90

The Importance of Aluminum Oxide Aerosols to Stratospheric Ozone Depletion

by

Darryl Day Spencer

B.A. Chemistry
University of Utah
(1991)

Submitted to the Department of Chemistry in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Chemistry

at the

Massachusetts Institute of Technology

June 1996

Signature of Author ....[illegible]........
Department of Chemistry
May 22, 1996

Certified by ....[illegible]........
Mario J. Molina
Martin Professor of Environmental Chemistry
Thesis Supervisor

Accepted by ....[illegible]........
Dietmar Seyferth
Professor of Chemistry
Chairman, Departmental Committee on Graduate Studies

This doctoral thesis has been examined by a committee of the Department of Chemistry as Follows:

Professor Jeffrey I. Steinfeld .... [illegible]
Chairman

Professor Mario J. Molina .... [illegible]
Thesis Supervisor

Professor Ronald G. Prinn .... [illegible]
Department of Earth, Atmospheric and Planetary Sciences

The Importance of Aluminum Oxide Aerosols to Stratospheric Ozone Depletion

by

Darryl Day Spencer

Submitted to the Department of Chemistry on May 22, 1996
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy in Chemistry

Abstract

Experiments and calculations were conducted to estimate the stratospheric ozone depletion potential of α-alumina (aluminum oxide) in solid rocket motor exhaust, such as from the boosters used by the space shuttle, compared to the HCl exhausted by the same rockets. The reaction probability, γ, for the chlorine activation reaction $ClONO_2 + HCl \rightarrow Cl_2 + HNO_3$ was measured on aluminum oxide, glass, 60% sulfuric acid, and Teflon surfaces. The amount and type of adsorption of HCl, $ClONO_2$, and H_2O was also measured on α-alumina and glass.

The results indicate that alumina and glass have a γ of approximately 0.02. These surfaces were totally deactivated by coating with 60% sulfuric acid. Teflon also had an undetectable low γ. The alumina released from a space shuttle launch would take about 6 months to activate the amount of HCl released by the same launch. Hence, it is likely for heterogeneous chemistry on alumina to significantly affect the ozone depletion potential of each shuttle launch or other solid fuel rocket launches and so it should be included in stratospheric ozone models.

HCl chemically and physically adsorbs to the alumina surfaces with hysteresis. HCl adsorption on glass is reversible. H_2O adsorption on both surfaces is reversible. $ClONO_2$ shows adsorption with slight hysteresis on both materials that may be from hydrolysis with adsorbed layers of H_2O on the surface. It may be that any refractory, hydrophilic material with surface hydroxyl groups would have similar values of γ.

Thesis Supervisor: Mario J. Molina
Title: Martin Professor of Environmental Chemistry

Table of Contents

Acknowledgments

I would like to first thank Prof. Molina for providing space and funding for me in his laboratory, and for his support in my finishing my Ph.D. The classes that I enjoyed taking the most were the ones I took during my second year and were taught by my committee members (which is one of the reasons I chose them). Prof. Steinfeld taught an advanced kinetics course and Prof. Prinn and Prof. Molina co-taught an atmospheric chemistry and physics course that I have been involved with as a teaching assistant ever since.

I'd like to thank those other teachers and mentors who made it possible for me to achieve my dream of coming to a school like MIT. I'd like to mention my high school chemistry teacher, Richard Smith. It was he who installed a fascination with chemistry within me and demanded performance in its study. Prof. Ragsdale and Dr. Driscole at the University of Utah gave me my first real laboratory experience during a summer program for high school students. I'd especially like to thank Prof. Eyring at the University of Utah for being the first to invite me to work for him in his lab. He's been a good friend to get feedback on ideas ever since. Dr. Michael D. Morse was also gracious enough to let me change labs and work for him so I could broaden my experience.

I'd like to thank the people in our lab for the good things they've done for me. Paul Wooldridge always knew a reference when you needed it. Roger Meads knew how to get things to work quickly. John Seeley kept up my sense of humor and was a good example by graduating. Matt Elrod gave very critical, but fair evaluation of ideas I presented him. Danna Leard often had good advice and was a good listener to sort ideas out with. I'd like to thank Rose Koch and Luisa Molina for the work they did which indirectly or directly helped me. Dora Farkas was a UROP who wouldn't quit like other undergraduates when the details got complicated.

I'd like to thank those who gave me the emotional support I've needed to work here. My parents supported me in my choices for more than two decades. Now, although marriage places demands on a graduate student's time, I could not have lived so far from home and friends without my wife Alycia. She's a great listener. My daughter Amelia is a joy. Finally, I'd like to thank my God and my Savior for touching my heart when I reached out in prayer for comfort and strength in times of discouragement.

Chapter I: Introduction

I.1 Review of Ozone Homogeneous and Heterogeneous Chemistry

At an altitude between approximately 15 kilometers to 60 kilometers above the earth's surface lies the stratosphere. In contrast to where we live in the troposphere, it is very inhospitable. It can be very dry (~5 ppmv water), very cold (as low as -78°C), and is exposed to larger amounts of ultra-violet radiation. Yet, it is very beneficial to our lives because it is where the ozone layer is found.

The ozone layer shields the earth from ultra-violet radiation (wavelength 240-320 nm). In a continuous cycle, ozone is continually formed, destroyed, and reformed. The balance between the rates of formation and destruction determine the amount of ozone in the stratosphere. The first mechanism proposed for this process was by Chapman (1930):

$$O_2 + h\nu_1 \rightarrow O + O$$

$$O + O_2 + M \rightarrow O_3 + M$$

$$O_3 + h\nu_2 \rightarrow O_2 + O$$

$$O + O_3 \rightarrow 2O_2$$

Thus two ultra-violet photons of different wavelengths are absorbed and are converted into heat in this process. This heating, in fact, results in a temperature inversion that creates the boundary between the troposphere and the stratosphere.

The amount of ozone and energy involved in these reactions are very large. However, there are other important cycles with trace amounts of chemicals in which man can have an impact. In 1995, the Swedish Academy of Science awarded the Nobel Prize in Chemistry to Crutzen, Rowland, and Molina. Crutzen [Crutzen, 1971] studied the impact of nitrous oxides on the destruction of ozone through the following catalytic cycle:

$$NO + O_3 \rightarrow NO_2 + O_2$$

$$NO_2 + O \rightarrow NO + O_2$$

$$\text{Net: } O_3 + O \rightarrow O_2 + O_2$$

In 1974, Rowland and Molina [Molina and Rowland, 1974] connected chlorofluorocarbons (CFC's) to ozone depletion. High energy ultra-violet photons could cleave the chlorine bonds to form chlorine radicals in the stratosphere. These radicals could then catalytically destroy ozone through the following cycle:

$$Cl + O_3 \rightarrow ClO + O_2$$

$$ClO + O \rightarrow Cl + O_2$$

$$\text{Net: } O_3 + O \rightarrow O_2 + O_2$$

The expectation from homogeneous gas phase chemistry was a gradual decrease in global ozone as depicted in Figure I.1. However, during the 1980's, substantial ozone loss was observed over the Antarctic during the polar springtime as depicted in Figure I.2. Figure I.3 depicts how the phenomenon covered larger areas each year. This amount of ozone depletion could not be accounted for with only homogeneous gas phase chemistry. The study of the Antarctic ozone hole (Molina et al., 1987; Solomon, 1990; Abbatt et al., 1993; Shen et al., 1995) lead to the discovery of new chemical mechanisms that could also destroy ozone.

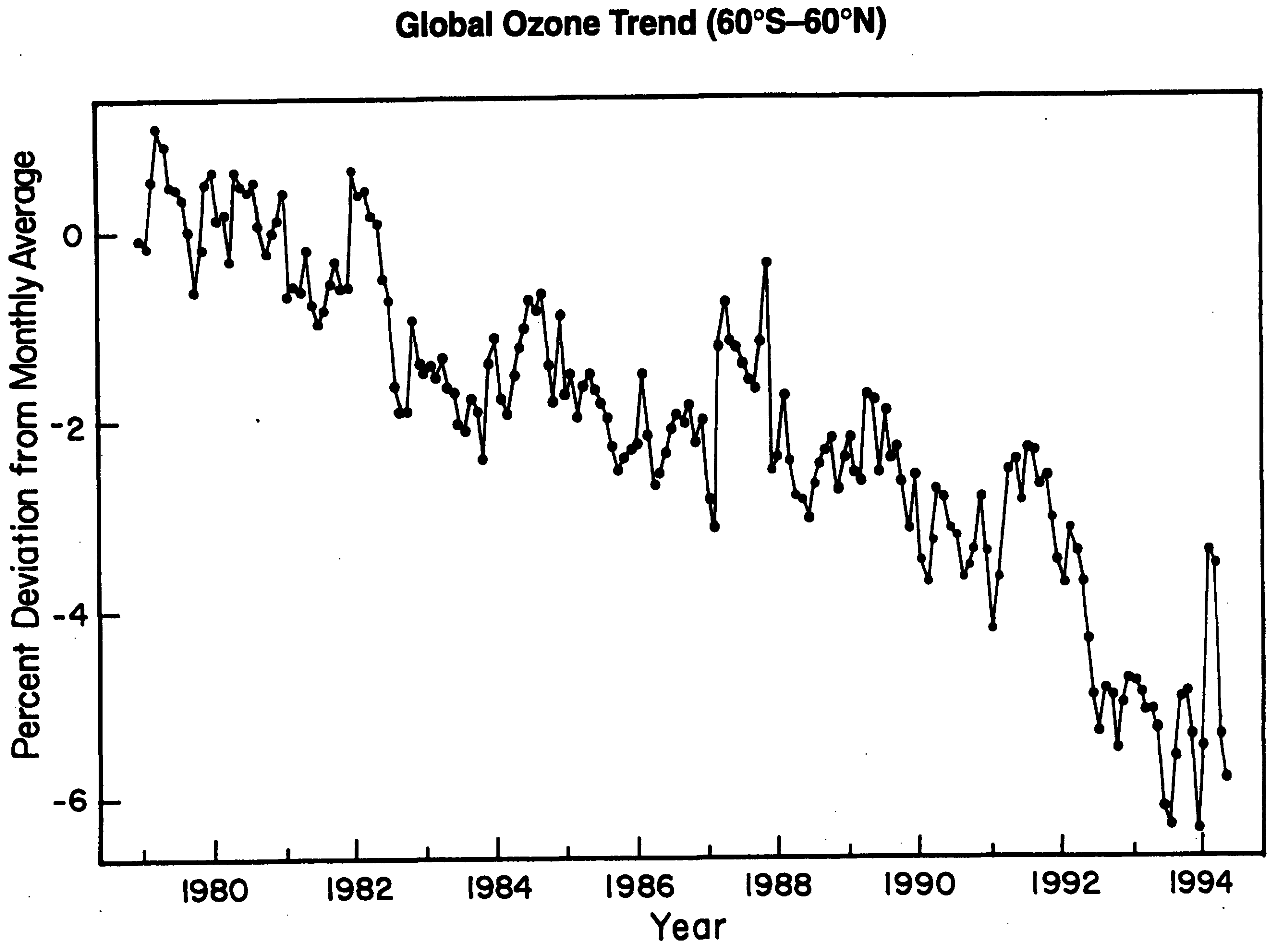


Figure I.1. Gradual global decline in total ozone column measurements [World Meteorological Association, 1994].

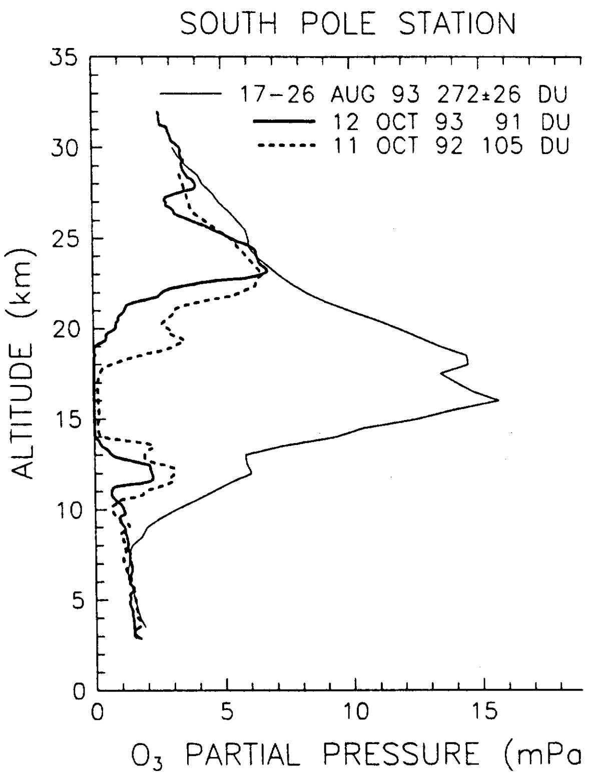


Figure I.2. Dramatic ozone depletion during polar springtime [World Meteorological Association, 1994].

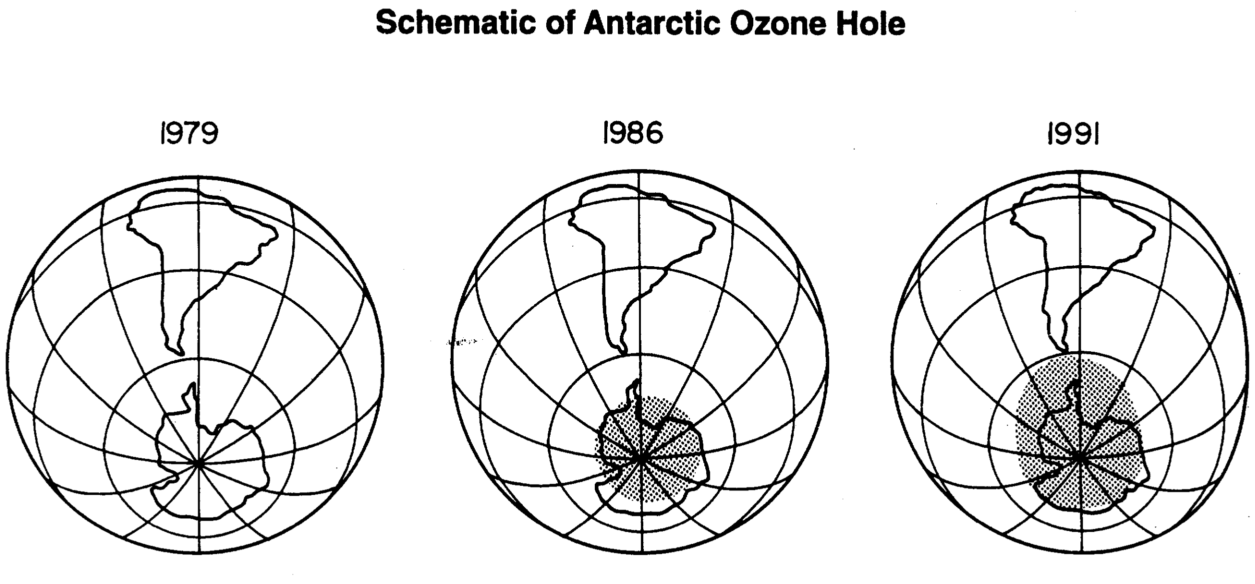


Figure I.3. Schematic of rapid appearance and growth of the Antarctic Ozone Hole [World Meteorological Association, 1994].

Not all chlorine in the stratosphere is in the form of radicals. Nitrogen compounds can tie up chlorine radicals as chlorine nitrate through the following reaction:

$$ClO + NO_2 + M \rightarrow ClONO_2 + M$$

Chlorine radicals can react to form HCl upon reaction with stratospheric methane:

$$Cl + CH_4 \rightarrow HCl + CH_3$$

These compounds, $ClONO_2$ and HCl, are not active in the catalytic destruction of ozone. Thus, Cl and ClO are called "active" compounds while HCl and $ClONO_2$ are called "reservoir" compounds. The more chlorine is partitioned in active forms rather than the reservoir forms, the more ozone loss there will be.

During the Antarctic winter, a polar vortex forms. The wind circulates around the continent and prevents mixing with the rest of the stratosphere. Temperatures drop to minus 78°C. At these temperatures, polar stratospheric clouds (PSC's) form from ice and nitric acid trihydrate (NAT). Upon the surfaces of these clouds, a heterogeneous reaction with the gas phase chlorine reservoir compounds occurs that turns them to active form:

$$HCl + ClONO_2 \rightarrow HNO_3 + Cl_2$$

The resulting chlorine gas molecules readily photolyze upon exposure to the first light rays of spring to form the chlorine radicals. Due to the lack of oxygen atoms in the weak sunlight, ozone loss is accomplished by another mechanism.

$$Cl_2 + h\nu \rightarrow 2Cl$$

$$2(Cl + O_3 \rightarrow ClO + O_2)$$

$$ClO + ClO \rightarrow (ClO)_2$$

$$\underline{(ClO)_2 + M \rightarrow Cl_2 + O_2 + M}$$

$$\text{Net Reaction: } 2O_3 \rightarrow 3O_2$$

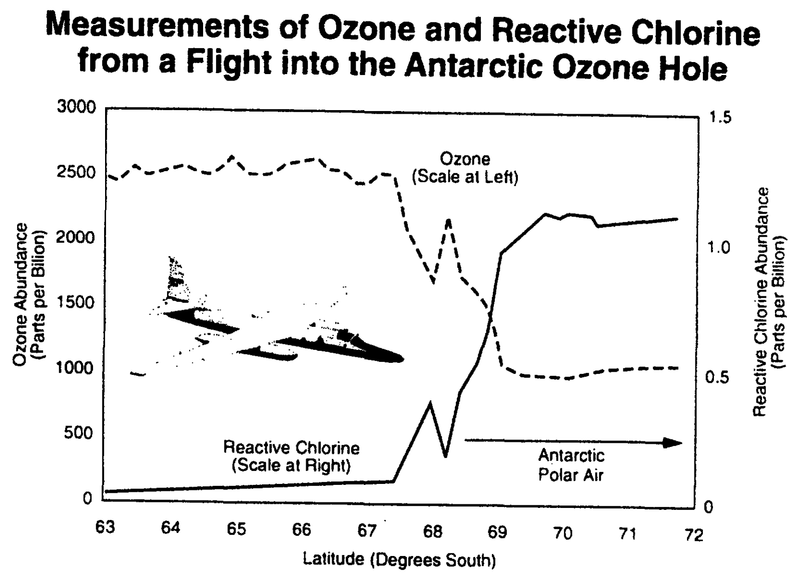


Figure I.4. Strong anti-correlation of ClO and O_3 levels give strong evidence that chlorine chemistry is responsible for ozone hole [World Meteorological Association, 1994].

In addition to PSC's, there is a background stratospheric sulfate aerosol (SSA) layer. Inclusion of both sources of aerosols have lead to better agreement with observed ozone depletion and partitioning of NO_X (Chipperfield and Pyle, 1988; Solomon, 1988; Hofmann and Solomon, 1989; Isaksen et al., 1990; Rodriguez et al., 1991; Brasseur and Granier, 1992; Considine et al., 1994; Solomon et al., 1996). The main effect of the SSA layer is on the hydrolysis of N_2O_5 to HNO_3 (Hanson et al, 1994) in the following important process:

$$NO_2 + O_3 \rightarrow NO_3 + O_2$$

$$NO_3 + NO_2 \rightarrow N_2O_5$$

$$N_2O_5 + H_2O \rightarrow 2HNO_3$$

After the N_2O_5 hydrolyzes in the final step, the nitric acid can photolyze to form NO_2 again to deplete more ozone.

The anthropogenic injection of chlorine into the stratosphere from solid rocket motors became a source of serious environmental concern in the 1970's. The possible threat from chlorine in solid rocket motors, including the shuttle was recognized as part of the Climate Impact Assessment Program (Hoshizaki, 1975). Solid rocket motors, including the booster rockets used on the space shuttle, use aluminum powder as the main fuel and oxidize it with ammonium perchlorate (NH_4ClO_4). The third major component in the fuel is poly-isobutylene, a synthetic rubber, which is the matrix that holds the powders together in the solid mold. The aluminum forms alumina (Al_2O_3) and the chlorine is mostly exhausted as HCl. Potter (1978) calculated possible ozone losses in the northern hemisphere from the shuttle's HCl to be 0.25% and the southern hemisphere would experience 0.025% to 0.05% for a schedule of 60 launches per year. Prather (Prather et al., 1990) did an extensive study using only homogeneous chemistry. Each shuttle launch exhauts 68 tons of chlorine into the stratosphere. The comparable value for the Titan IV rocket is 32 tons of chlorine, for the Ariane-5, 57 tons (based on Pyale and Jones, 1991).

Recent work on the shuttle's or other solid rocket motor's influence has concentrated on local effects from homogeneous chemistry with exhausted HCl in the rocket plume. Aftergood [1991], hypothesized that there could be significant loss of column ozone after a space shuttle launch. Model simulations by Karol et al. [1992], Danilin [1993] and Kruger [1994] predict an ozone loss of less than 10% over any one point because of the shuttle's curved flight path. McPeters et al. [1991] could not find any evidence of ozone depletion from a study of the TOMS satellite data taken after eight shuttle launches. This evidence confirms that ozone losses from the plume chemistry should be small.

Danalin [1993] concluded that the homogeneous chemistry in the hot, highly concentrated plume was far more significant than any heterogeneous effect from alumina. However, this does not mean that alumina is not significant. Most of the globe does not meet the conditions necessary to form PSC's whereas alumina can exist at any latitude and temperature. If these particles showed an ability to help activate chlorine for months after their injection to the stratosphere, their contribution to ozone depletion could be significant even if their efficiency was much smaller than polar stratospheric clouds. The most recent available model (Jackman et al., 1996, in press) of the space shuttle's effect on ozone chemistry makes mention of this, but does not include it in their calculations.

In heterogeneous chemistry, γ defines the probability that a reaction between two molecules, A and B, will occur if molecule A collides with a surface which is covered to some extent by molecule B. To determine the activity of alumina, γ must be measured. Knowledge of the interaction of HCl and $ClONO_2$ and H_2O with the surface of alumina is important to give clues to a mechanism.

Cofer and Pellett [1978] investigated the interaction of H_2O and HCl on alumina surfaces and showed that physical and chemical adsorption takes place. However, the partial pressures of HCl and H_2O used were orders of magnitude above stratospheric conditions and their experiments were not conducted at low temperatures. This is because they were trying to mimic conditions within the rockect plume. There is a need for further study on adsorption properties as well as the value of γ.

I.2 References for Chapter I

Abbatt, J.P.D., Molina, M.J., *Annual Review of Energy & Environment*, **18**, 1, 1993.

Aftergood, S., Comment on "The Space Shuttle's Impact on the Stratosphere" by Michael J. Prather et al., *J. Geophys. Res.*, **96**, 17377, 1991

Brasseur, G., Granier, C., Mount Pinatubo aerosols, chlorofluorocarbons, and ozone depletion, *Science*, **257**, 1239-1242, 1992

Chapman, S., A theory of upper-atmosphere ozone. Mem. Roy. Meterol. Soc. **3**, 103, 1930.

Chipperfield, M.P., and J.A. Pyle, Two-dimensional modeling of the Antarctic lower stratosphere, *Geophys. Res. Lett*, **15**, 875-878, 1988.

