ; Reset pointer to BG data

 BG_Loop1:
    mov   [X+BGCHARS],08h             ; Load cycle pointer
 BG_Loop2:
    mov   A,[X+BGDATA_PTR]
```

```
    cmp   [X+BGTYPE],00h              ; Check which bargraph
    jnz   BG_OTHER
    index BG_TYPE1
    jmp   BG_Load
 BG_OTHER:
    index BG_TYPE2
 BG_Load:
    call  LCD_1_WriteData
    dec   [X+BGCHARS]                 ; Character builder counter
    jnz   BG_Loop2
    inc   [X+BGDATA_PTR]              ; Advance to next character
    cmp   [X+BGDATA_PTR],07h
    jnz   BG_Loop1

    add   SP,-3
    mov   A,DISP_ON
    call  LCD_1_Control
    RAM_EPILOGUE RAM_USE_CLASS_2
    ret
.ENDSECTION

ENDIF

; End of File LCD_1.asm
```