Coffer, W.R., Pellet, G.L., Adsorption and Chemical Reaction of Gaseous Mixtures of Hydrogen Chloride and Water on Aluminum Oxide and Applications to Solid-Propelant Rocket Exhuast Clouds, NASA Technical Paper 1105, 1978.

Considine, D.B., Douglass, A.R., Jackman, C.H., Effects of a polar stratospheric cloud parameterization on ozone depletion due to stratospheric aircraft in a two-dimensional model, *J. Geophys. Res.*, **99**, 18779-18894, 1994.

Crutzen, P.J., *J. Geophys. Res.*, **30**, 7311, 1971.

Danilin, M.Y., Local stratospheric effects of solid-fueled rocket emissions, *Ann. Geopysicae*, **11**, 828-836, 1993.

Hanson, D.R., Ravishankara, A.R., Solomon, S., Heterogeneous reactions in sulfuric acid aerosols: A framework for model calculations, *J. Geophys. Res.*, **99**, 3615-3629, 1994.

Hoshizaki, H., Aircraft wake microscale phenomena, Chap. 2, pp 60-73, in *The Stratosphere Perturbed by Propulsion Effluents, CIAP Monogr. 3*, Climatic Impact Assessment Program, U.S. Dept. of Transportation, Washington, D.C., Sept. 1975.

Jackman, C.H., Considine, D.B., Fleming, E.L., The Space Shuttle's Impact on the Stratosphere: An Update, *J. Geophys. Res.*, In press, 1996.

Karol, I.L., Ozolin, Y.E., Rozanov, E.V., Effects of space rocket launches on ozone, *Ann. Geophysicae*, **10**, 810-814, 1992.

Kruger, B.C., Ozone depletion in the plume of a solid-fueled rocket, *Ann. Geophysicae*, **12**, 409-416, 1994.

McPeters, R., Prather, M., Doiron, S., Reply, *J. Geophys. Res.*, **96**, 17379-17381, 1991.

Molina, M.J., Rowland, F.S., Stratospheric Sink for Chlorofluoromethanes. Chlorine atom-Catalyzed Destruction of Ozone, *Nature,* **249**, 810 (1974)

Molina, M.J., Tso, T.L., Molina, L.T., Wang, F.C.Y., Antarctic Stratospheric Chemistry of Chlorine Nitrate, Hydrogen Chloride, and Ice: Release of Active Chlorine, *Science*, **238**, 1253-1258, 1987.

Potter, A.E., Environmental effects of the Space Shuttle, *J. Environ. Sci.*, **21**, 15-21, 1978

Prather, M.J., Garcia, M. M., Douglass, A.R., Jackman, C.H., Ko, M.K.W., Sze, N.D., The Space Shuttle's impact on the stratosphere, *J. Geophys. Res.*, **95**, 18583-18590, 1990.

Pyle, J.A., Jones, A.E., An investigation of the impact of the Ariane-5 launches on stratospheric ozone, prepared for the European Space Agency, 1991.

Rodriguez, J.M., Ko, M.K.W., Sze, N.D., The role of heterogeneous conversion of N_2O_5 on sulphate aerosols in global ozone losses, *Nature*, **352**, 134-137, 1991

Shen, T.L., Wooldrige, P.J., Molina, M.J., Stratospheric Pollution and Ozone Depletion, *Composition, Chemistry and climate of the Atmosphere*, edited by H.B. Sign, Van Nostrand Reinhold, 1995.

Solomon, S., The mystery of the Antarctic ozone "hole", *Rev. Geophys.*, **26**, 131-148, 1988.

WMO, Scientific Assessment of Ozone Depletion: 1994, World Meteorological Organization Global Ozone Reasearch and Monitoring Project- Report No. 37, 1995

Chapter II: Reactivity of $ClONO_2$ + HCl on α-Alumina Surfaces

II.1 Introduction

This chapter describes the experiments which attempt to quantify the rate of the heterogeneous catalytic chlorine activation reaction $ClONO_2 + HCl \rightarrow Cl_2 + HNO_3$ on the α-alumina surface. This reaction's rate or efficiency, γ, is the probability that a collision of a $ClONO_2$ molecule from the gas phase onto the α-alumina surface with adsorbed HCl will result in a chemical reaction to form the products. The value of γ for this reaction has not been investigated previously by other research groups.

Experiments were conducted under typical stratospheric conditions of temperature and reactant partial pressures. The experiments were performed in a low pressure - fast flow reactor, operated at steady state, fitted with a movable injector, and coupled to a quadrupole mass spectrometer. This technique is similar to that used previously by our group to measure the reactions probabilities such as $ClONO_2$ + HCl and HOCl + HCl on ice, NAT, and SAT (Abbatt and Molina, 1992; Abbatt and Molina, 1992b; Zhang et al., 1994). Control measurements were also performed on the unprotected glass flowtube, the flowtube with a Teflon sleeve, glass beads, and glass beads coated with 60% by weight sulfuric acid. The experimental procedure consisted of measuring the $ClONO_2$ reactant decay and product Cl_2 appearance at steady state in the presence of excess HCl vapor as a function of injector position, and from these we determined the respective pseudo first order rate coefficients, and then the γ's.

II.2 Experimental Details

II.2.1 Forms of Alumina Used

The α-alumina employed in our experiments were obtained in two physical forms—particles and a cylindrical tube. The particles were obtained from Aldrich Chemical Company and consisted of < 3 mm diameter sintered pieces (with irregular shapes). The alumina tube had an ID of 1.2 cm and was smooth and polished.

II.2.2 The Flowtube

All kinetic experiments were conducted in a flowtube as described below:

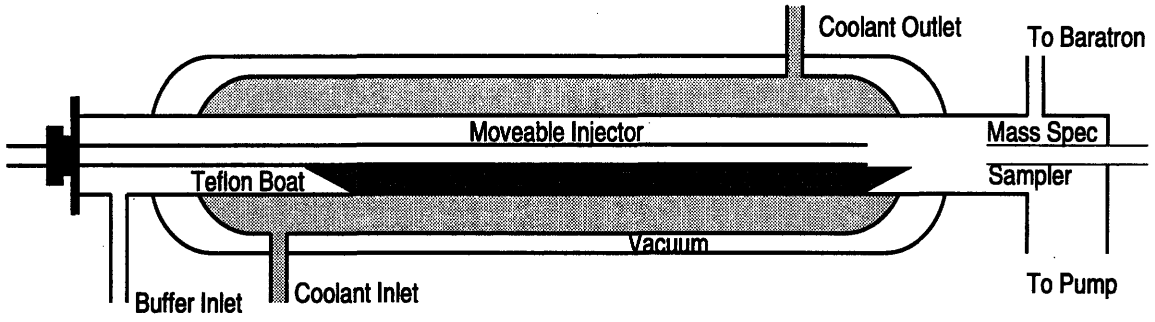


Figure II.1. Fast flow reactor. The main center tube is jacked by a second one through which coolant is recirculated. Buffer gas and first reactant enters near the rear while the second reactant enters through the injector. Solid sample is contained in hollowed out Teflon boat on the bottom. Mass spectrometer samples output flow.

Figure II.1 shows alumina pieces placed within the reactor flowtube inside of a Teflon boat. This Teflon boat also held the plain glass beads or glass beads coated with 60% by weight sulfuric acid. To use the alumina cylindrical tube, rather than the Teflon boat, the outside of the cylindrical tube was wrapped with thin Teflon sheet to form a snug fit with the flowtube's inner diameter. This ensured that all gas flow was down the center of the alumina tube.

II.2.3 The Extended Experimental Setup

Figure II.2 describes all the equipment attached to the flowtube. Trace gases (HCl, H_2O, $ClONO_2$) could be introduced from several sources. HCl was diluted with Helium inside of a glass bulb. The total pressure in the bulb was higher than that in the flowtube itself, so a needle valve was all that was needed to control the flow rate. A mass flow meter measured the flow rate and reported it to the computer's data acquisition board as a voltage signal. The HCl mixture was mixed with the main buffer flow before entering the flow tube. The $ClONO_2$ was similarly mixed, stored, flow controlled and measured like the HCl. All gas flow rates were measured by mass flow meters. Water vapor could be introduced by bubbling helium through water in an ice bath (to obtain a known vapor pressure) and the total pressure above the liquid was controlled by a needle valve and measured by a MKS Baratron. A quadrupole mass spectrometer sampled the flow from the reactor tube and the bulk flow was removed by a large rotary pump.

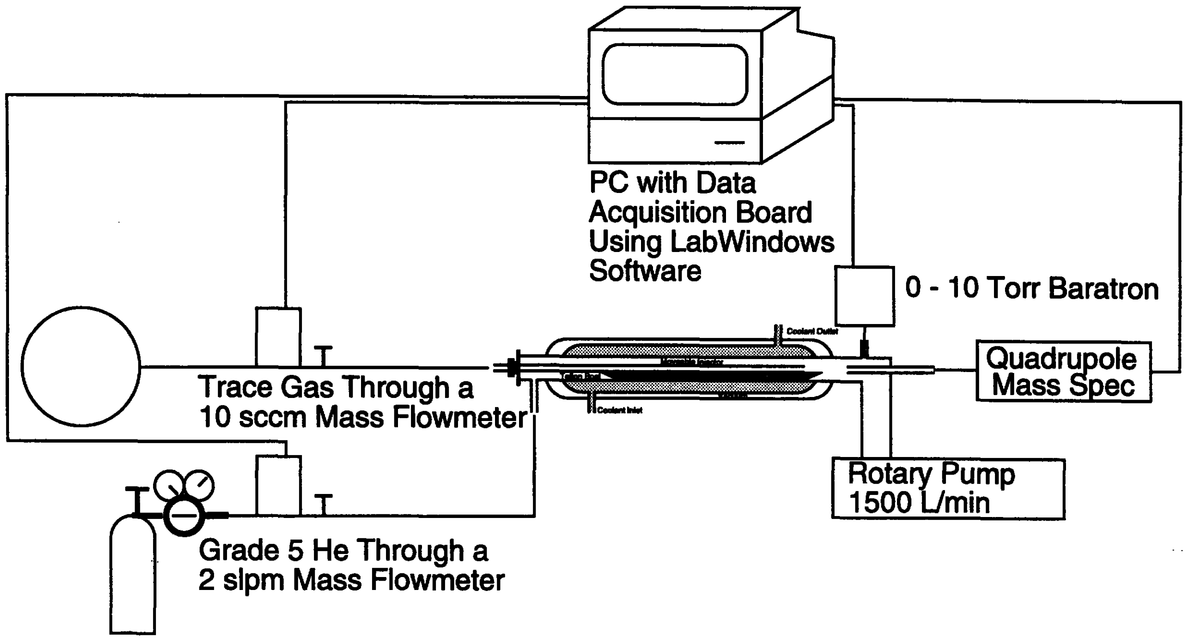


Figure. II.2. Schematic of complete setup. Computer monitors gas inputs through mass flowmeters and controls the quadupole mass spectrometer. Partial pressures of all species are recorded and data are presented and stored in useful form.

II.2.4 Mass Spectrometer Construction

The differentially pumped mass spectrometer system is illustrated in Figure II.3. The flowtube enters the gap before the first chamber. A 1200 liter/min vacuum pump line perpendicular to the flowtube removes the majority of the flow. The gas enters the first vacuum chamber by a pin hole of 0.1 mm in diameter. The first chamber is evacuated by a Varian diffusion pump capable of pumping enough to maintain pressures of 1.0 x 10^{-3} torr or less when the flowtube pressure is one torr. The expanding gas stream hits a skimmer cone at the entrance to the second chamber. The second chamber is pumped by a turbo molecular pump which keeps pressures below 1.0 x 10^{-6} torr. Thus, the narrow beam of gas that passes into the second chamber forms a molecular beam. In this chamber, the gas is ionized by electron impact and is filtered by the quadrupole mass spectrometer. To reduce any background signal, a chopper chops the molecular beam and the mass spectrometer signal is demodulated with a lock-in amplifier. This reduces the background signal by at least a factor of ten.

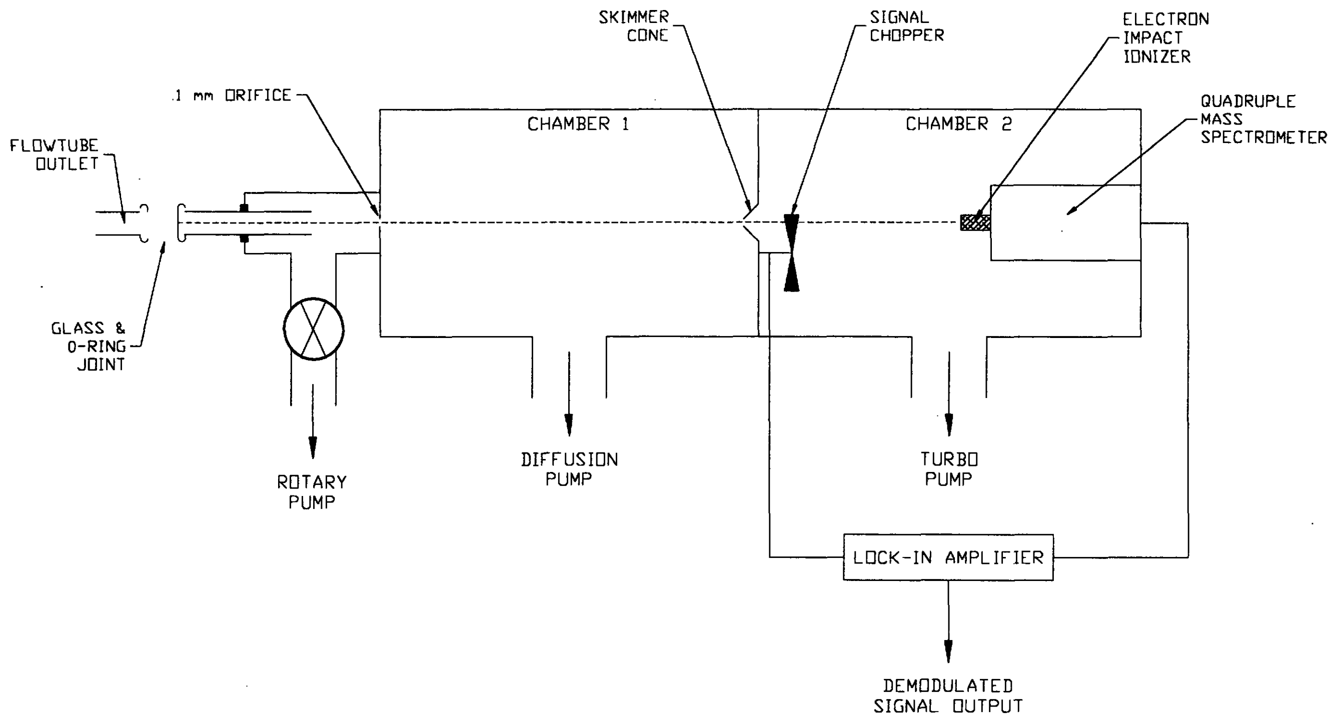


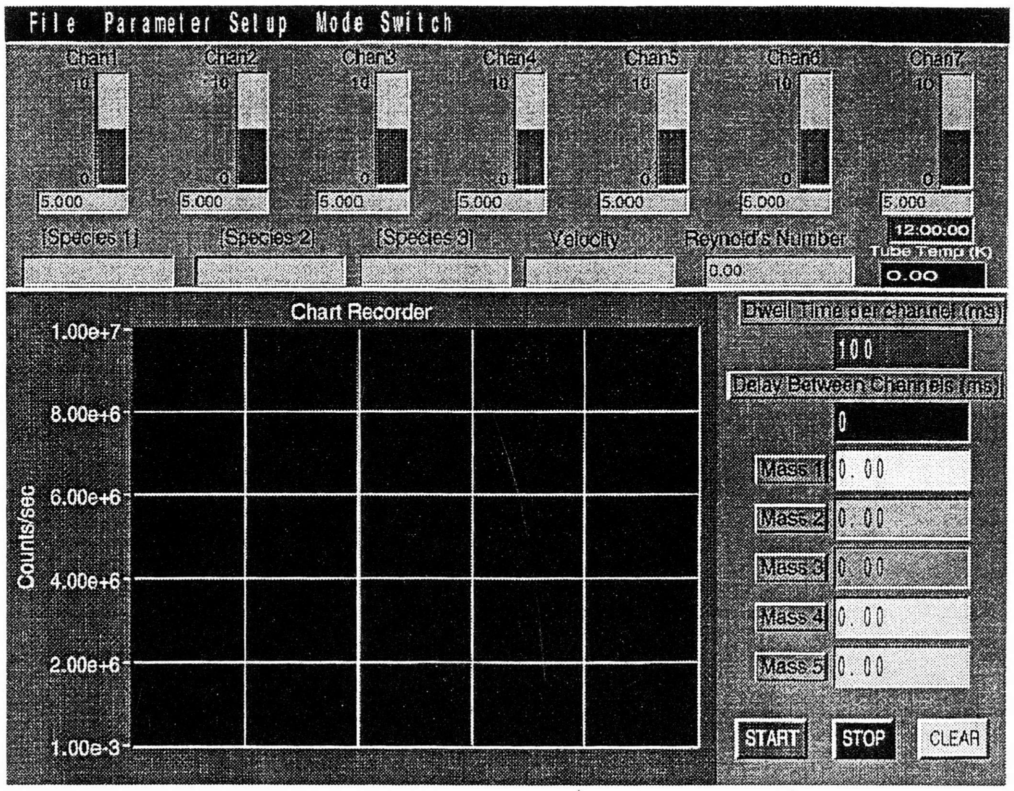
Figure II.3. Schematic of the electron impact mass quadrupole spectrometer setup. It contains two differentially pumped chambers. Gas from the flowtube is sampled by a 0.1 mm orifice to the first chamber which is pumped by a large diffusion pump. The sample gas forms a molecular beam as is enters into the second chamber through the skimmer cone. A chopper in line with the beam, coupled with a lock-in amplifier on the spectrometer's output signal helps to eliminate background signal.

II.2.5 Computer Control, Data Acquisition and Processing

The experiment was closely controlled and monitored by an Intel 286 based personal computer complete with a data acquisition card and software written specefically for these experiments. The data acquisition card was a National Instruments LabPC. The card was capable of monitoring and digitizing eight analog inputs and could control the mass spectrometer by means of one of two digital to analog outputs. Each mass flow meter was connected to the computer as well as all the baratron pressure gauges and the mass spectrometer signal. The mass flow meters gave a voltage signal 0-5 V DC linearly proportional to their maximum rated flow speed. The baratrons and the mass spectrometer gave signals 0-10 V DC linearly proportional to their maximum readings.

The software used was built using subroutines from National Instrument's LabWindows for DOS 2.3. This gave the software a graphical interface much like Windows or the Macintosh. The code is included in Appendix I at the end of this thesis. The software was designed with several duties in mind. It provided an easy way to input data related to the experimental setup without altering the source code. An example would be changing the $ClONO_2$ source from a pressurized source of known mixing ratio to a gas flowing through a bubbler of known vapor pressure and a measured total pressure. Futhermore, it automatically calculated and displayed vital parameters such a flow velocity and species partial pressures in the flow tube based on previously entered data.

The software allowed the computer to function as several different instruments simultaneously while always displaying the important information mentioned above. The computer could function as a chart recorder to monitor several masses over time. It could take a mass spectrum and allow the user to zoom in on a portion or pick out the masses and intensities of any peak. It also had a mode that measures the signal from a single mass versus an outside variable such as injector position. The software could easily switch between these modes. For instance, if the operator were measuring signal versus injector position and suspected that the signal wasn't stable, he could evaluate its stability by switching to chart recorder mode. He could search for contamination peaks by using the mass spectrometer mode. When satisfied, the operator could switch back to recording signal versus injector position mode.



Chan1 5.000 Chan2 5.000 Chan3 5.000 Chan4 5.000 Chan5 5.000 Chan6 5.000 Chan7 5.000

Reynold's Number 0.00

12:00:00

Tube Temp (K) 0.00

Dwell Time per channel (ms) 100

Delay Between Channels (ms) 0

Mass 1 0.00

Mass 2 0.00

Mass 3 0.00

Mass 4 0.00

Mass 5 0.00

Figure II.4. User interface of data acquisition software using LabWindows 2.3. In this case, the top portion reports gas flows, pressures, concentrations, and other experimental data in real time. The bottom panel contains controls to operate the chart recorder.

Finally the software stores the data in tab delineated columns that can easily read into other programs such as spreadsheets. These can, in turn, be used for further analysis or graphing.

II.2.6 Experimental Procedure and Conditions

Experiments were conducted in dry helium gas or helium humidified with 3-5 x 10^{-4} torr of water at total pressure near 1 torr. The partial pressures of $ClONO_2$ and HCl were in the range of 10^{-7} to 10^{-6} torr, which corresponded to typical lower stratospheric values. The measured first order rate coefficients, determined by non-linear least squares fitting routine, were corrected for the effects of radial diffusion (Brown, 1978).

The experiment was conducted by changing the injector position and measuring the reactant $ClONO_2$ signal decay and Cl_2 product signal rise. The mass spectrometer measured $ClONO_2$ by observing the NO_2^+ peak and the Cl_2 by observing the Cl_2^+ peak. The temperatures used ranged from -20 to -75 ^oC.

II.3 Results and Discussion

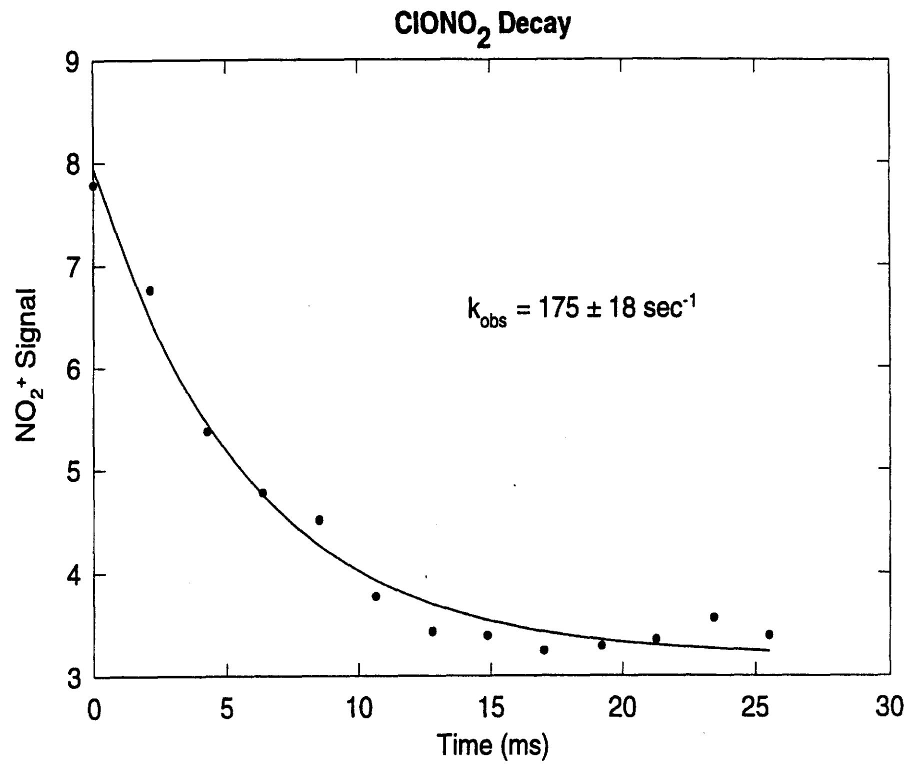
Typical measured reactant decays and product buildups are shown in Figures II.5-8; the injector position has been converted to time, taking into account the gas flow velocity. On average, the partial pressures used for each figure are [$ClONO_2$] = 7.6 x 10^{-7} Torr, [HCl] = 8 x 10^{-6} Torr, [H2O] = 4.7 x 10^{-4} Torr. The temperature for each experiment was -60°C. Figure II.5 is from the reaction on the glass flowtube. Figure II.6 is with a Teflon sheeth covering the flowtube and with glass beads in a Teflon boat. Figure II.7 is is with a Teflon sheeth covering the flowtube and alumina pieces in the Teflon boat. Figure II.8 is from the polished alumina tube fitting tightly in the flowtube. The value k_{obs} is the result from the non-linear fit to:

$$NO_2^+ \text{ Signal} = Ae^{-k_{obs}t} + C$$

or

$$Cl_2^+ \text{ Signal} = A(1 - e^{-k_{obs}t}) + C$$

From these derived values of k_{obs}, we can calculate the value of γ [Brown, 1978]. Note that glass has a similar activity as alumina. A Teflon sleeve on the flowtube resulted in no detectable reaction.



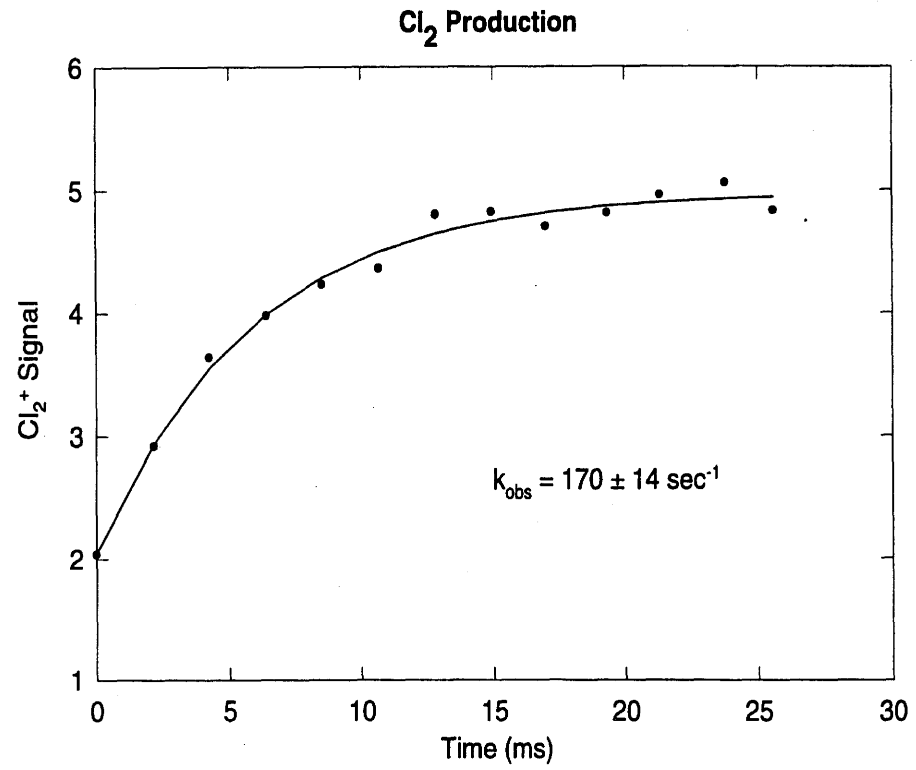
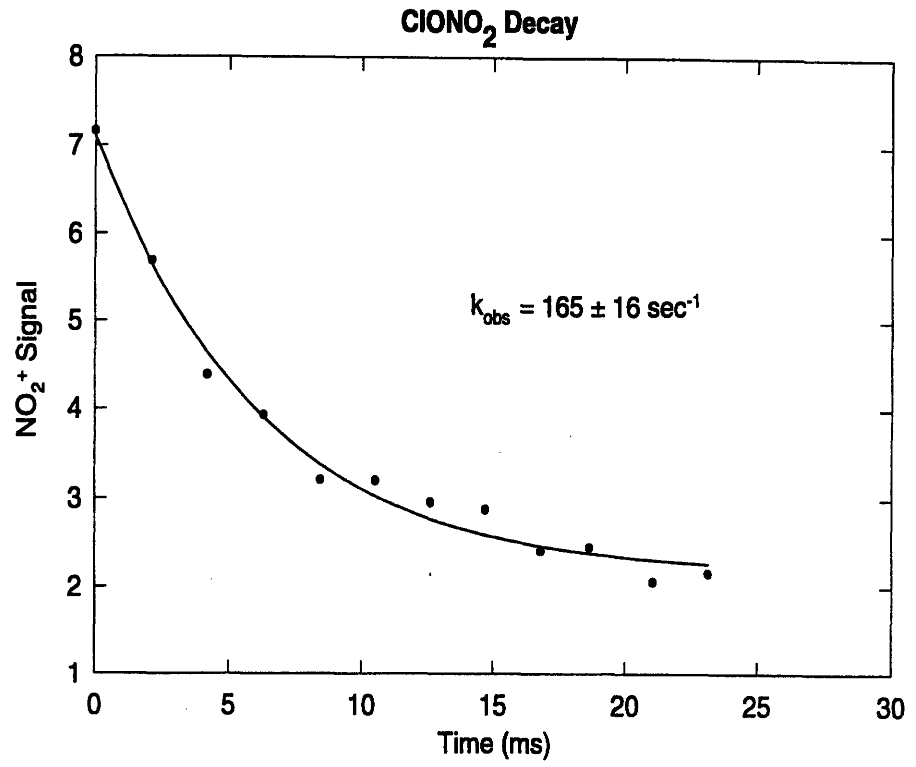


Figure II.5. Observed derived first order rate constants for NO_2^+ and Cl_2^+ signal for $ClONO_2$ + HCl on the glass flowtube at -60°C. [ClONO2] = 7.54x10^{-7} Torr, [HCl]= 7.46 x 10^{-6} Torr, [H_2O] = 4.81 x 10^{-4} Torr.



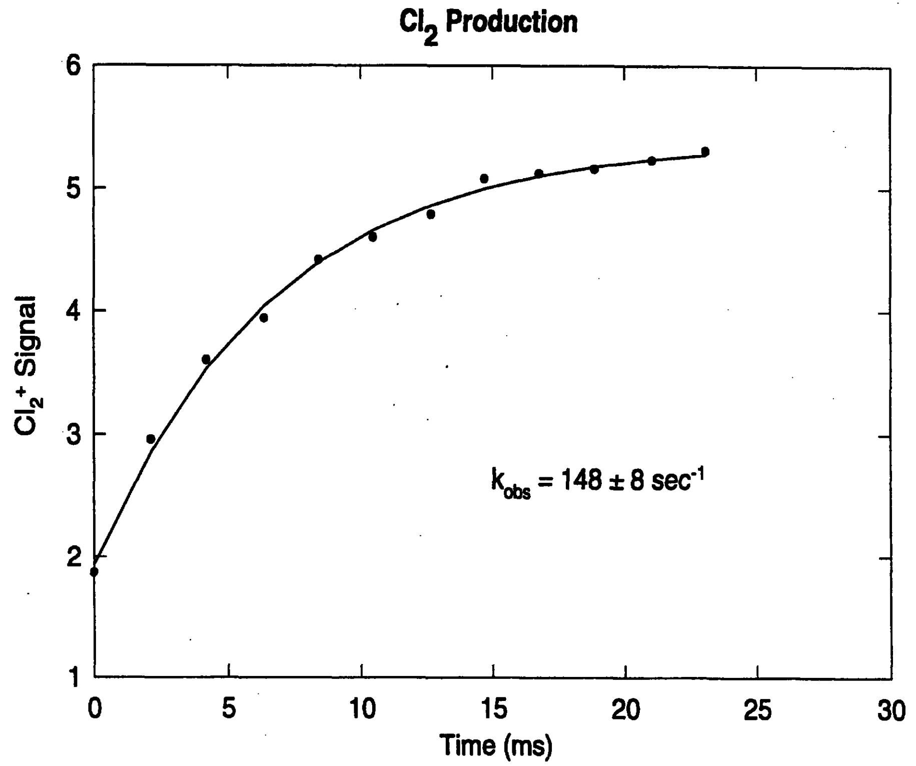


Figure II.6. Observed derived first order rate constants for NO_2^+ and Cl_2^+ signal for ClONO2 + HCl on Glass beads in a Teflon boat at -60°C. $[ClONO_2]=7.81\times10^{-7}$ Torr, $[HCl]=7.71\times10^{-6}$ Torr, $[H_2O]=4.57\times10^{-4}$ Torr.

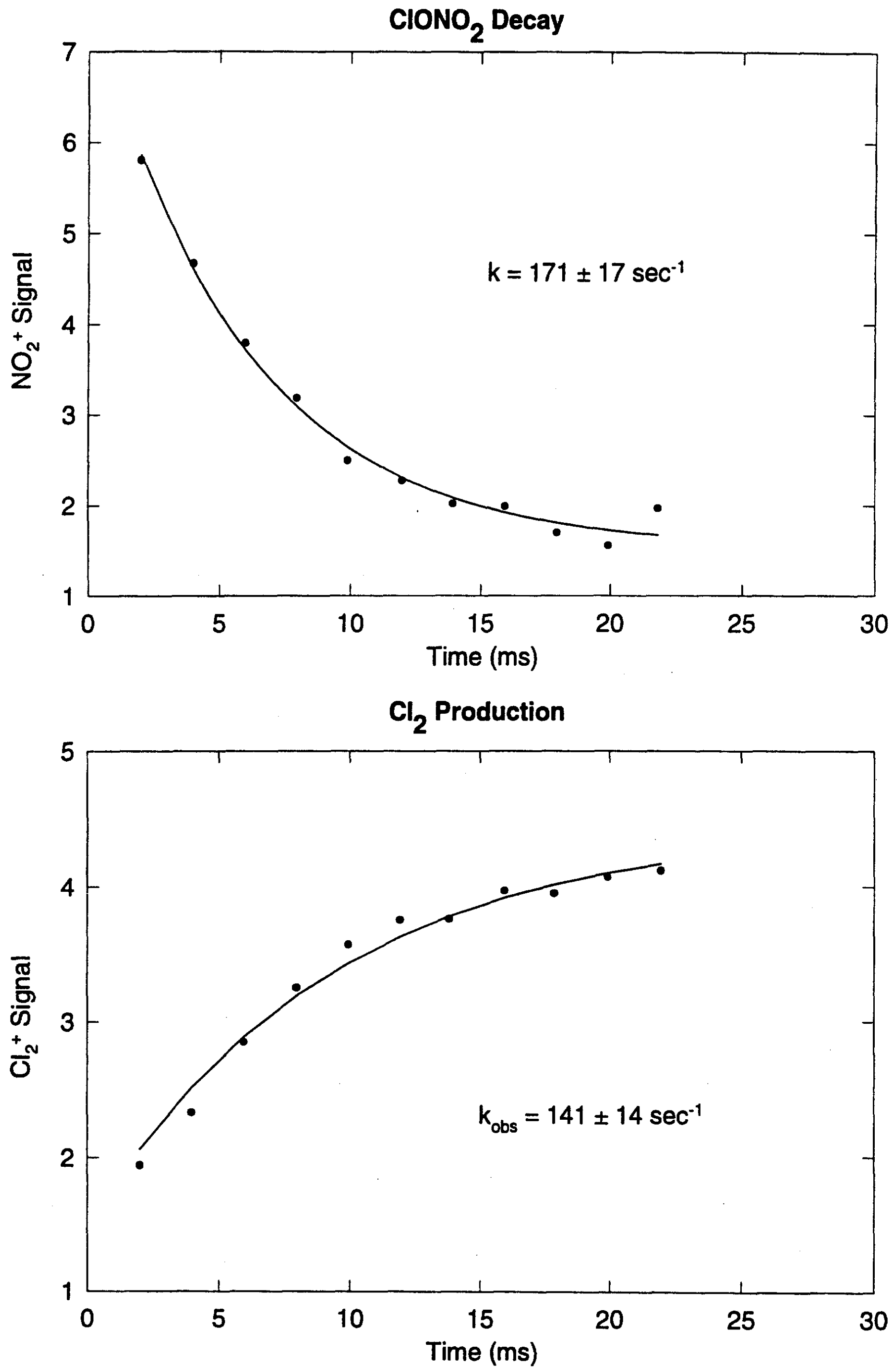


Figure II.7. Observed derived first order rate constants for NO_2^+ and Cl_2^+ signal for $ClONO_2$ + HCl on α-alumina pieces at -60°C. $[ClONO_2]$ = 7.56x10^{-7} Torr, [HCl] = 8.33x10^{-6} Torr, $[H_2O]$ = 4.41x10^{-4} Torr.

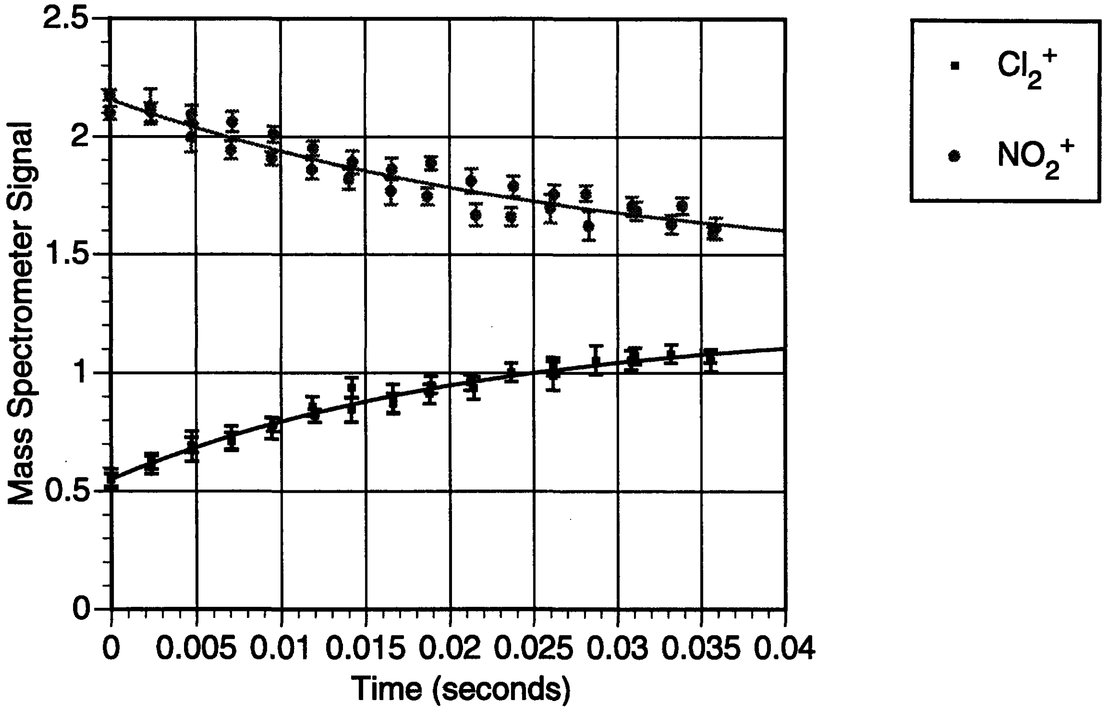


Figure II.8. Kinetic results from alumina tube. The pseudo-first order rate constant from the $Cl_2{}^+$ peak is 46.7 s^{-1} while the one from the NO_2^+ peak is only 35.6 s^{-1}. This is because both reactant $ClONO_2$ and product HNO_3 yield NO_2^+ under electron impact and make this curve less reliable.

Due to the challenges of modeling the flow dynamics with alumina pieces sitting in the bottom of the tube, the reaction rate was measured relative to the value for the cylindrical glass flow tube for which the flow dynamics are well understood. The Teflon sleeve was again inserted into the flow tube and glass beads of comparable size to the alumina were placed in the Teflon boat inside the flow tube. After the reaction probability was measured, the experiment was repeated with alumina particles. We calculated the reaction probability of the alumina to be the ratio between the alumina and the glass beads multiplied by the value for the glass cylinder. Table 1 yields the results.

These reaction probabilities varied very little over the range of -20 to -75 ^{o}C. The addition of water vapor at stratospheric partial pressures of 5 x 10^{-4} torr also made no appreciable change. Control experiments on ice films yielded the expected reaction probability of > 0.1 (Abbatt and Molina, 1992). Control experiments on a 60% solution by weight sulfuric acid coating the glass beads also showed complete deactivation.

Table 1. Average results from 10 experiments.

	Glass Tube	Glass Beads	α-Alumina
$<k_{obs}>$	177 ± 18 sec^{-1}	153 ± 10 sec^{-1}	138 ± 11 sec^{-1}
$<\gamma_{measured}>$	$.022 \pm .005$	$.017 \pm .002$	$.014 \pm .002$

$$\gamma_{\alpha\text{-alumina}} = \gamma_{glass} \bullet \frac{\gamma_{\alpha\text{-alumina, measured}}}{\gamma_{\text{glass beads, measured}}} = .019 \pm .005$$

The same experiments were repeated on a smooth α-alumina tube when one became available. Experiments were conducted in dry conditions below that of the stratosphere. Each consecutive run of the experiment resulted in a lower value of γ. This could be from depletion of water on the surface of the alumina or from poisoning the surface with nitric acid. Another set of experiments were conducted using stratospheric levels of water and nitric acid (5ppmv H_2O, 5ppbv HNO_3). Table 2 summarizes results from data taken by Roger Meads. The γ values did not decrease as under dry conditions. Therefore, availability of water on the surface is important and poisoning of the surface by nitric acid appears to not be significant.

Table 2. Results from kinetic experiments on smooth α-alumina tube.

Consecutive Run #	γ_{Dry}	$\gamma_{H_2O\,(5ppmv)\,+\,HNO_3(5ppbv)}$
1	0.025	0.023
2	0.015	0.019
3	0.010	0.022
4	0.007	0.017
5	0.007	0.016
Average Results	0.013 ± 0.008	0.019 ± 0.003

The glass flowtube, the glass beads, and the alumina pieces did not show great sensitivity to water, but the alumina tube did. This may be because the alumina tube had less than half the total surface area of all the other materials, but had the same flow rates of chemicals over the surface. Thus, the surface could be depleted of its adsorbed water much faster than the other surfaces. The dependence of γ on water is important evidence for our mechanism theory that hydrophilic surfaces with adsorbed water will catalyze this reaction.

Althought the values for γ for the alumina pieces and the alumina tube under stratospheric conditions agree well within experimental error, there are differences between the surfaces that should be discussed. There are reasons that could make the value of γ to be different between the alumina pieces and alumina tube. The alumina pieces were irregularly shaped from the mechanical crushing that produced them. The crushed alumina pieces could have highly activated lattice sites that are not present on the smooth alumina tube. This is noteworthy because the alumina from solid rocket motors could have many of these activtated sites.

The experiment to determine γ on alumina pieces was vulnerable to unknown flow dynamics and diffusion. The glass beads were smooth spheres. Thus, the irregularly shaped alumina could have a higher surface area than the same amount of glass beads in the Teflon boat. Therefore, when calculating the ratio between the value for glass and the value for alumina, a higher surface area than expected would result in a higher calculated value for γ on alumina. Also, it could take longer for the gas to flow through the irregularly shaped alumina pieces than the glass beads. The longer residence time would result in more reaction and thus a higher value for γ. However, these effects could be mitigated by failure of the gas to diffuse below the top layer of the alumina.

Because the values for γ for stratospheric conditions on alumina agree well between Tables 1 and 2, the effects discussed above are either small or cancel each other out. Chapter V dicusses future work using powdered alumina that would not be as vulemable to flow dynamic and surface area affects and would be a better test for the presence of highly activated sites.

It should be noted that these reaction probabilities were determined from the rate of product rise using a non-linear least squares fit. Both $ClONO_2$ and one of the products, HNO_3, yield NO_2^+ under electron impact ionization in the mass spectrometer with slightly different ionization efficiencies. Thus, even with complete reaction, the reactant signal levels off above zero. This made the reactant decay curve unreliable, so the product rise curve was used instead.

Figures II.9 and II.10 illustrate this phenomenon. In both graphs, the injector is pulled back the same length. In figure II.9, we see that the chlorine product appears instantaneously. However, in figure II.10, we see that the NO_2^+ peaks first dips strongly and then rises a bit. This is consistent with $ClONO_2$ being rapidly depleted by reacting on the surface to form HNO_3 followed by saturation of the surface with HNO_3 and then HNO_3 degasses.

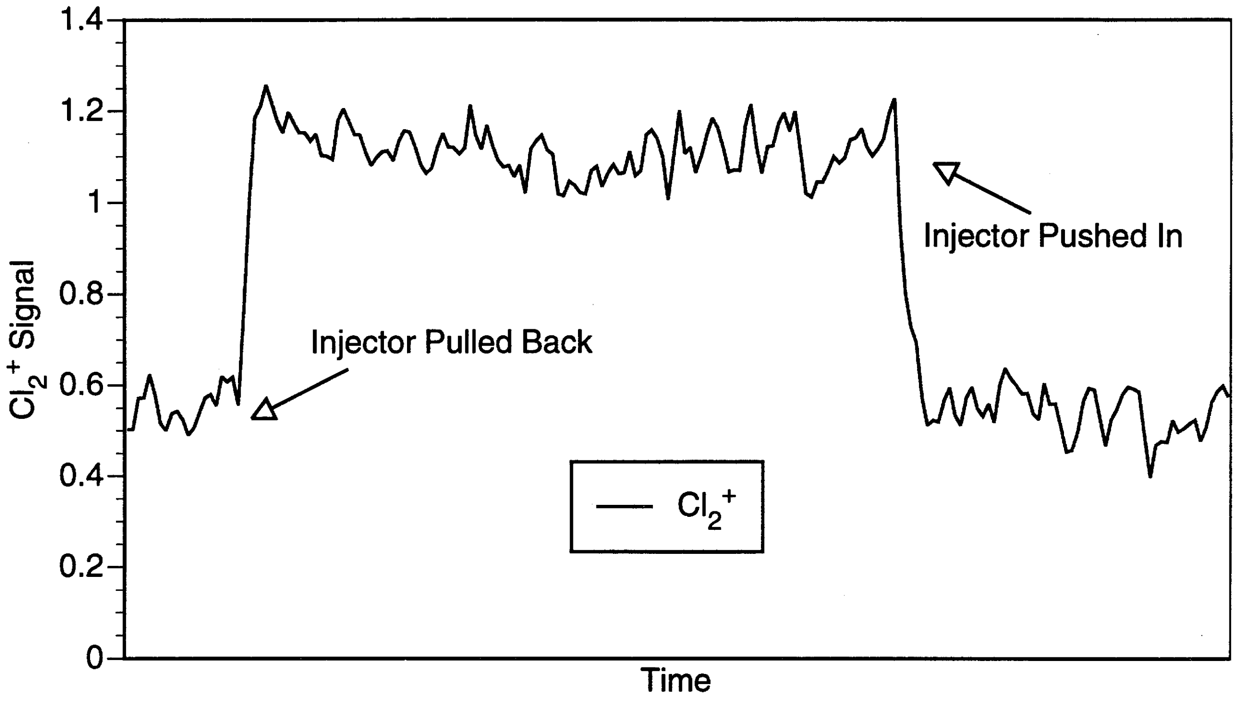


Figure II.9. Reaction between $ClONO_2$ and HCl on alumina seen over time. Chlorine production rises instantaneously to a new steady–state level as the injector is pulled back to increase reaction time.

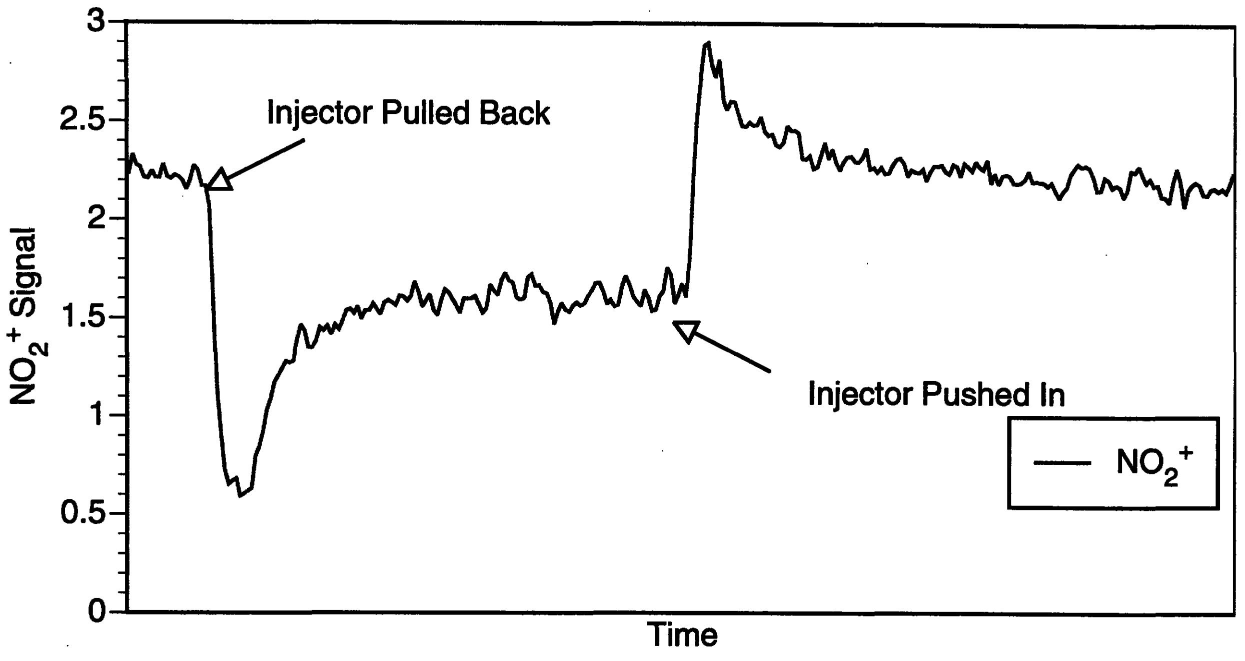


Figure II.10. Reaction between $ClONO_2$ and HCl on alumina seen over time. The more complex NO_2^+ signal indicates depletion of $ClONO_2$ and buildup of HNO_3.

II.4 References for Chapter II

Abbatt, J.P.D., Molina, M.J., *Geophys. Res. Lett.*, **19**, 461, 1978.
Abbatt, J.P.D., Molina, M.J., *J. Phys. Chem.*, **96**, 7674, 1992.
Brown, R.L., *J. Res. Nat. Bur. Stan.*, **83**, 1, 1978.
Zhang, R., Jayne, J.T., Molina, M.J., *J. Phys. Chem.*, **98**, 867, 1994.

Chapter III: Adsorption Experiments

III.1 Introduction

In order to better understand the mechanism of the heterogeneous reactions under study, the uptake and release on alumina of HCl, $ClONO_2$, and H_2O was studied. Chemical adsorption rather than physical adsorption would support the idea that this substrate does indeed influence the chemical pathway of reaction.

III.2 Experimental Details

III.2.1 Synthesis of $ClONO_2$: Reaction of FCl with $Pb(NO_3)_2$

The synthesis used was patterned after Schmeißer et al. (1980). A stainless steel vacuum line was assembled in a fume hood as pictured in Figure III.1. A known amount of lead nitrate was added to a stainless steel cylinder with a valve on the end. The valve end was attached to the vacuum line via a stainless steel bellows and the whole line was pumped down. Next, the lead nitrate line was closed off and the FCl source was opened and the line pumped down. After closing off the vacuum pump, the regulator to the FCl tank was briefly opened to passivate the line including the stainless steel beaker reservoirs. After closing the regulator, the line vacuum source was reopened and the line was pumped down again. This process was repeated. Then, with the vacuum source closed off, the stainless steel beaker reservoirs were filled with approximately one atmosphere of FCl and the FCl source was then closed. The valve to the lead nitrate cylinder was opened and the cylinder was placed in a dewar of liquid nitrogen to draw in all of the FCl. After the pressure in the line dropped to only a few torr, the valve was closed and the vacuum line pumped down. The lead nitrate cylinder was then place in a dewar in a dry ice/acetone bath for several hours. After reaction, the cylinder was briefly pumped down to remove any FCl or Cl_2 present. The $ClONO_2$ was then transferred over to a glass receptacle by vacuum distillation with the glass finger being kept at -70°C and the steel cylinder being kept at -30°C. The drawback of this process is that side products in the form of chlorine oxides are clearly visible. The oxides are only slightly more volatile than chlorine nitrate and must be removed by careful vacuum distillation. Some of the $ClONO_2$ used was synthesized by reaction of Cl_2O and N_2O_5 as reported before (Molina et al, 1977).

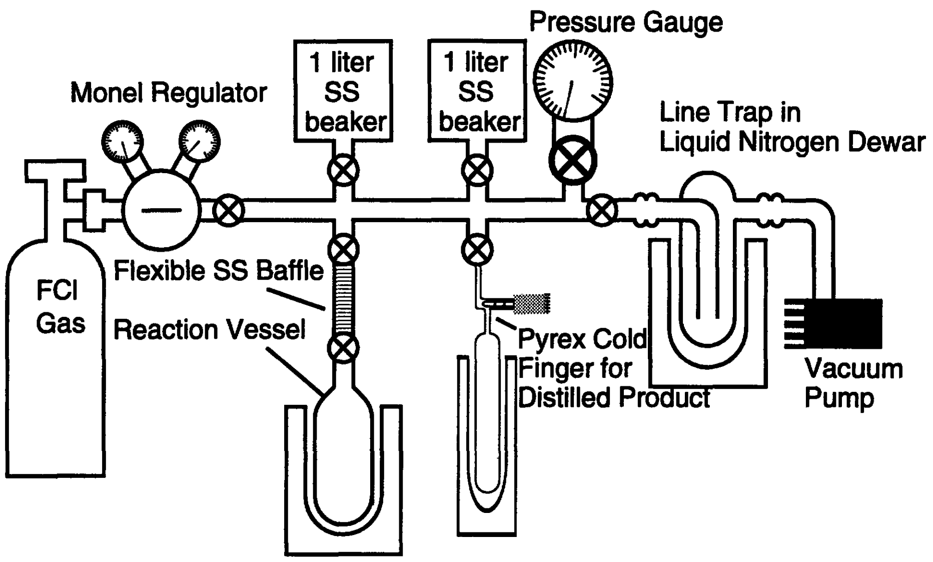


Figure III.1. Stainless steel vacuum line for synthesis of $ClONO_2$. FCl is condensed onto $Pb(NO_3)_2$ in the first cylinder and then warmed. Upon completion, product is vacuum distilled over into the cold finger.

III.2.2 Surface Area Determinations

The surface area of the α-alumina was measured by BET (Brunauer, Emmett, and Teller, 1938) using Krypton gas. Krypton gas filled a reservoir and vacuum line of known volume with a range of pressures measured by a high accuracy, low pressure MKS Baratron. The alumina sample was held in a glass container which was immersed in liquid nitrogen. The valve from the Krypton gas to the alumina was opened and the pressure drop upon equilibration was measured. The volume of gas adsorbed at each pressure was determined by comparing the drop in pressure to the drop in pressure expected with no adsorption and correcting for the volume of the alumina. The volume of the alumina itself was determined by weighing the sample and dividing by the density provided by the supplier of the alumina, Aldrich Co.

III.2.3 Flowtube Construction

The flowtube used was the same one as described in chapter II (Abbatt et al, 1992a, Abbatt et al., 1992b, Zhang, 1994). However, in this case, alumina or glass beads were placed along the entire length of the flowtube rather than just in a Teflon boat.

III.2.4 Mass Spectrometer Setup

A new experimental setup was built to do the adsorption experiments. Rather than a pin-hole, the gas was sampled by a 1/4" OD glass tube that was connected to a

1/16" ID SilcoSteel™ tube that is glass coated on the inside. The capillary tube was positioned to aim at the electron impact grid. A diffusion pump equipped with a cryrotrap eliminated most of the oil peak background.

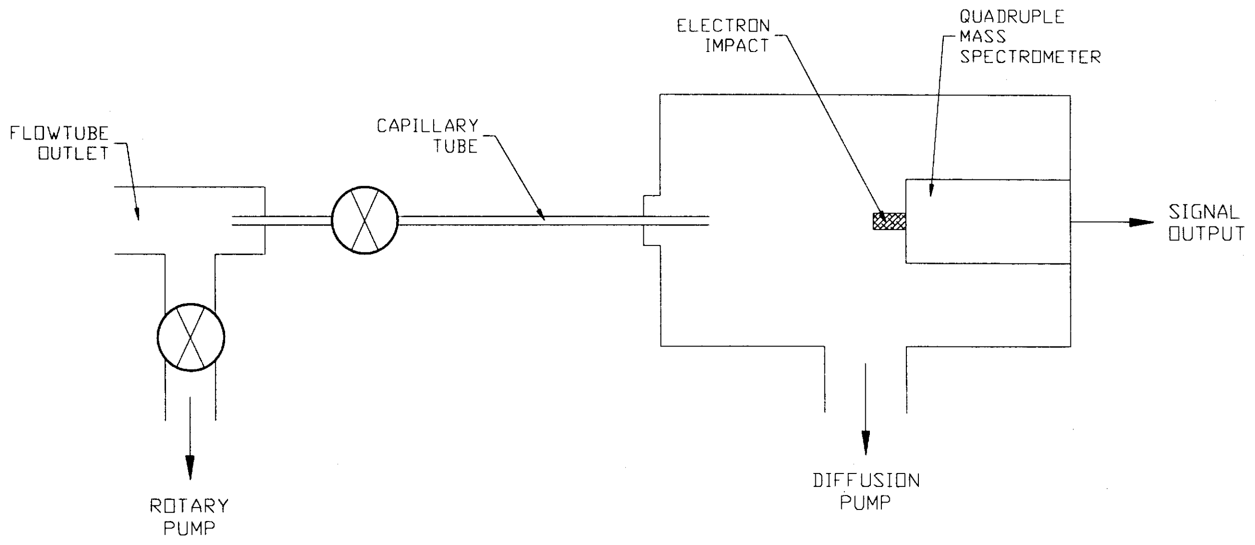


Figure III.2. Mass spectrometer system was a single chamber design. A small tube of 1/16" ID sampled the flowtube and delivered the gas into the chamber where the mass spectrometer was located.

III.2.5 Experimental Procedure

For adsorption studies, the buffer stream consists of pure He and the trace gas flows down the injector. As the injector is rapidly drawn back, the signal drops sharply as the gas adsorbs onto the surface of the solid. After some time, the signal returns to the its previous position as the surface equilibrates. If the injector is then pushed to its original downstream position, the mass spectrometer signal increases as the gas desorbs from the surface and then gradually decreases to the equilibrium position. By measuring the integrated difference between the mass spectrometer signal and its equilibrium position, we can determine the amount of gas adsorbed or desorbed.

HCl

The HCl source was a pressurized cylinder of HCl in grade 5 He purchased from Matheson Gas Co. Analysis of the mixture by Matheson Co. determined that the HCl mixing ratio was 0.075%. The regulator on the tank was constructed of monel alloy. The equivalent flow rate from the tank at STP was between 1 and 10 cc/min. Before entering the injector tube, this flow was diluted by at least a factor of ten. The total flow in the flowtube was the equivalent of 500 cc/min at standard temperature and pressure (STP). The pressure in the flowtube was maintained at approximately 1 torr. Thus, the absolute pressure of HCl in the flowtube for adsorption experiments was between 1.0×10^{-4} and 1.0×10^{-5} torr. A survey of the various possible peaks determined that mass 36 from the HCl^{35} parent ion was the strongest signal.

The first experiment was to measure the uptake of HCl on the glass flowtube itself. This was done for two reasons. First, it would determine the background adsorption level of interference that we would have to compensate for when doing our experiments with materials in the tube. Second, it would give us a value for the adsorption of HCl on glass which would be useful for later comparison with alumina.

At each temperature and pressure, the injector was pulled back and measurements conducted at a variety of injector distances. Because the inside of the flowtube is a smooth cylinder, the adsorption of HCl should be linear with the distance the injector is pulled back. Thus, a linear regression of the adsorption versus injector distance was used to calculate the extent of adsorption per unit area, i.e., the surface coverage.

This result was compared with that obtained by placing glass beads in the bottom of the flowtube. The glass beads were smooth and 3 mm in diameter, the geometric area of all the beads was several times greater than that of the flowtube. Because it was difficult to spread the beads out evenly, the injector was pulled back to the same position every time to expose the majority of the beads. The amount adsorbed was divided by the area of the beads plus the area of the flowtube to give the adsorption per unit area.

The surface coverage of HCl on alumina was determined in the same fashion as it was done for the glass beads. The alumina pieces were similarly placed on the bottom of the flowtube. Again, the injector was pulled back the same distance every time to expose most of the alumina.

H_2O

H_2O was introduced into the system by using a bubbler. A grade 5 helium flow enters the bubbler inlet tube at a flow rate equivalent to 1 to 10 cc/min at STP and passes through a small glass fritted filter to form small bubbles in the water. These bubbles rapidly acquire the equilibrium vapor pressure of water. This temperature was held at 0° C by placing the bubbler in a dewar filled with ice water. A needle valve is located at the exit of the bubbler at a T joint. A baratron pressure gauge measures the total gas pressure over the water, which was controlled by adjusting the needle valve. The mixing ratio of the water is calculated by dividing the vapor pressure of the water by the total pressure of the gas. From this point, the humidified helium was diluted and handled as the HCl gas in the previous example. The flows were adjusted until the water partial pressure in the flowtube was 5×10^{-4} torr, which is same as the stratosphere at about 16 km altitude. Then, the adsorption experiments on the glass beads and alumina pieces were conducted in the same fashion as the HCl experiments.

$ClONO_2$

$ClONO_2$ was introduced to the system in the same bubbler device as H_2O. The $ClONO_2$ was placed in a thoroughly dried bubbler. Because the vapor pressure of $ClONO_2$ is much higher than water, the bubbler was kept in a dewar filled with a dry ice—acetone slush bath. The partial pressure of $ClONO_2$ in the flowtube ranged from 1.0×10^{-6} to 1.0×10^{-5} torr.

$HCl + H_2O$, $ClONO_2 + H_2O$

To observe the possible impact of H_2O on the adsorption of HCl or $ClONO_2$, all the HCl and $ClONO_2$ experiments were repeated with a background water vapor pressure of 5.0×10^{-4} torr, corresponding to typical lower stratospheric values.

III.2.6 Procedure for Determining Surface Coverage

To measure the uptake of HCl, $ClONO_2$, or H_2O on alumina or glass, the instrument was placed into mass spectrometer mode and the peak of interest was monitored. It is important to verify the identity of the signal peak for two reasons. First, the parent ion peak may not be the largest or it may interfere with another signal such as a pump oil peak. Second, instrumental drift and calibration difference may result, for example, in H_2O appearing as mass 17.5 rather than 18.

Next, the instrument was placed into chart recorder mode. If more than one peak was to be monitored, care had to be taken to have enough of a delay between samples to prevent an averaging of the two signals.

To start, the injector flowing the trace gas was placed downstream of the flowtube. Then, the injector would be pulled upstream to expose the substrate material to the trace gas. The resulting adsorption would lead to a temporary dip in the signal from the mass spectrometer until the surface came into equilibrium with the gas phase and the signal would return to its previous level. Next, the injector would be pushed downstream again to its previous position. This results in a temporary increase in the mass spectrometer signal as gas desorbed from the surface. The volume of gas adsorbed or desorbed was calculated by integrating the difference between the signal and its equilibrium position.

III.3 Results and Discussion

Typical adsorption data of HCl, $ClONO_2$, and H_2O are shown in Figures III.3-6. Coffer and Pellet [1978] studied adsorption on high surface area powders, reporting their measurements in mg/m^2. For comparison, we also report our results in mg/m^2. The strong hysteresis noted confirms their observation of chemical adsorption, followed by physical adsorption. However, our surface coverages are approximately 1000 times lower. This is expected as a consequence of the difference in partial pressure of HCl used. Their experiments used more than 1000 times the partial pressure as ours. Furthermore, the samples might not have had enough time to adsorb. However, Figures III.3-6 show quick return to baseline signal. Any further adsorption would have to be very slow and beyond the sensitivity and stability of our instrument. Third, their manner of preparation to form high surface area powders may also be responsible. It should be noted that a study of actual shuttle alumina (Coffer et al., 1984) yielded adsorptions of 1/3 of what these authors measured in their 1978 laboratory experiments.

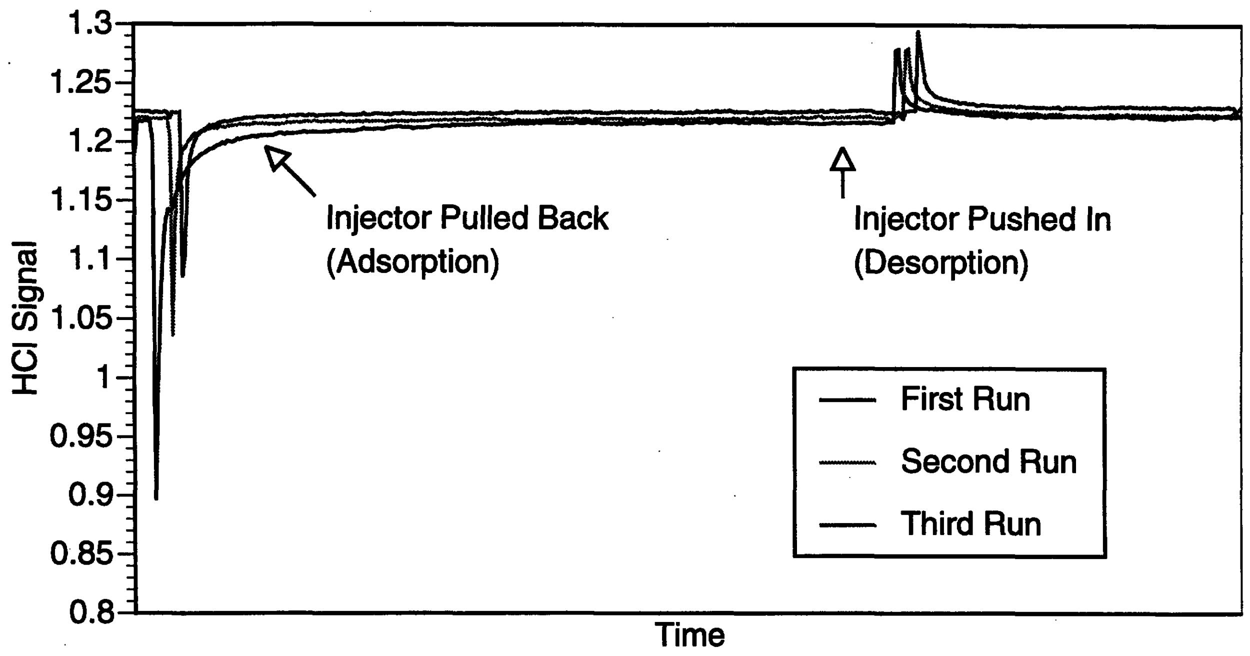


Figure III.3. HCl adsorption on α-alumina. Consecutive runs show strong hysteresis.

The laboratory experiments of Coffer [1978] were meant to simulate the conditions present in the rocket plume. The rocket plume has many times the water vapor and HCl partial pressures than the background levels. Perhaps our data is more applicable to alumina that forms in those background levels, such as alumina from meteorites or falling satellite debris.

Yet, it should be noteworthy that similar adsorption patterns are observable even at low partial pressures. In addition, our water adsorption are also completely reversible, even at lower partial pressures. Glass did not show hysteresis upon adsorption of HCl. This agrees with Coffer's [1978] conjecture that aluminum oxy-chloride salts are the mechanism for chemical adsorption. These types of salts are unlikely to occur with silica. Slight hysteresis occurred with $ClONO_2$ on glass, but this may due to the hydrolysis of $ClONO_2$ with water on the surface of glass. Our mass spectrometer is not sensitive enough to detect HOCl at these levels. If glass and alumina do have a sufficient coating of water on their surfaces, this would also help explain the reversible adsorption of water that we observed.

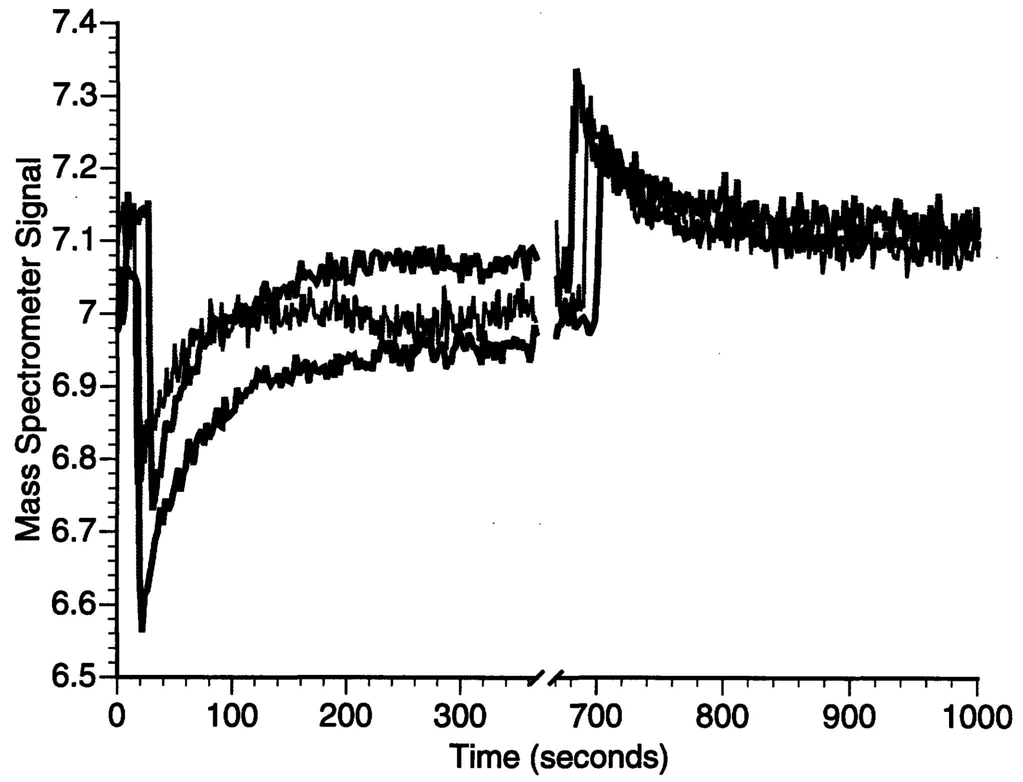


Figure III.4. Uptake of $ClONO_2$ on α-alumina for three consecutive tests. First run adsorbed twice as much as the following ones. Desorption was always less than adsorption.

The observation of chemical, rather than only physical, adsorption of HCl occurs on alumina at stratospheric conditions is consistent with alumina having surface sites with a wide range of activities. Uptake measurements as a function of the partial pressure of the gas yield adsorption isotherms which provide fundamental information on the nature of the adsorption process (Adamson, 1990). If HCl formed only very strong hydrogen bonding with the surface (i.e. 9 kcal/mole), it would result in a coverage of less than 10^{-3} of a monolayer (Molina, 1994). The surface coverage observed on alumina is greater than 10^{-3} but less than a full monolayer. Thus, the forming of a strong chemical bonding (18 kcal/mole), such as a Al-Cl bond, is not likely except at highly activated lattice sights. The nature of the bond between HCl and the alumina must be in the range of weak chemical bonding corresponding to a range of site activites. This is important because if the Cl were bonded too strongly with Al, it would not be available for further reaction.

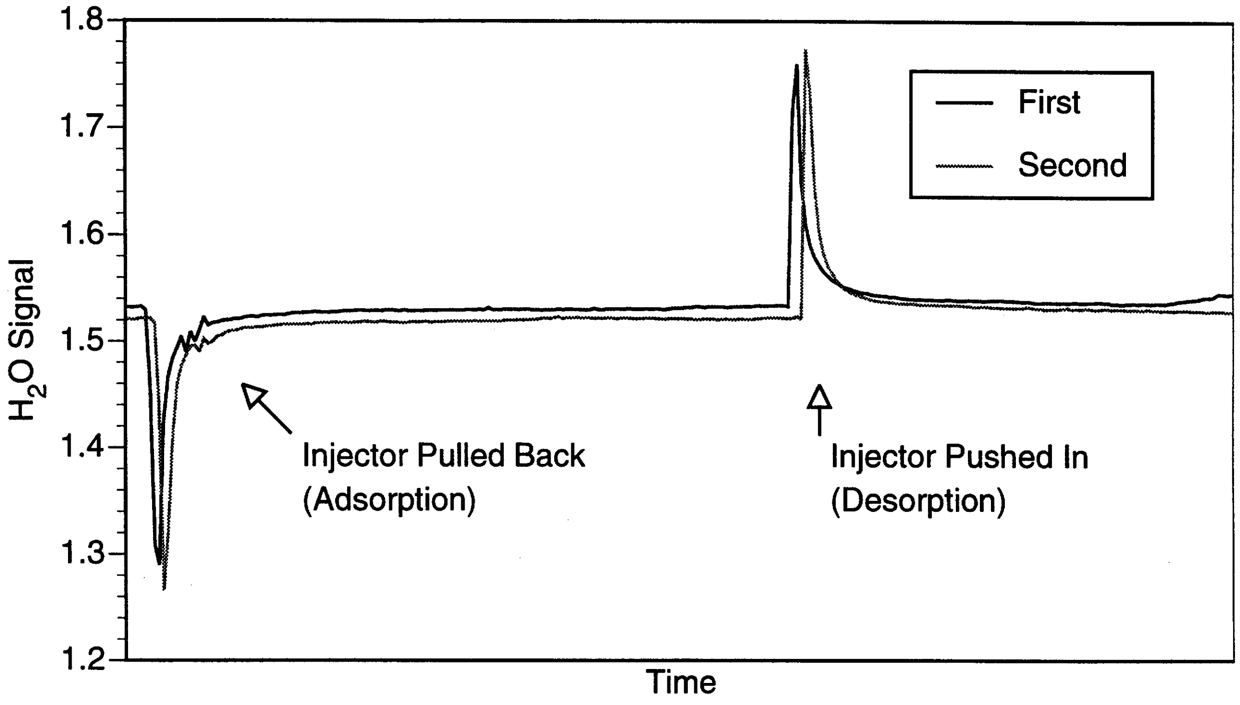


Figure III.5 Adsorption of H_2O on α-alumina. Unlike HCl or $ClONO_2$, there is no hysteresis.

	Alumina		Glass	
	Initial Adsorption	Ave. Desorption	Initial Adsorption	Ave. Desorption
H_2O mg/m^2	0.011	0.011	0.22	0.22
HCl mg/m^2	0.0073	0.0024	0.0036	0.0036
$ClONO_2$ mg/m^2	0.015	0.0049	0.067	0.032

Table 1. Surface coverage of various species on α-alumina and Pyrex™ glass.

TABLE 2.- REVERSIBLE AND IRREVERSIBLE SORPTION COVERAGES ON ALUMINA PREPARATIONS

(a) 75 percent H_2O saturation in N_2

Preparation	Near-equilibrium sorption coverage		Chemisorption coverage	
	mg/g	mg/m^2	mg/g	mg/m^2
3	32	0.34	12	0.13
4	33	.36	11	.12
6	84	.59	30	.21
7	---	----	---	-----
8	510	1.7	96	.32
9	173	.76	22	.097

(b) 75 percent HCl/H_2O saturation in N_2

Preparation	Near-equilibrium sorption (a)		Chemisorption (a)	
	mg/g	mg/m^2	mg/g	mg/m^2
3	224	2.4	108	1.2
4	223	2.4	101	1.1
6	314	2.2	149	1.05
7	8.7 ± 1	1.2	5 ± 1	.7
8	970	3.2	470	1.6
9	515	2.2	254	1.1

[a]Results from 300, 79, and 34 ppm HCl exposure data averaged.

(Taken from Cofer et al. (1978))

III.4 References for Chapter III

Abbatt, J.P.D., Molina, M.J., *Geophys. Res. Lett.*, **19**, 461, 1978.
Abbatt, J.P.D., Molina, M.J., *J. Phys. Chem.*, **96**, 7674, 1992.
Adamson, A.W., Physical Chemistry of Surfaces, 5 ed., published by Wiley-Interscience,1990.
Brunauer, S., Emmett, P.H., Teller, E., *J. Amer. Chem. Soc.*, **60**, 309 (1938).
Cofer, W.R., Pellet, G.L., Adsorption and Chemical Reaction of Gaseous Mixtures on Hydrogen Chloride and Water on Alumium Oxide and Application to Solid-Propellant rocket Exhuast Clouds, NASA Technical Paper 1105, 1978.
Molina, L.T., Spencer, J.E., Molina, M.J., *J. Chem. Phys. Lett.*, (1977), **45**, 158.
Molina, M.J., The Chemistry of the Atmosphere: Its Impact on Global Change, Edited by Jack G. Calvert, Blackwell Scientific Publications, 1994.
Zhang, R., Jayne, J.T., Molina, M.J., *J. Phys. Chem.*, **98**, 867, 1994.

Chapter IV: Possible Impact of Alumina at Mid Latitudes

IV.1 Introduction

Chapters II and III of this thesis have established that α–alumina adsorbs HCl and catalyzes the reaction:

$$ClONO_2 + HCl \rightarrow Cl_2 + HNO_3 \qquad (1)$$

with a reaction probability, γ, of approximately 0.02. The rate of this heterogeneous reaction is given by:

$$\gamma \cdot \frac{\overline{\upsilon}}{4} \cdot \frac{S}{V}[ClONO_2] \qquad (2)$$

where $\overline{\upsilon}$ is the average molecular speed of $ClONO_2$ and $\frac{S}{V}$ is the amount of surface area of the aerosol per unit volume of air. This compares to the homogeneous gas phase reaction rate which would be given by k[HCl][$ClONO_2$] where k is the bimolecular rate constant. Values for measured upper limits to k in the literature are listed in Table 1. This means that the acutal value may be lower because no one has yet been able to measure it. Each experiment reports the lower limit that they could detect.

Table 1.
Upper Limits for the Bimolecular Rate Constant for Reaction 1
(units of cm^3•$molec^{-1}$•s^{-1})

k	Source
$< 5 \times 10^{-18}$	*Friedl et al.*, 1986
$< 8.41 \times 10^{-21}$	*Hatakeymama et al.*, 1986
$< 1 \times 10^{-19}$	*Molina et al.*, 1985
$< 8.4 \times 10^{-21}$	*Leu et al.,* 1989
$< 2 \times 10^{-20}$	*Atkinson et al,* 1989

For the heterogeneous reaction to be faster than the gas phase reaction, it is neccesary that:

$$\gamma \cdot \frac{\overline{\upsilon}}{4} \cdot \frac{S}{V} > k[\mathrm{HCl}] \tag{3}$$

An effective first order rate constant for the homogeneous verses heterogeneous mechanisms with units of inverse seconds could be expressed as:

$$k_{\mathrm{eff}_{\mathrm{heterogeneous}}} = \gamma \cdot \frac{\overline{\upsilon}}{4} \cdot \frac{S}{V} \tag{4}$$

$$k_{\mathrm{eff}_{\mathrm{homogeneous}}} = k[\mathrm{HCl}] \tag{5}$$

When comparing, for example, aerosol A vs. aerosol B, to determine which one accelerates Reaction 1 more, aerosol A is more significant than B if:

$$\gamma_A \cdot \frac{S_A}{V} > \gamma_B \cdot \frac{S_B}{V} \tag{6}$$

The most recent stratospheric ozone models [Solomon et al., 1996] do not calculate the possible contribution from heterogeneous chemistry on alumina. Instead, they focus on other aerosols. These aerosols include the Polar Stratospheric Clouds (PSC's) and the sulfate aerosols which are present at mid latitudes and low altitudes (SSA). This is in an effort to account for the underprediction of ozone loss below 25 km in previous models [McCormick et al., 1992, WMO/UNEP, 1994, and references therein]. This chapter will examine the relative significance of these alumina particles to the sulfate aerosols at mid latitudes and low altitudes. An examination will also be made of another source of stratospheric aerosols, meteorite dust.

IV.2 Heterogeneous Chemistry on Sulfate Aerosols

The best known aerosols for heterogeneous chemistry in the stratosphere are the PSC's. As sulfuric acid particles move toward the poles, temperatures cool and the aerosols adsorb water and nitric acid. Eventually, during the polar winter, they form type I or II PSC's upon which Reaction 1 occurs relatively rapidly. However, these PSC's only last during the polar winter and only at ~200 K or below. Therefore, there other factors in the ozone depletion in the mid latitudes and lower altitudes that the models are trying to account for.

Table 2 Values used in Hofmann and Solomon (1989)

Altitude, km	Temperature, °C	H_2SO_4, %		γ values[b]		Estimated Surface Area, $\mu m^2\ cm^{-3}$	
		Measured	Theory[a]	$ClONO_2$	HCl	Volcanic	Background
15.9	−65	61.0 ± 2.0	62.2	0.0060	0.0045	10	0.25
16.4	−65	62.7 ± 1.7	62.6	0.0040	0.0027	16	0.28
18.0	−65	60.5 ± 4.5	64.2	0.0065	0.0050	38	0.40
19.2	−65	60.5 ± 2.5	65.4	0.0065	0.0050	45	0.58
21.5	−58	73.5 ± 1.0	72.2	0.0004	0.00012	24	0.63
25.1	−53	79.5 ± 1.0	76.7	0.00011	0.000020	5.5	0.39
30.5	−47	78.8 ± 1.2	81.0	0.00013	0.000024	1.2	0.10

[a]For 5 ppmv H_2O, [*Steele and Hamill*, 1981].
[b]Based on measured H_2SO_4 weight %, [*Hofmann and Rosen*, 1984*b*].

The rate of chlorine activation via Reaction 1 on SSA's was overestimated in earlier work. Hofmann and Solomon (1989) published a paper which studied the effect of the El Chichón on heterogeneous chemistry . Table 2 contains the values they used for γ for warmer temperatures and higher sulfuric acid concentrations. Figures IV.1 (Hanson and Ravishankara, 1994) and IV.2 (Hanson et al., 1994) show more recent experimentally derived values of γ for Reaction 1. It is clear that γ drops far below 10^{-5} at temperatures above 205 K. Extrapolating the curves for sulfuric acid to 230 K result in a γ of less than 10^{-11}.

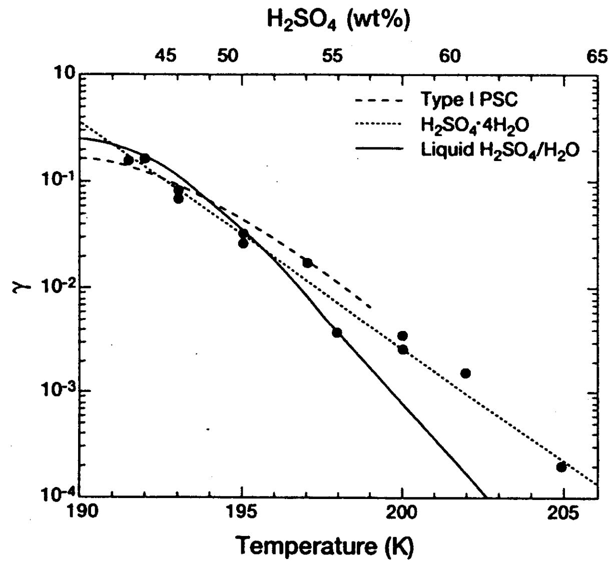


Figure IV.1. Temperature dependence of the reaction probability γ verses temperature for $ClONO_2 + HCl \rightarrow Cl_2 + HNO_3$. Surfaces are sulfuric acid tetrahydrate ($H_2SO_4{\cdot}4H_2O$ = SAT), nitric acid trihydrate (NAT), and Type I PSC's, a mixture of water and SAT. As temperature drops, droplets condense more water and the value of γ increases (adapted from Hanson and Ravishankara, 1994).

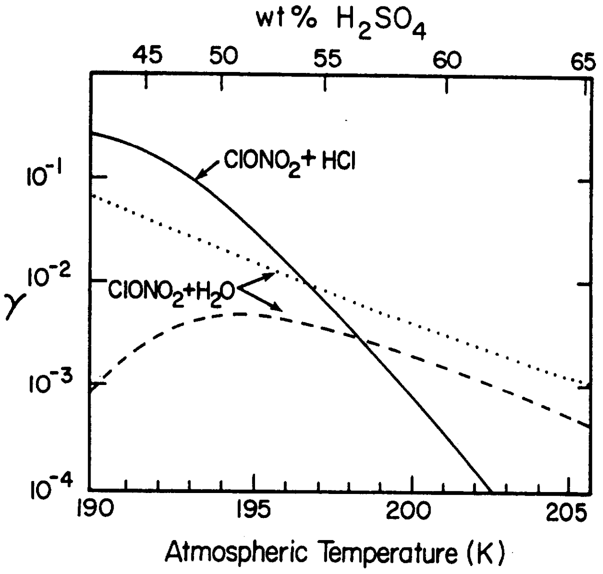


Figure IV.2. The uptake coefficients (γ) for $ClONO_2$ onto small liquid sulfuric acid droplets due to reactions with HCl (solid curve) and H_2O (dashed curve) are shown here from Hanson et al., 1994. The calculation was made with a typical water stratospheric value of 5 ppmv. The axis across the top shows the approximate equilibrium sulfuric acid concentration for the combination of water and temperature. Note that droplets are more dilute at lower temperatures and have higher γ values.

It becomes clear that SSA's are not important for Reaction 1 as previously supposed if one compares the rate of chlorine activation on SSA's to the upper limits to the homogeneous rate. To compare sulfate aerosol heterogeneous chemistry to background homogeneous chemistry at mid latitudes and low altitudes and 230 K, the following values will be used for Equation 4 and 5: [HCl] = 10^9 molec/cm^3 (JPL, 1994), γ = 1.0×10^{-11} (from extrapolations of Figures IV.1-2) , k = 8.4×10^{-21} cm^3·molec^{-1}·s^{-1} (the smallest upper rate limit in Table 1) and S/V of 10^{-8} cm^2/cm^3 (the background level in Figure IV.3):

$$k_{eff_{homogeneous}} = 8.4 \times 10^{-21}\ \text{cm}^3\cdot\text{molec}^{-1}\cdot\text{s}^{-1}\cdot 10^9\ \text{molec/cm}^3 = 8.4\times 10^{-12}\ \text{s}^{-1}$$

$$k_{eff_{heterogeneous}} = (1.0 \times 10^{-11} \cdot 19800\ \text{cm/s} \cdot 10^{-8}\text{cm}^2/\text{cm}^3)/4 = 1.2 \times 10^{-16}\text{s}^{-1}$$

SSA's are four orders of magnitude less in significance than the homogeneous rate which itself is too small to be measured. Clearly, SSA's are not a major direct mechanism for chlorine activation.

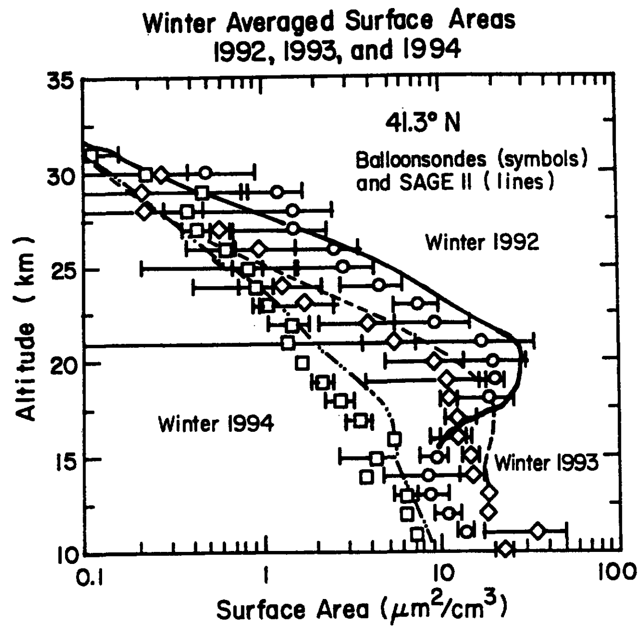


Figure IV.3. Observed aerosol surface areas near 40°N from satellite and balloon observations in winter 1992, 1994, and 1994. The SAGE II observations represent zonal and seasonal averages over the latitude band from 40° to 50°N, while the balloon sondes are in situ measurements from Laramie, Wyoming (41.3°N). The error bars indicate the 1-σ standard deviation of the available balloon observations in each year. Taken from Solomon et al., (1996.)

Solomon et al. (1996) examined the effects of the increased surface area of sulfates due to the eruption of Mt. Pinatubo and concluded that these sulfate aerosols were significant. The surface increase is depicted in Figure IV.3 and its effect is represented in Figure IV.4. The SSA impact on ozone depletion is not from direct chlorine activation, but from affecting the levels of NO_2 by helping to convert it to nitric acid by the following process:

$$NO_2 + O_3 \rightarrow NO_3 + O_2$$

$$NO_3 + NO_2 \rightarrow N_2O_5$$

$$N_2O_5 + H_2O \xrightarrow{\text{Hydrolysis on SSA's}} 2HNO_3$$

The reduction in NO_2 results in an enhanced role of HO_x and ClO_x ozone depletion processes.

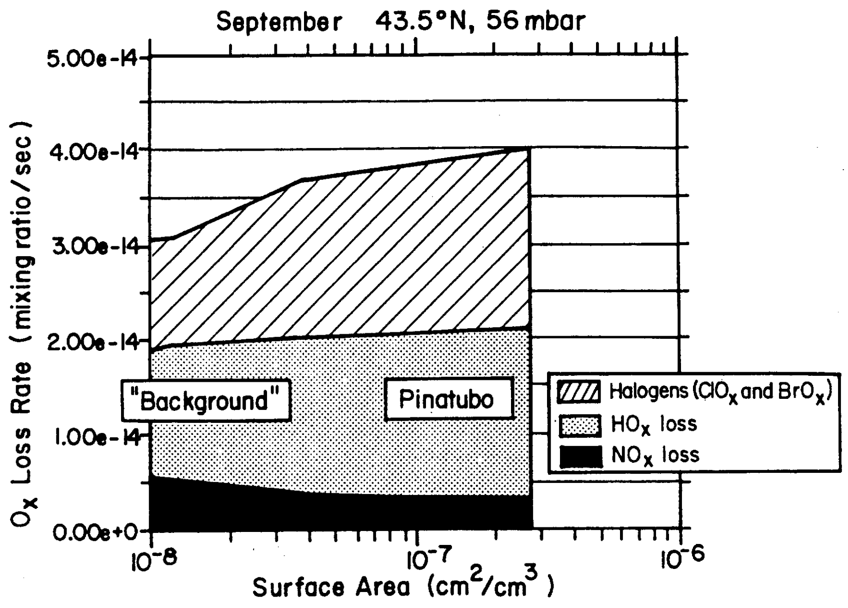


Figure IV.4. Calculated odd oxygen loss rates verses surface area at 43.5°N near 20km (56 mbar) for 1990 levels of total chlorine and bromine, for NO_X, HO_X, and halogen chemistry. Taken from Solomon et al, (1996).

IV.3 Meteorite Dust

Approximately 1.6 x 10^4 metric tons of meteorite dust hits the earth's atmosphere each year (Hughes, 1978). They are made of silicates, Fe, Ni, Al, Mg, and Ti metals and oxides. Chapter II noted that glass has a γ similar to that of alumina. To estimate to maximum possible impact of meteorite dust to heterogeneous chemistry in the lower stratosphere, we shall assume that all the material has the same γ as alumina.

Figure IV.5 by Turco et al. [1981] presents the complicated physical processes meteorites undergo in the earth's atmosphere. The dust particles start out small, often only sub micron size. These ablate in the mesosphere and the vapors condense as smoke of very small diameters.

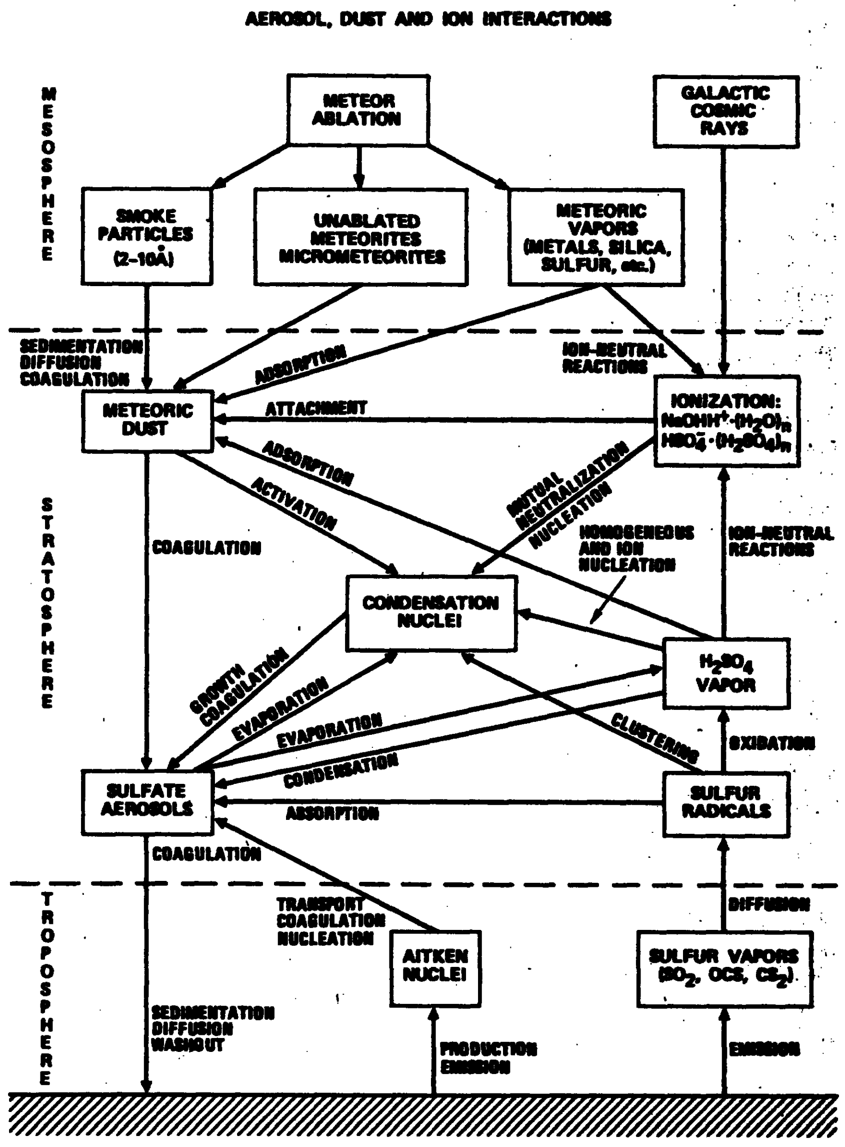


Figure IV.5. Schematic outline of the physical processes treated in this and earlier aerosol modeling studies. Taken from Turco et al [1981].

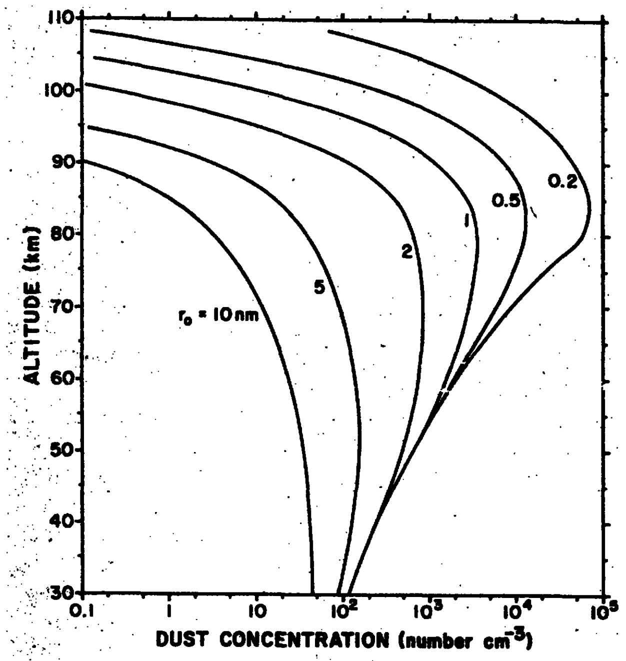


Figure IV.6. Steady state particle concentrations predicted using a detailed dust microphysics model. Concentration profiles are shown for several assumed initial smoke sizes, with the total dust mass input (versus height) held fixed. Taken from Hunten et al, (1980).

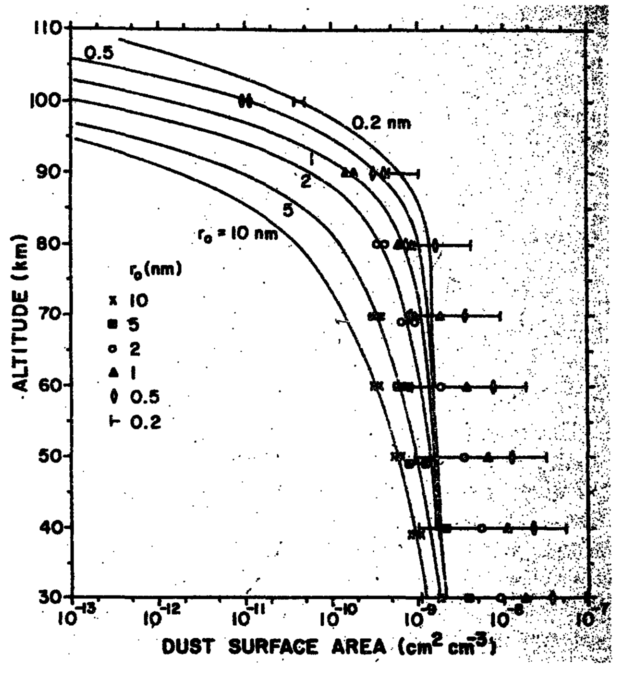


Figure IV.7. Total particle surface areas corresponding to the dust calculations in Fig. 6. The harmonic mean of the surface area for smooth, solid spheres, and the surface area for loosely packed agglomerations of spheres of the original smoke size are given. The crossbars delimit the surface area extremes. Taken from Hunten et al [1980].

Figures IV.6 and IV.7 report the results of models by Hunten et al. [1980]. Their models predict particles on the nanometer size range. This small size would give them a very long residence time in the stratosphere. Figure IV.7 reports a surface area per unit volume of approximately 10^{-9} cm^2/cm^3. This is within one tenth of the background value for sulfate aerosols in Figures IV.1 and IV.2!

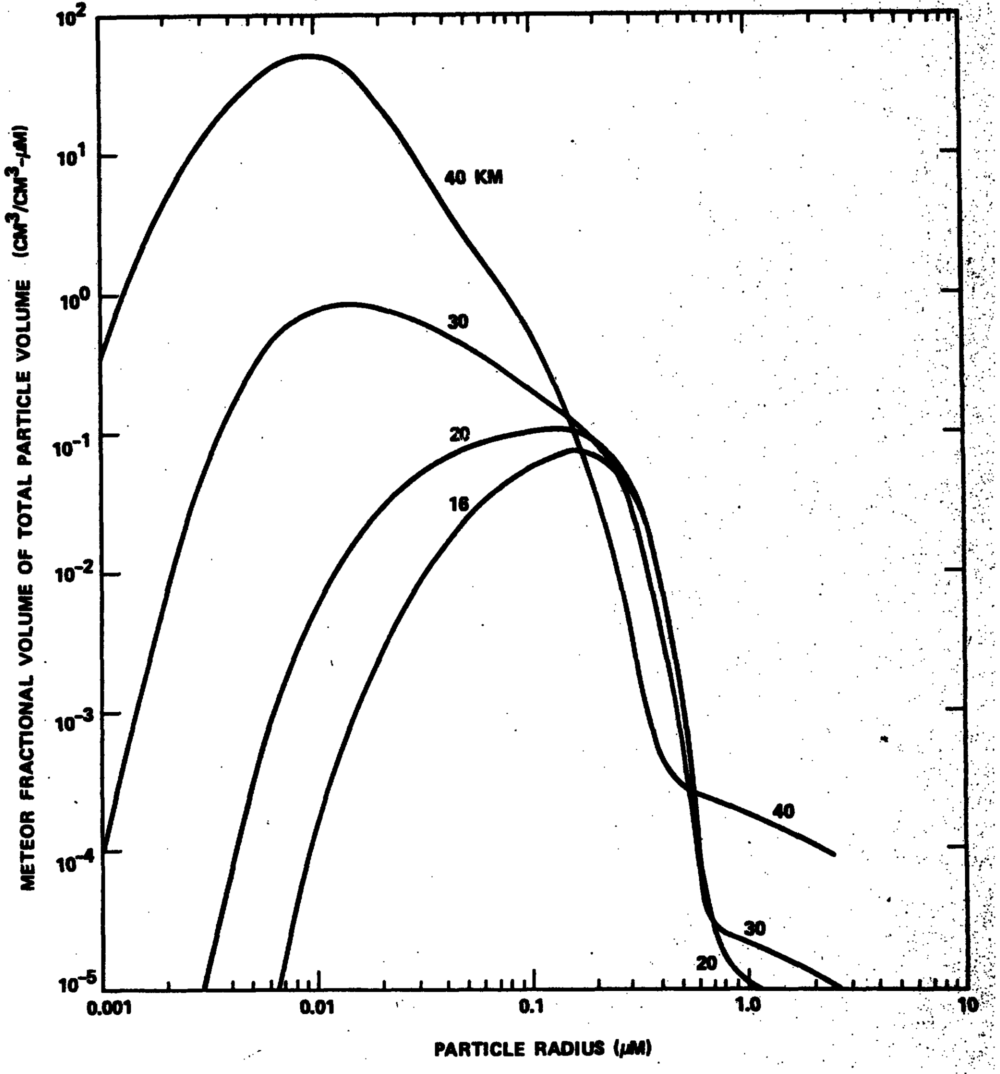


Figure IV.8. The contribution (per unit radius) of each particle size to the average meteor volume fraction of all the particles at that height. Thus the curves can be used to estimate the fraction of the total mass which is contained in any given particle size interval. Taken from Turco et al. [1981].

Figure IV.8 reveals that as the particles descend to lower elevations, they aggregate. This, coupled with their long residence time, makes them subject to becoming condensation nuclei for sulfuric acid! Figures IV.9 and IV.10 indicate that the fate of meteorite dust in the lower stratosphere is to be coated by sulfuric acid. Thus, they are already accounted for in the SSA measurements in Figures IV.1 and IV.2.

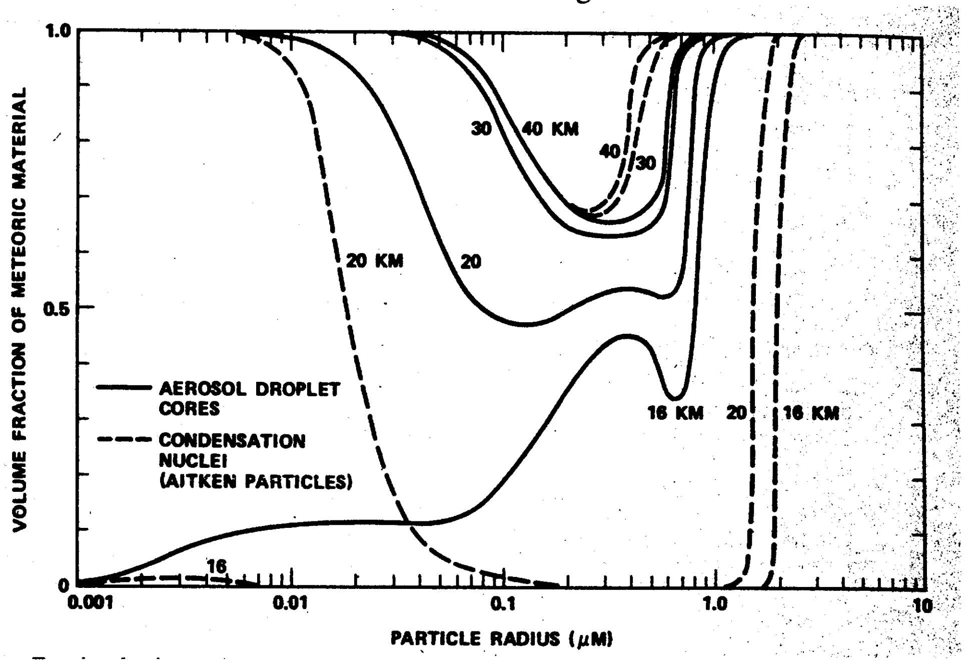


Figure IV.9. Fractional volume of meteoric material in condensation nuclei, and in aerosol droplet cores, as a function of particle (nuclei or droplet) size at several altitudes. The data can be interpreted as giving the average composition of individual particles. From Turco et al., (1981).

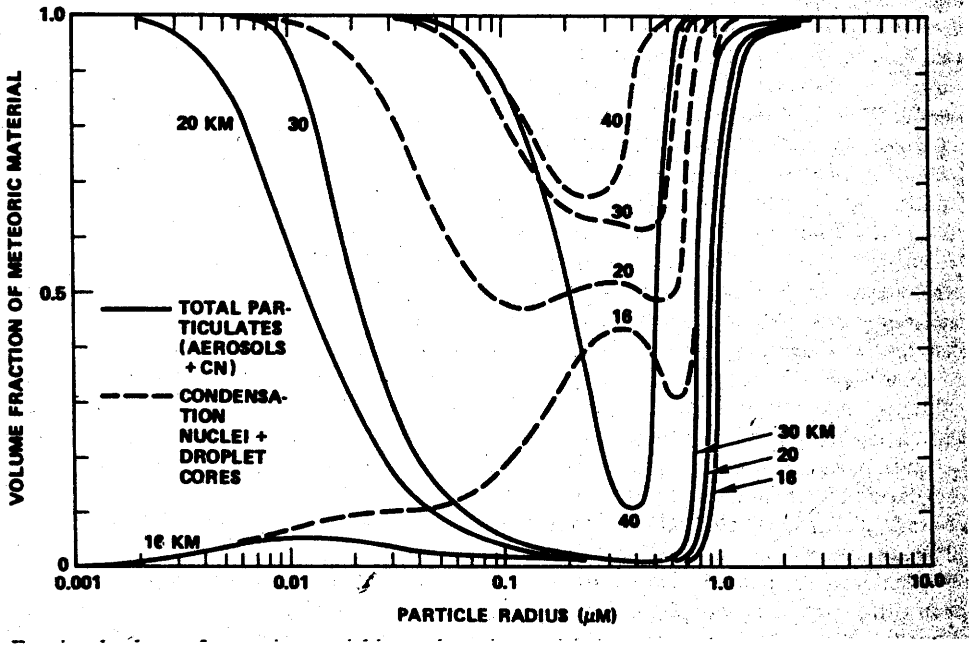


Figure IV.10. Fractional volume of meteoric material in condensation nuclei plus aerosol droplets and in condensation nuclei plus cores, as a function of particle size at several altitudes. From Turco et al. (1981).

IV.4 Shuttle Alumina

Several papers have been published on the environmental effects of the shuttle solid rocket motor exhaust and on the alumina's properties (Beiting et al., 1995; Cofer et al., 1985, 1987, 1992; Denison, 1994; Kruger, 1994; Park, 1976; Strand, 1981; and others). Shuttle alumina aerosols are spherical, have a density around 2 g/cm^3 (indicating porosity or a hollow center), and most of the mass is contained in particles greater than 1 µm in diameter. Figure IV.11 shows the alumina and chlorine emissions to the stratosphere from several launch vehicles.

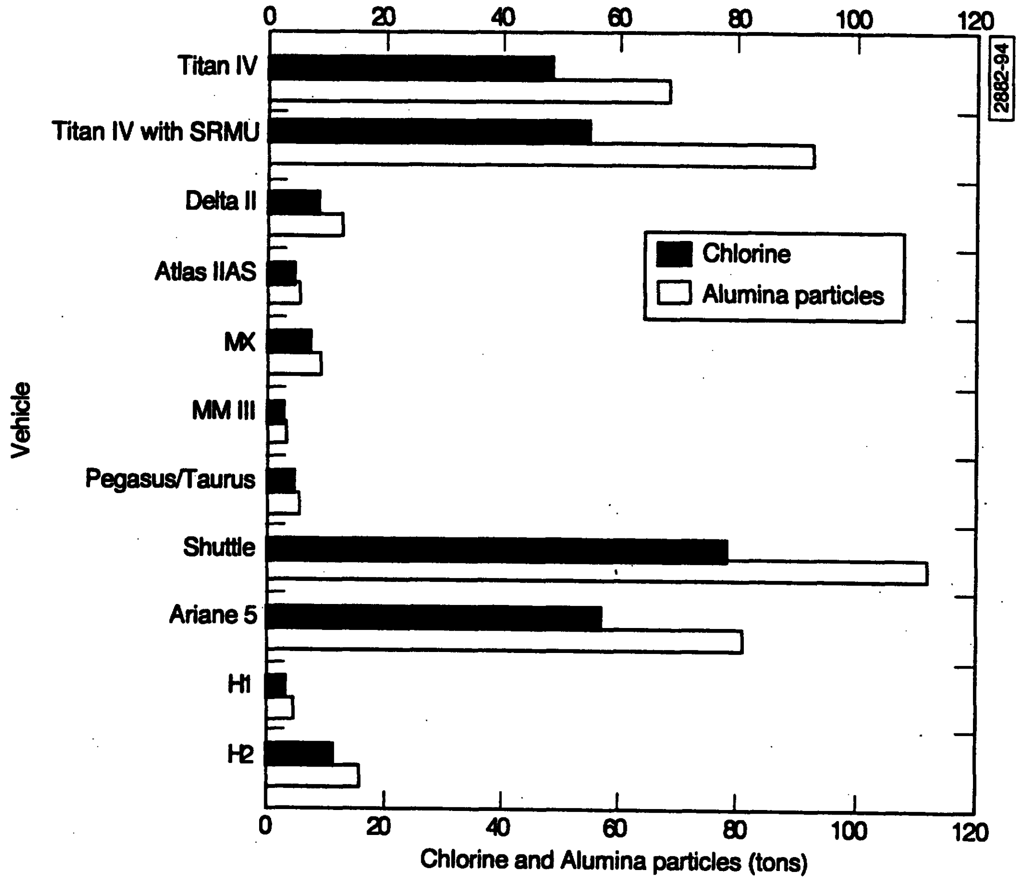


Figure IV.11. Chlorine and alumina particles deposited in the stratosphere by launch vehicles, per launch. Taken from Brady et al., 1994.

Table 2. Number densities of Ambient Stratospheric Particles (m^{-3}). Table from Beiting [1995]. Data from Zolensky et al. [1989].

Year	*Chon-dritic*	*Silicate*	*Al*	*Al'*	*Fe-S*	*Fe+S*	*CAS*	*Low-Z*	*Other*	*Total*
1976	0.010	0.030	0.004	0.018	0.007	0.004	0.002	0.002	0.013	0.089
1981	0.018	0.037	0.037	0.013	0.013	0.005	0.002	0.018	0.021	0.16
1984	0.017	0.22	0.71	0.15	0.051	0.051	0.002	0.27	0.22	1.7

Notes: Chondritic = extraterrestrial origin; Al = Aluminum and alumina components only; Al' = Primarily aluminum with lesser amount of other elements; CAS = calcium aluminum silicates. Activity and ablation of spacecraft material produces particles in the classes of Al, Al', silicate, Fe-S, Fe+S, low-Z, and other. Densities are in units of m^{-3}

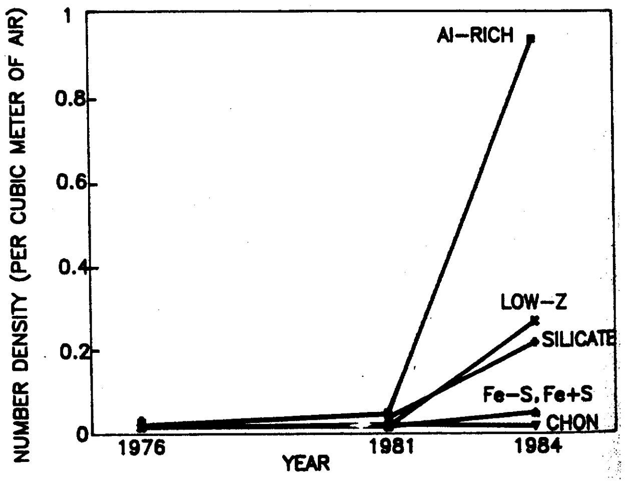


Figure IV.12. The number density of large (> 1-μm diameter) solid particles in the stratosphere, at 17-19 km altitude, at three sampling time during 1976-1984. Taken from Zokensy et al, 1989.

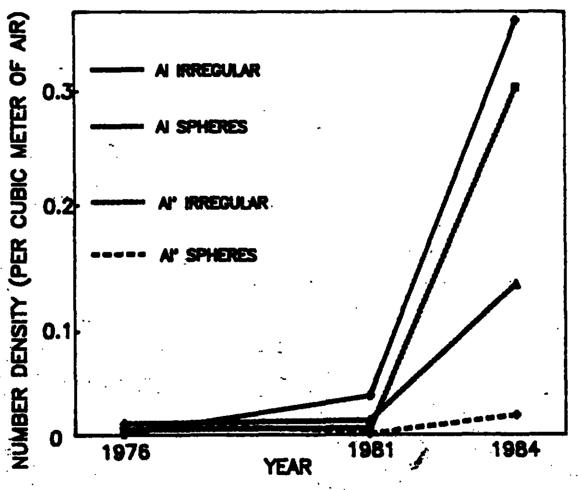


Figure IV.13. Number density of large (>1-μm diameter) Al rich particles in the stratosphere, at 17-19 km altitude, at three sampling times during the period 1976-1984. Taken from Zolensky et al., (1989).

Table 2 and Figures IV.12 and IV.13 show the results from Zolensky et al., [1989]. The amount of alumina and aluminum metal increased dramatically in number and in relation to other aerosols, such as those of extraterrestrial origin. This is due to man's activities. For example, the shuttle program began launching vehicles regularly in 1981 and each launch deposits 110 metric tons of alumina as shown in Figure IV.11. The average number of particles in only 1 per cubic meter. However, given that the number increased dramatically between 1981 and 1984 and that shuttle launches are still occurring, that number may be well below the current value. There appears to be no measurements post 1984.

Figure IV.14 contains a fit to the size distribution of the alumina exhaust from the space shuttle. The average diameter, weighted by surface area, is 3.2 μm for the STS flights. Figure IV.15 show the number and size distribution of samples taken in the troposphere after the shuttle was launched. Their peaks around 3 μm agree with the fit in Figure IV.14.

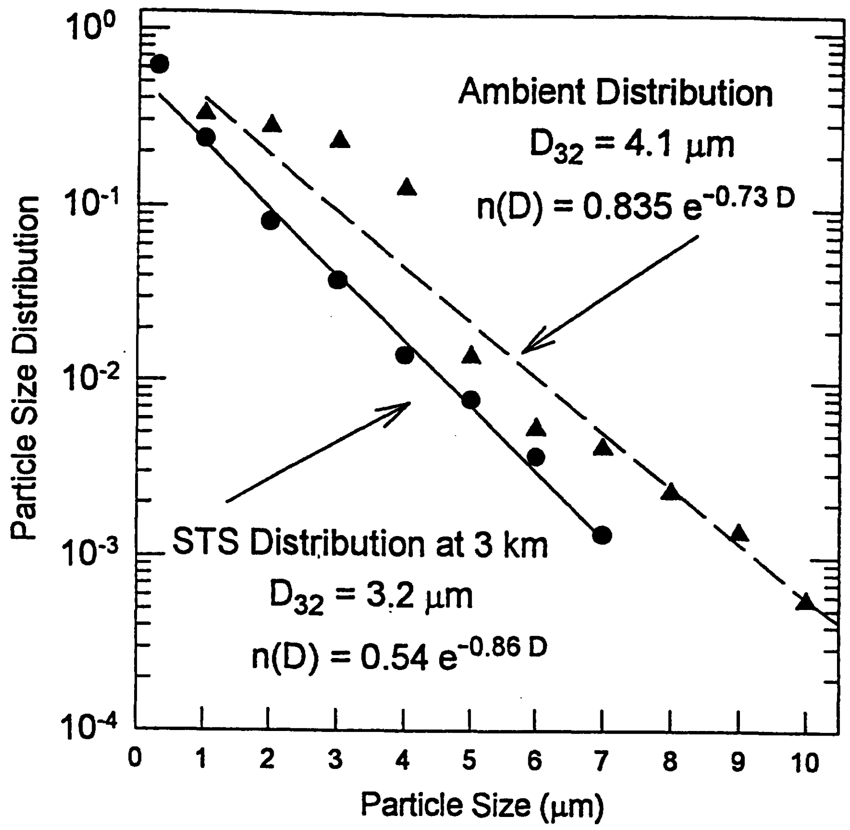


Figure IV.14. Taken from Beiting [1995]. Fits to ambient stratospheric particle data of Zolensky et al. [1989] and tropospheric STS plume particle data of Cofer et al. [1991].

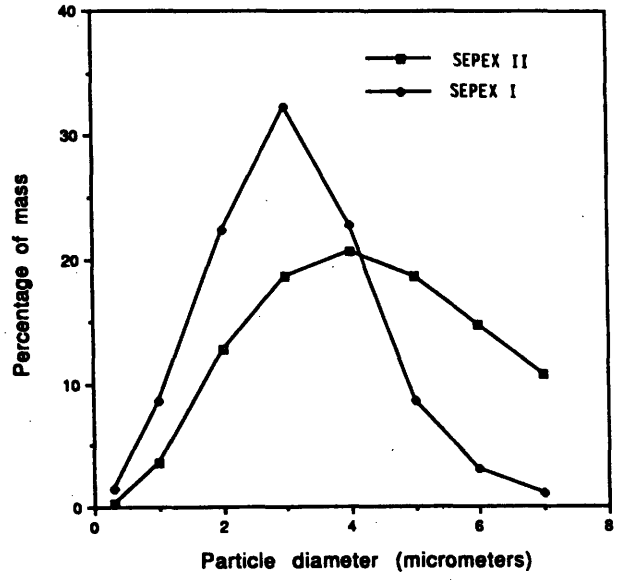


Figure IV.15. Percentage of mass at each diameter for SEPEX I and II particles. Taken from Cofet et al. [1991].

It is of interest to estimate the relative effect of the alumina from the shuttle vs. the chlorine it emits. Given the previous information, the total surface area of the alumina per launch is:

$$110 \text{ tons } Al_2O_3 \cdot 10^6 \frac{\text{g}}{\text{ton}} \cdot \frac{1\text{cm}^3}{2.0\text{g}} \cdot \frac{1 \text{ particle}}{\frac{4}{3}\pi(1.6\times10^{-4}\text{cm})^3} \cdot \frac{4\pi(1.6\times10^{-4}\text{cm})^2}{\text{particle}} = 1.0\times10^{12}\text{cm}^2$$

The rate of chlorine production with this surface area according to Equation 2 (using $[ClONO_2]$= 8 x 10^8 molec/cm^3 at 25 km altitude) would be:

$$0.02 \cdot \frac{19800 \,\text{cm}/\text{s}}{4} \cdot \frac{8\times10^8 \text{molec}}{\text{cm}^3} \cdot 1.0\times10^{12}\text{cm}^2 = 8.0\times10^{22} \,\text{molec}/\text{s}$$

The time to convert 80 metric tons of HCl to Cl_2 would be:

$$80 \text{ tons HCl} \cdot \frac{10^6 \text{g}}{\text{ton}} \cdot \frac{1 \text{ mole}}{36.453 \text{ g}} \cdot \frac{6.022\times10^{23}\text{molec}}{\text{mole}} \cdot \frac{1 \text{ s}}{8.0\times10^{22}\text{molec}} = 1.6\times10^7 \text{ s} = 6 \text{ months}$$

The chlorine gas produced from this reaction would rapidly photolyze to form chlorine radicals that would catalytically destroy ozone. However, the lifetime of chlorine radicals is very short because they quickly react with methane to form HCl which has a much greater lifetime. Thus, mostly of the chlorine gas produced by the alumina catalysis would quickly from HCl again. Therefore, the increase in the turn around time between HCl and Cl_2 can be compared to slowly adding an additional 80 tons of HCl over a year's time. The net result is a doubling of the ozone depletion potential of the solid rocket boosters. The alumina particulates appear to be as important as the original injection of HCl. This result merits the inclusion of heterogeneous chemistry on alumina in models that calculate ozone depletion from HCl released from solid rocket motors in order to better calculate the impact.

IV. References

Atkinson, R., Baulch, D.L., Cox, R.A., Hampson, R.F., Kerr, J.A., Troe, J., Evaluated kinetic and photochemical data for atmospheric chemistry: Supplement III, *J. Phys. Chem. Ref. Data*, **18**, 881, 1989.

Beiting, E.J., Characteristics of alumina particles from solid rocket motor exhuast in the stratosphere, Aerospace Report No. TR-95(5231)-8, 15 September 1995

Cofer, W.R., Bendura, R.J., Sebacher, D.I., Pellet, G.L., Gregory, G.L., Maddrea, G.I., Airborne measurements of space shuttle exhaust consitutuents, *AAIA Journal*, Vo. 23, No. 2, 1985

Cofer, W.R., Lala, G.G., Wightman, J.P., Analysis of mid-topospheric space shuttle exhausted aluminum oxide particles, *Atmospheric Envrionment*, **21**, 1187-1196, 1987

Cofer, W.R., Purgold, G.C., Winstead, E.L., Edalh, R.A., Space shuttle exhausted aluminum oxide: A measured particle size distribution, *J. Geophys. Res.*, **96**, D9, 17371-17376, 1991.

Denison, M.R., Lamb, J.J., Bjorndahl, WD., Wong, E., Lohn, P.D., Solid rocket exhaust in the stratosphere: plume diffusions and chemical reactions, *J. Spacecraft and Rockets*, **31**, No. 3, 435-442, 1994

Friedl, R.R., Goble, J.H., Sander, S.P., A kinetics study of the homogeneous and heterogeneous components on the HCl + $ClONO_2$ reaction, *Geophys. Res. Lett.*, **13**, 1351, 1986.

Hanson, D.R., Ravishankara, A.R., Heterogeneous bromine species in sulfuric acid under stratospheric conditions, *J. Geophys. Res.*, **99**, 3615-3629, 1994.

Hatakeyama, S., Leui, M.T., Reactions of chlorine nirate with HCl and H_2O, *Geophys. Res. Lett.*, **13**, 1343, 1986.

Hoffman, D.J., Solomon, S., Ozone destruction through heterogeneous chemistry following the eruption of El Chichon, *J. Geophys. Res.*, **94**, 5029-5041, 1989.

Hofmann, D.J., Rosen, J.M., Measurement of the sulfuric acid weight percent in the stratospheric aerosol from the El Chichòn eruption, *Geofis. Int.*, 23-3, 309-320, 1984b

Hughes, D.W., Meteors. *Cosmic Dust*, J. A. M. McDonnel, Ed., Wiley, Chap. 3, 1978

Hunten, D.M., Turco, R.P., Toon, O.B., Smoke and Dust Particles of Meteoric Origin in the Mesosphere and Stratosphere, *J. Atmos. Sci.*, **37**, 1342-1357, 1980.

Jackman, C.H., Considine, D.B., Fleming, E.L., The Space Shuttle's Impact on the Stratosphere: and Update, *J. Geophys. Res.*, in press, 1996.

JPL Publication 94-26, Chemical Kinetics and Photochemical Data for Use in Stratospheric Modeling, Evaluation Number 11, NASA, 1994

Kruger, B.C., Ozone depletion in the plume of a solid-fueled rocket, *Ann. Geophysicae*, **12**, 409-416, 1994.

Leu, M.T., Hatakeyama, S., Hsu, K.J., Rate constants for reactions between atmospheric resevoir species, *J. Phys. Chem*, **93**, 5778,1989.

McCormick, M.P., Veiga, R.E., Chu, W.P., Stratospheric ozone profile and total ozone trends derived from SAGE I and SAGE II data, *Geophys. Res. Lett.*, **19**, 269-272, 1992.

Molina, L.T., Molina, M.J., Stachnik, R.A., Tom, R.D., An upper limit to the rate of HCl + ClONO2 reaction, *J. Phys. Chem.*, **89**, 3779, 1985.

Park, C., High temperature reformation of aluminum and chlorine compounds behind the mach disk of a solid-fuel rocket exhaust, *Atmospheric Environment*, Vol. 10, pp. 693-702, 1976.

Solomon, S., Portmann, R.W., Garcia, R.R., Thomason, L.W., Poole, L.R., McCormick, M.P., The role of aerosol variations in anthropogenic ozone depletion at nothern midlatitudes, *J. Geophys. Res.*, **101**, D3, 6713-6727, 1996

Steele, H.M., Hamill, P., Effects of temperature and humidity on the growth and optical properties of sulphuric acid-wave droplet in the stratosphere, *J. Aerosol. Sci.*, **12**, 517-528, 1981.

Strand, L.D., Bowyer, J.M., Varsi, G., Laue, E.G., Gauldin, R., Characterization of particulates in the exhaust plume of large solid-propellant rockets, *J. Spacecraft*, **18**, 297-305, 1981

Turco, R.P., Toon, O.B., Hammill, P., Whitten, R.C., Effects of Meteoric Debris on Stratospheric Aerosols and Gases, *J. Geophys. Res.*, **86**, 1113-1128, 1981

World Meteoerological Organization/United Nations Environment Programme (WNO/UNEP), Scientific assessment of ozone depletion: 1994, Rep. 37, World Meteorol. Org., Geneva, 1994.

Zeman, E.J., Complex organic molecules found in interplanetary dust particles, *Physics Today*, 17-19, March 1994

Zolensky, M.E., McKay, D.S., Kaczor, L.A., A tenfold increase in the abundance of large solid particles in the stratosphere, as measured over the period of 1975-1984, *J. Geophys. Res.*, **94**, 1047-1056, 1989.

Zolensky, MacKinnon, I.D.R., Accurate stratospheric size distributions from a float plate collection surface, *J. Geophys. Res.*, D3, 5801-5808, 1985.

Chapter V: Current Direction and Future Work

This work has found that α-alumina does activate chlorine in the reaction

$$HCl + ClONO_2 \rightarrow Cl_2 + HNO_3 \qquad (1)$$

with a γ value of 0.02. The resulting level of chlorine activation makes the environmental impact of the alumina significant compared to the HCl emitted by solid rocket motors—including the boosters used by the space shuttle. The alumina exhauted from one launch would take only about 6 months to activate all the HCl released from the same launch. Therefore, any calculation that attempts to calculate the environmental impact of shuttle launches or other solid rocket motor launches must pay equal attention to the long term impact of the heterogeneous chemistry on the alumina particulates. Adsorption of sulfuric acid or settling out from the stratosphere would be the main sinks for this active aerosol. Modelers are strongly encouraged to incorporate these results along with the rest of the heterogenous chemistry they now use.

This work has also found that α-alumina near stratospheric conditions chemically adsorbs HCl with hysteresis and reversibly adsorbs H_2O. It may also chemically adsorb $ClONO_2$. Glass also adsorbed HCl but in a much more reversible fashion like H_2O. Slight hysteresis of adsorption of $ClONO_2$ similar to alumina may be due to to hydrolosis reactions with adsorbed layers of H_2O on the surface. Glass also had a value of γ close to that of alumina. However, other materials such as Teflon or 60% sulfuric acid displayed no chlorine activation. These facts point to the possibility that any hydrophlic substance with water available on the surface could catalyze reaction 1 with a γ on the order of 0.01.

To provide more accurate data for computer models to caculate the impact of alumina in the stratosphere, more work needs to be done. This work crosses the boundary of work in the laboratory, in the field, and theoretical modeling.

First, what is the current level of alumina in the stratosphere? The latest data is over a decade old (Zolensky, 1989). Is there a good prediction of alumina levels in the future forcasted launch rates? At the same time it would be useful to measure the ratio of α to γ alumina. Previous work (Cofer et al., 1984) states that shuttle alumina is 72% α phase while a more recent report (Dill et al., 1990) concludes it is 64% γ phase.

Next, we need to repeat the same experiments on the γ phase of alumina to see if it behaves significantly different. Also, since Cofer (1989) measured more highly chlorinated alumina in shuttle dust than we used, would this affect the value of the sticking coefficient γ? Would more heavily chlorinated samples be more or less active? Does alumina promote the hydrolysis of $ClONO_2$ to HOCl? How quickly do large alumina aerosols get coated and deactivated by sulfuric acid?

Our lab is currently pursuing a course of action that can answer some of these questions. We could do more experiments on more heavily chlorinated alumina. However, the mass spectrometer described in chapter II is currently being modified to work in negative ion mode. The sampled gas will be selectively ionized by SF_6 that passes through a corona discharge. This process is more selective because it doesn't ionize the helium buffer gas and it also avoids ionizing pump oil in the chamber. Thus, our machine will be more sensitive to more peaks of interest. For instance, we could more easily see HOCl, the result from hydrolysis of $ClONO_2$ on alumina. Also, it will be able to distinguish between HNO_3 and $ClONO_2$. This will allow repeats of our kinetic experiments without interference from the HNO_3 signal. It will also help us see if HNO_3 can poison the surface.

Because negative ion mode can work at higher pressures, the experimental setup could be modified to move the buffer gas through a short length of tubing filled with alumina powder. This would give us greater surface areas and would make adsorption experiments easier. Experiments could also be done with background levels of sulfuric acid vapor to try and determine how quickly it coats the alumina particles.

There currently exists work on sulfuric acid condensing on aerosols (Hofmann et al., 1975; Pollack et al. 1976; Turco et al. 1980) but we would encourage modelers to better refine their models to the rate of condensation on shuttle aerosols as well a other aerosols that well up from the troposphere, by eruptions or convection. This would help us know how much more of the hydrophilic aerosols present could help activate chlorine.

Finally, modelers are encouraged to incorporate of value of γ of 0.02 in their global 2-D models. If alumina does not appear to be significant now, it would still be useful to know how many launches it would take per year for it to become significant. That way, we would not be caught unprepared by a long-term growth in satellite launch rates or other space activity such as building and maintaining a space station.

References for Chapter V

Cofer, W.R., Pellett, G.L., Sebacher, D.I., Wakelyn, N.T., Surface chloride salt formation on Space Shuttle exhaust alumina, *J. Geophys. Res.*, **89**, 2535-2549, 1984.

Cofer, W.R., Winstead. E.L., Key, L.E., Surface composition of solid-rocket exhausted aluminum oxide particles, *J. Propulsion*, **5**, No. 6, 675-677, 1989

Dill, K.M., Reed, R.A., Calia, V.S., Schulta, R.J., Analysis of crystalline phase aluminum oxide particles from solid propellant exhauts, *J. Propulsion Power*, **6**, 688-691, 1990.

Hofmann, D.J., Carol, D.E., Rosen, J.M., *Geophys. Res. Lett.*, **2**, 113-116, 1975.

Pollack, J.B., Toon, O.B., Summers, A., Van Camp, W., Baldwin, B.J., *Appli. Meteor.*, **15**, 247-258, 1976.

Turco, R.P. et al., *J. Appl. Meteor.*, **19**, 78-89, 1980

Appendix I: Data Acquisition Code

This appendix contains the computer code that runs several of the mass spectrometer systems in our laboratory. It is included for several reasons. First, it was as major project as building an instrument. Second, it provides a permanent record for future graduate students to evaluate if they suspect the computer is making an error in control, measurement, or calculation. Finally, it provides a road map for future students who may be ambitious enough to change the program to better suit their needs.

National Instruments LabWindows 2.3 for DOS provides a graphical user interface for scientific work. Using a visual resource editor, you create a "UIR" file (user interface resource) which contains the definitions of all menu items, controls, graphs, colors, etc. Then you write the code that performs your calculations or calls the appropriate library routines to handle input or output.

This program is called MODULAR.EXE and consists of the following files:

MODULAR.UIR	User Interface Resource Definition.
MODULAR.H	Contains heads giving labels to all user interface items.
STRUCT.H	Contains function and data definitions needed for this program.
MODULAR.C	Contains the main() function.
EDIT2.C	Code to modify experimental parameters and labels.
MASSSPEC.C	Code for mass spectrometer mode.
CHART.C	Code for chart recorder mode.
KINETICS.C	Code for kinetics experiments mode.
READFLOW.C	Code to read all flowmeters and pressure inputs.
DRIVERS.C	Code to call any one of supported data acquisition cards.
DT2801.C, DT2819.C	Drivers for various data translation boards.

```
                              /* MODULAR.H*/
/* ======================================================================= */
/* LabWindows User Interface Resource (UIR) Include File                   */
/* Copyright (c) National Instruments 1993. All Rights Reserved.           */
/*                                                                         */
/* WARNING: Do not add to, delete from, or otherwise modify the contents   */
/*          of this include file.                                          */
/* ======================================================================= */

#define  CHARTMENU  0
#define  CHARTMENU_FILE  0
#define  CHARTMENU_FILE_SAVEDATA  1
#define  CHARTMENU_FILE_SAVESETUP  2
#define  CHARTMENU_FILE_PRINT  3
#define  CHARTMENU_FILE_PRINTSETUP  4
#define  CHARTMENU_FILE_QUIT  5
#define  CHARTMENU_SETUP  256
#define  CHARTMENU_SETUP_CHAN  257
#define  CHARTMENU_SETUP_FLOWTUBPAR  258
#define  CHARTMENU_SETUP_YLOGON  259
#define  CHARTMENU_SETUP_YLOGOFF  260
#define  CHARTMENU_SETUP_CHANGEXY  261
#define  CHARTMENU_SWITCH  512
#define  CHARTMENU_SWITCH_KINETICS  513
#define  CHARTMENU_SWITCH_MASS_SPEC  514

#define  MASSMENU  1
#define  MASSMENU_FILE  0
#define  MASSMENU_FILE_SAVEDATA  1
#define  MASSMENU_FILE_SAVESETUP  2
#define  MASSMENU_FILE_GETOLDSPEC  3
#define  MASSMENU_FILE_PRINT  4
#define  MASSMENU_FILE_PRINTSETUP  5
#define  MASSMENU_FILE_QUIT  6
#define  MASSMENU_SETUP  256
#define  MASSMENU_SETUP_CHAN  257
#define  MASSMENU_SETUP_FLOWTUBPAR  258
#define  MASSMENU_SETUP_YLOGON  259
#define  MASSMENU_SETUP_YLOGOFF  260
#define  MASSMENU_SETUP_CHANGEXY  261
#define  MASSMENU_SWITCH  512
#define  MASSMENU_SWITCH_KINETICS  513
#define  MASSMENU_SWITCH_CHART  514

#define  KINMENU  2
#define  KINMENU_FILE  0
#define  KINMENU_FILE_SAVEDATA  1
#define  KINMENU_FILE_SAVESETUP  2
#define  KINMENU_FILE_PRINT  3
#define  KINMENU_FILE_PRINTSETUP  4
#define  KINMENU_FILE_QUIT  5
#define  KINMENU_ACQUIRE  256
#define  KINMENU_ACQUIRE_GETPOINT  257
#define  KINMENU_ACQUIRE_DELPOINT  258
#define  KINMENU_ACQUIRE_INCREASEX  259
#define  KINMENU_ACQUIRE_DECREASEX  260
#define  KINMENU_ACQUIRE_RESETX  261
#define  KINMENU_ACQUIRE_LINFIT  262
```

```
#define   KINMENU_ACQUIRE_DELGRAPH  263
#define   KINMENU_ACQUIRE_CHANGEMASS  264
#define   KINMENU_SETUP  512
#define   KINMENU_SETUP_CHAN  513
#define   KINMENU_SETUP_FLOWTUBPAR  514
#define   KINMENU_SETUP_YAUTON  515
#define   KINMENU_SETUP_YAUTOFF  516
#define   KINMENU_SETUP_XAUTON  517
#define   KINMENU_SETUP_XAUTOFF  518
#define   KINMENU_SETUP_CHANGEXY  519
#define   KINMENU_SWITCH  768
#define   KINMENU_SWITCH_MASS_SPEC  769
#define   KINMENU_SWITCH_CHART  770

#define   DYCORMENU  3
#define   DYCORMENU_FILE  0
#define   DYCORMENU_FILE_SAVEDYCOR  1
#define   DYCORMENU_FILE_SAVEBET  2
#define   DYCORMENU_FILE_SAVESETUP  3
#define   DYCORMENU_FILE_PRINTDYCOR  4
#define   DYCORMENU_FILE_PRINTBET  5
#define   DYCORMENU_FILE_QUIT  6
#define   DYCORMENU_ACQUIRE  256
#define   DYCORMENU_ACQUIRE_GETDATA  257
#define   DYCORMENU_ACQUIRE_VOLABS  258
#define   DYCORMENU_ACQUIRE_PLOTPOINT  259
#define   DYCORMENU_ACQUIRE_DELPOINT  260
#define   DYCORMENU_ACQUIRE_LINFIT  261
#define   DYCORMENU_ACQUIRE_DELGRAPH  262
#define   DYCORMENU_SETUP  512
#define   DYCORMENU_SETUP_CHAN  513
#define   DYCORMENU_SETUP_FLOWTUBPAR  514

#define   STARTMENU  4
#define   STARTMENU_BOARD_SET  0
#define   STARTMENU_SWITCH  256
#define   STARTMENU_SWITCH_KINETICS  257
#define   STARTMENU_SWITCH_MASS_SPEC  258
#define   STARTMENU_SWITCH_CHART  259
#define   STARTMENU_QUIT  512

#define   FLOWSPANEL  0
#define   FLOWSPANEL_CHAN1  0
#define   FLOWSPANEL_CHAN2  1
#define   FLOWSPANEL_CHAN3  2
#define   FLOWSPANEL_CHAN4  3
#define   FLOWSPANEL_CHAN5  4
#define   FLOWSPANEL_CHAN6  5
#define   FLOWSPANEL_CHAN7  6
#define   FLOWSPANEL_RN  7
#define   FLOWSPANEL_VEL  8
#define   FLOWSPANEL_SPECIES1  9
#define   FLOWSPANEL_SPECIES2  10
#define   FLOWSPANEL_SPECIES3  11
#define   FLOWSPANEL_TIME  12
#define   FLOWSPANEL_TEMP  13

#define   SETCHANNEL  1
```

```
#define  SETCHANNEL_CHANUM  0
#define  SETCHANNEL_TYPE  1
#define  SETCHANNEL_BARORANGE  2
#define  SETCHANNEL_FLOWRANGE  3
#define  SETCHANNEL_BULBMIX  4
#define  SETCHANNEL_MIXRATIO1  5
#define  SETCHANNEL_MIXRATIO2  6
#define  SETCHANNEL_MIXRATIO3  7
#define  SETCHANNEL_BUBVP  8
#define  SETCHANNEL_BUBVP1  9
#define  SETCHANNEL_BUBVP2  10
#define  SETCHANNEL_BUBVP3  11
#define  SETCHANNEL_ASSOCBARO  12
#define  SETCHANNEL_FINISHED  13
#define  SETCHANNEL_EDITNAMES  14
#define  SETCHANNEL_METERTYPE  15

#define  SPECNAME  2
#define  SPECNAME_SPEC1  0
#define  SPECNAME_SPEC2  1
#define  SPECNAME_SPEC3  2
#define  SPECNAME_CHAN1  3
#define  SPECNAME_CHAN2  4
#define  SPECNAME_CHAN3  5
#define  SPECNAME_CHAN4  6
#define  SPECNAME_CHAN5  7
#define  SPECNAME_CHAN6  8
#define  SPECNAME_CHAN7  9
#define  SPECNAME_SPECIESBAR  10
#define  SPECNAME_CHANBAR  11
#define  SPECNAME_FINISHED  12

#define  FLOWTPARAM  3
#define  FLOWTPARAM_FLOWTEMP  0
#define  FLOWTPARAM_RADIUS  1
#define  FLOWTPARAM_FINISHED  2
#define  FLOWTPARAM_CARRIERGAS  3

#define  SELECTMASS  4
#define  SELECTMASS_MASSVALUE  0
#define  SELECTMASS_INCRBY1  1
#define  SELECTMASS_DECRBY1  2
#define  SELECTMASS_INCRBYP1  3
#define  SELECTMASS_DECRBYP1  4
#define  SELECTMASS_FINISHED  5
#define  SELECTMASS_MASSIGNAL  6

#define  XYRANGE  5
#define  XYRANGE_XMAX  0
#define  XYRANGE_XMIN  1
#define  XYRANGE_YMAX  2
#define  XYRANGE_YMIN  3
#define  XYRANGE_FINISHED  4

#define  KINETICS  6
#define  KINETICS_GRAPH  0
#define  KINETICS_SLOPE  1
#define  KINETICS_INTERCEPT  2
```

```
#define  KINETICS_SLOPE_ERR  3
#define  KINETICS_INTERCEPT_ERR  4
#define  KINETICS_SIGNAL  5
#define  KINETICS_INJECTOR_POSITION  6
#define  KINETICS_TIME  7

#define  MASSSPEC  7
#define  MASSSPEC_SCAN  0
#define  MASSSPEC_DELETE  1
#define  MASSSPEC_SPECTRUM  2
#define  MASSSPEC_CURSORBOX  3
#define  MASSSPEC_X_POS  4
#define  MASSSPEC_Y_POS  5
#define  MASSSPEC_C1_LABEL  6
#define  MASSSPEC_C2_LABEL  7
#define  MASSSPEC_DELTA_LABEL  8
#define  MASSSPEC_C1_X  9
#define  MASSSPEC_C2_X  10
#define  MASSSPEC_DELTA_X  11
#define  MASSSPEC_C1_Y  12
#define  MASSSPEC_C2_Y  13
#define  MASSSPEC_DELTA_Y  14
#define  MASSSPEC_ZOOMIN  15
#define  MASSSPEC_ZOOMOUT  16
#define  MASSSPEC_RESTORE  17
#define  MASSSPEC_RANGEBOX  18
#define  MASSSPEC_SWEEPRATE  19
#define  MASSSPEC_STARTMASS  20
#define  MASSSPEC_ENDMASS  21
#define  MASSSPEC_DATAPOINT  22
#define  MASSSPEC_SCANBOX  23

#define  STRIPCHART  8
#define  STRIPCHART_CHART  0
#define  STRIPCHART_DWELL  1
#define  STRIPCHART_CHAN_DELAY  2
#define  STRIPCHART_MASS1  3
#define  STRIPCHART_MASS2  4
#define  STRIPCHART_MASS3  5
#define  STRIPCHART_MASS4  6
#define  STRIPCHART_MASS5  7
#define  STRIPCHART_START  8
#define  STRIPCHART_STOP  9
#define  STRIPCHART_CLEAR  10

#define  DYCOR  9
#define  DYCOR_DYCOR  0
#define  DYCOR_BET  1
#define  DYCOR_VOLABS  2
#define  DYCOR_VMON  3
#define  DYCOR_BET_C  4
#define  DYCOR_GASMAX  5

#define  DAQSETUP  10
#define  DAQSETUP_ADC_TYPE  0
#define  DAQSETUP_DAC_LABEL  1
#define  DAQSETUP_DETECTOR  2
#define  DAQSETUP_SLOT_NUMBER  3
```

```
#define  DAQSETUP_QUAD_RANGE  4
#define  DAQSETUP_OUT  5
#define  DAQSETUP_AOUT_LABEL  6
#define  DAQSETUP_FINISHED  7

#define  PRINTSETUP  11
#define  PRINTSETUP_PRINT_TO  0
#define  PRINTSETUP_PAGE_ORIENTATION  1
#define  PRINTSETUP_SCALE_FACTOR  2
#define  PRINTSETUP_PAGE_EJECT  3
#define  PRINTSETUP_OUTPUTFILE  4
#define  PRINTSETUP_FINISHED  5
```

```
                                /* STRUCT.H */
/*=== Includes ===========================================================*/


#include <stdio.h>
/*#include <math.h>*/
#include <dos.h>
#include <string.h>
#include "c:\lw\include\lwsystem.h"
#include "c:\lw\include\userint.h"
#include "c:\lw\include\dataacq.h"
#include "b:\modular\modular.h"
/*=== Insert other necessary LabWindows includes here====================*/

/*=== DEFINES ============================================================*/
#define RESOURCE_FILE "modular.uir"
#define MASS_CNVRT 20
#define WAIT 1
#define NO_WAIT 0
#define INTEGER 2
#define DOUBLE 8
#define NO_RESTRICT 0
#define RESTRICT 1
#define FILE_EXISTS 1
#define NEW_FILE 2
#define TRUE 1
#define FALSE 0
#define TUBE_BARO 0
#define BUBBLE_BARO 1
#define BUBBLER 2
#define BULB 3
#define LABEL 0
#define MAX 13
#define MIN 12
#define BAROTRON_VOLTAGE 10.0
#define MASS_FLOWTMETER_VOLTAGE 5.0
#define TINY 1.0e-12
#define PI 3.14159
#define HE 0
#define NITROGEN 1
#define ARGON 2
#define  THOUSAND 0
#define  HUNDRED 1
#define TEN 2
#define ONE 3
#define PCLPM16 13
#define KINETICS_PANEL 1
#define MASSSPEC_PANEL 2
#define STRIPCHART_PANEL 3
#define QUIT 0

struct device {
   int index;           /* Control panel index
       */
   double range;        /* Maximum range of device
       */
   double gas_correct; /* Correction factor if carrier gas is different   */
```

```
                                      /* than the device was meant to measure
*/
};

struct gas_input {
   int type;          /* 0=TUBE_BARO,1=BUBBLE_BARO,2=BUBBLER,3=BULB       */
   struct device barotron;
   struct device flowmeter;
   double range;      /* Corrected range for device                       */
   double amount[3];  /* mixing ratios in bulb or vapor press. in bubbler */
   int press_chan;    /* channel # for bubbler's barotron                 */
   double voltage;    /* scaled voltage from board                        */
   double max_volts;   /* Maximum voltage from device
      */
   double reading;    /* Scaled reading in appropriate units for user     */

};

struct FlowTubeParameters {
   int gas_type;      /* Control panel index                             */
   double gas_correct;/* heat capacity correction in flowmeters          */
   double gas_mw;     /* Molecular weight of carrier gas                 */
   double temp;       /* Flowtube Temperature in Kelvin                  */
   double radius;     /* Flowtube radius in centimeters                  */
   double eata;       /* Reynold's number constant                       */
};

struct PlotParameters {
   double x[100];           /* x coordinate                                */
   double y[100];           /* y coordinate                                */
   double y_sigma[100];     /* standard deviation of each y point          */
   int numpoints;           /* number of points taken                      */
   int plot_hdl[100];         /* plot handle for each point                   */
   int err_hdl[100];        /* plot handle for error bar on each point     */
   int plotstyle;           /* plot style number                           */
   int plotcolor;           /* plot color number                           */
   double slope;            /* slope of line fit                           */
   double intercept;        /* intercept of fit                            */
   double sigma_slope;      /* standard deviation of slope                 */
   double sigma_intercept;  /* standard deviation of intercept             */
};
struct sample {
   double mean;             /* Statistical Mean of the sample              */
   double variance;         /* Statistical Variance of the sample          */
};
struct Titles {
   char species[3][40];     /* names of the chemical species               */
   char channel[8][40];     /* names given to each input channel           */
};
struct Axis {
   int yauto;
   int xauto;
   double x_min;
   double x_max;
   double y_min;
   double y_max;
};
struct handles {
   short hFlowsPanel;
   short hChannelSetup;
   short hEditLabels;
   short hFlowtubeParams;
   struct {
      short hPrintSetup;
      short PrintTo;
      short Orientation;
      short ScaleFactor;
      short PageEject;
      char OutputFile[80];
   } PrintSetup;
};

/*=== FUNCTION DECLARATIONS ==========================================*/
void  ActivateBarotron(struct handles *panel);
void  ActivateBubblerFlow(struct handles *panel);
void  ActivateBulbFlow(struct handles *panel);
unsigned char bin(char *string);
void  Board2819Ready(void);
void  ChangeXYRange(short hPanel,short graph,short change_x,short change_y,
                        short xauto,double x_min,double x_max,short yauto,
                         double y_min,double y_max);
short chart(struct gas_input *channel, struct handles *panel,
                  struct FlowTubeParameters *flowtube,struct Titles *labels);
float CounterRate(short milliseconds);
void  DeactivateCtrls(struct handles *panel);
float DTVIn(unsigned char channel);
void  DTVOut(unsigned char channel,float voltage);
short dycor(struct gas_input *channel, struct handles *panel,
                  struct FlowTubeParameters *flowtube, struct Titles *labels);
void  EditNames(struct Titles *labels,struct handles *panel);
void  EditFlowtubeParamters(struct FlowTubeParameters *flowtube,
                           struct handles *panel,struct gas_input *channel);
float GetCounts(short milliseconds);
short InitializeBoards(void);
void  InitDT2801(void);
void  InitDT2819(void);
short kinetic(struct gas_input *channel, struct handles *panel,
                  struct FlowTubeParameters *flowtube, struct Titles *labels);
short LoadProgramPanels(struct handles *panel);
short mass_spec(struct gas_input *channel, struct handles *panel,
                        struct FlowTubeParameters *flowtube,struct Titles
*labels);
void  PrintSetup(struct handles *panel);
void  ReadFlows(struct gas_input *channel,struct handles *panel,
                        struct FlowTubeParameters *flowtube,
                        double *partial_pressure,double *total_flow,
                        double *tube_pressure,double *velocity);
float ReadVoltage(short channel);
short SaveSetupParameters(struct gas_input *channel,
                                     struct FlowTubeParameters
*flowtube,struct Titles *labels);
void  SetChanNum(struct gas_input *channel,struct handles *panel);
void  SetupChannels(struct gas_input *channel,struct Titles *labels,
                  struct handles *panel,struct FlowTubeParameters *flowtube);

short SetupParameters(struct gas_input *channel,struct handles *panel,
                  struct FlowTubeParameters *flowtube,struct Titles *labels);
void  UpdateChannelPanel(struct Titles *labels,struct handles *panel);
void  UpdateMainCtrls(struct gas_input *channel, struct Titles *labels,
         struct handles *panel);
void  wlsf(struct PlotParameters *plot);
void  WriteVoltage(float voltage);
```

```
                                    /* MODULAR.C */
extern unsigned _stklen = 8192;
#define KINETICS_PANEL 1
#define MASS_SPEC_PANEL 2
#define CHART_PANEL 3
#define NI_12BIT 0
#define DT_12BIT 1
#include <stdio.h>

#include "b:\modular\struct.h"

/* Global Variables */
short AnalogBoardType, CounterOrAnalog, AnalogOutput, Slot, QuadRange;
float SamplesPerMS;
short GetHardwareInfo(void);

void main(void) {
   int hMenuBar,id,handle,hPanel,i,mode;
   struct FlowTubeParameters flowtube;
   struct gas_input channel[8];
   struct Titles labels;
   struct handles panel;
   FILE *fp;

   /* ready print setup parameters */
   panel.PrintSetup.PrintTo=1         /* LPT1 */;
   panel.PrintSetup.OutputFile[0]=0; /* null file */
   panel.PrintSetup.Orientation=0;    /* Portrait */
   panel.PrintSetup.ScaleFactor=0;    /* Screen Size */
   panel.PrintSetup.PageEject=0;      /* No Page Eject */

   /* check for hardware setup */
   if((fp=fopen("hardware.prm","rb"))!=NULL) { /* file exists, load values */
      /* get Analog Board Type */
      fread(&AnalogBoardType,sizeof(int),1,fp);
      /* Counter Board or Analog Board on Mass Spec? */
      fread(&CounterOrAnalog,sizeof(int),1,fp);
      /* Get Output Channel to Mass Spec */
      fread(&AnalogOutput,sizeof(int),1,fp);
      /* Get Slot number for possible NI Board */
      fread(&Slot,sizeof(int),1,fp);
      /* Get Range of Mass Spec */
      fread(&QuadRange,sizeof(int),1,fp);
      /* close file */
      fclose(fp);
   }
   else { /* create file and save results */
      /* inform user of need to creat hardware file */
      MessagePopup("HARDWARE.PRM NOT FOUND.  PREPARE TO ENTER BOARD
INFORMATION.");
      AnalogBoardType=0;
      CounterOrAnalog=0;
      AnalogOutput=1;
      Slot=0;
      QuadRange=500;
      if(!GetHardwareInfo()) return;

   }
   /* load main menu bar */
   hMenuBar = LoadMenuBar (RESOURCE_FILE, STARTMENU);
   if (hMenuBar < 0) {
      MessagePopup("Unable to load the required menu bar from the resource
file.");
      return;
   }

   id = STARTMENU_BOARD_SET;
   while (id == STARTMENU_BOARD_SET) {
      GetUserEvent (WAIT, &handle, &id);
      switch(id) {
         case STARTMENU_BOARD_SET:
            if(!GetHardwareInfo()) return;
            break;
         default:
            break;
      }
   }
   if (id == STARTMENU_QUIT) return; /* quit now */
   /* User is done modifying the hardware setup.  Now run regular setup with
*/
   /* all the flows.  Prepare to pass information to any of the three modes
*/

   if (!(LoadProgramPanels(&panel))) {
      MessagePopup("Error: can't get panels.  Exiting...");
      return;
   }

   DisplayPanel (panel.hFlowsPanel);
   /* Setup Paramters */

   if(!SetupParameters(channel,&panel,&flowtube,&labels)) return;

   /* Setup Boards */

   if(!InitializeBoards()) return;

   /* Unload menu bar */
   UnloadMenuBar();

   /* User has chosen one of the three modes.  Now loop between modes */
   switch (id) {
      case STARTMENU_SWITCH_KINETICS: /* ADD LATER */
         mode = kinetic(channel,&panel,&flowtube,&labels);
         break;
      case STARTMENU_SWITCH_MASS_SPEC:
         mode = mass_spec(channel,&panel,&flowtube,&labels);
         break;
      case STARTMENU_SWITCH_CHART:
         mode = chart(channel,&panel,&flowtube,&labels);
         break;
      default:
         mode = chart(channel,&panel,&flowtube,&labels);
         break;
   }
   while(TRUE) {
      switch(mode) {
         case KINETICS_PANEL:
            mode = kinetic(channel,&panel,&flowtube,&labels);
            break;
         case MASSSPEC_PANEL:
            mode = mass_spec(channel,&panel,&flowtube,&labels);
            break;
         case STRIPCHART_PANEL:
            mode = chart(channel,&panel,&flowtube,&labels);
            break;
         case QUIT:
            return;
         default:
            MessagePopup("Unknown choice.  Exiting...");
            return;
      }
   }
}

short GetHardwareInfo(void) {
   int hPanel,id,i;
   FILE *fp;

   hPanel = LoadPanel (RESOURCE_FILE, DAQSETUP);
   if (hPanel < 0) {
         MessagePopup("Unable to load the required panel from the resource
file.");
         return 0;
   }
   InstallPopup(hPanel);
   /* Let user input information */
   id = -1;
   /* Set Panel to Current Values */
   SetCtrlVal(hPanel,DAQSETUP_ADC_TYPE,AnalogBoardType);
   SetCtrlVal(hPanel,DAQSETUP_DETECTOR,CounterOrAnalog);
   SetCtrlVal(hPanel,DAQSETUP_OUT,AnalogOutput);
   SetCtrlVal(hPanel,DAQSETUP_SLOT_NUMBER,Slot);
   SetCtrlVal(hPanel,DAQSETUP_QUAD_RANGE,QuadRange);
   while(id != DAQSETUP_FINISHED ) {
      GetPopupEvent(WAIT,&id);
      switch(id) {
         case DAQSETUP_ADC_TYPE:
            /* See if it's a National Instruments board.  If not,  */
            /* deactive slot # control */
            GetCtrlVal(hPanel,DAQSETUP_ADC_TYPE,&i);
            if(i == NI_12BIT) { /* Activate slot # control */
               SetInputMode(hPanel,DAQSETUP_SLOT_NUMBER,TRUE);
            }
            else if( i == DT_12BIT ) { /* Deactivate slot # control */
               SetInputMode(hPanel,DAQSETUP_SLOT_NUMBER,FALSE);
            }
            break;
         default:
            break;
      }
   }
   /* Get all control values */
   GetCtrlVal(hPanel,DAQSETUP_ADC_TYPE,&AnalogBoardType);
   GetCtrlVal(hPanel,DAQSETUP_DETECTOR,&CounterOrAnalog);
   GetCtrlVal(hPanel,DAQSETUP_OUT,&AnalogOutput);
   GetCtrlVal(hPanel,DAQSETUP_SLOT_NUMBER,&Slot);
   GetCtrlVal(hPanel,DAQSETUP_QUAD_RANGE,&QuadRange);
   /* unload Panel */
   RemovePopup(0);
   UnloadPanel(hPanel);
   /* open file and save results */
   if((fp=fopen("hardware.prm","wb"))!=NULL) {
      /* save Analog Board Type */
      fwrite(&AnalogBoardType,sizeof(int),1,fp);
      /* Counter Board or Analog Board on Mass Spec? */
      fwrite(&CounterOrAnalog,sizeof(int),1,fp);
      /* save Output Channel to Mass Spec */
      fwrite(&AnalogOutput,sizeof(int),1,fp);
      /* save Slot number for possible NI Board */
      fwrite(&Slot,sizeof(int),1,fp);
      /* save Range of Mass Spec */
      fwrite(&QuadRange,sizeof(int),1,fp);
      /* close file */
      fclose(fp);
   }
   else { /* Error, notify user and continue */
      MessagePopup("Disk error, can't save hardware parameters.");
   }
   return 1;
}
```

```
                                /* EDIT2.C */
#include "struct.h"

/*=============================================================================
==*/
/* These routines are dedicated to mapping the channels on the board to either
*/
/* flowmeters or barotrons.  The baratrons are used to monitor the pressure of
*/
/* the flowtube or the pressure over a bubbler which sets its mixing ratio
when */
/* the vapor pressures of the solution are known.  Flowmeters are either
*/
/* attached to a bubbler or a gas bulb with the mixing ratios previously set.
*/
/* These routine also provide a way to link the barotron on a bubbler to the
*/
/* bubbler itself.  The interactive panel activates or deactivates the valid
*/
/* choices depending on the device chosen.  You can also change the names of
*/
/* the channels and the names of the chemical species.  You also can enter
their*/
/* vapor pressues or mixing ratios
*/
/*=============================================================================
==*/

void SetupChannels(struct gas_input *channel,struct Titles *labels,
      struct handles *panel,struct FlowTubeParameters *flowtube){
  /*  Logic of Channel Popup is:                              */
  /*      If the channel # is changed, update panel to show that       */
  /*      channel's parameters.  Deactivate or activate appropriate     */
  /*      controls based on channel's type.                             */
  /*      If the channel type is changed, activate or deactivate the    */
  /*      appropriate controls.                                         */
  /*      If a setting is changed, place value in appropriate variable. */

  int flag,control,i,val;

  InstallPopup(panel->hChannelSetup);
  flag = TRUE;
  while(flag) {
    GetPopupEvent(WAIT,&control);
    GetCtrlVal(panel->hChannelSetup,SETCHANNEL_CHANUM,&i);
    switch (control) {
      case SETCHANNEL_CHANUM:
        SetChanNum(channel,panel);
        break;
      case SETCHANNEL_TYPE:
        GetCtrlVal(panel->hChannelSetup,SETCHANNEL_TYPE,&channel[i].type);
        if((channel[i].type==TUBE_BARO)||(channel[i].type==BUBBLE_BARO)) {
          ActivateBarotron(panel);
          channel[i].max_volts = BAROTRON_VOLTAGE;
        }
        else if(channel[i].type == BUBBLER) {
          ActivateBubblerFlow(panel);
          channel[i].max_volts = MASS_FLOWTMETER_VOLTAGE;
```

```
                GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_ASSOCBARO,&channel[i].press_chan);
            }
            else if(channel[i].type == BULB) {
                ActivateBulbFlow(panel);
                channel[i].max_volts = MASS_FLOWTMETER_VOLTAGE;
            }
            break;
        case SETCHANNEL_BARORANGE:
            channel[i].max_volts = BAROTRON_VOLTAGE;
            /* Get variables of that type */
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_BARORANGE,&channel[i].barotron.index);
            switch (channel[i].barotron.index) {
                case ONE:
                    channel[i].barotron.range = 1;
                    break;
                case TEN:
                    channel[i].barotron.range = 10;
                    break;
                case HUNDRED:
                    channel[i].barotron.range = 100;
                    break;
                case THOUSAND:
                    channel[i].barotron.range = 1000;
                    break;
                default:
                    channel[i].barotron.range = 1000;
                    break;
            }
            break;
        case SETCHANNEL_FLOWRANGE:
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_FLOWRANGE,&channel[i].flowmeter.range);
            break;
        case SETCHANNEL_MIXRATIO1:
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_MIXRATIO1,&channel[i].amount[0]);
            break;
        case SETCHANNEL_MIXRATIO2:
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_MIXRATIO2,&channel[i].amount[1]);
            break;
        case SETCHANNEL_MIXRATIO3:
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_MIXRATIO3,&channel[i].amount[2]);
            break;
        case SETCHANNEL_BUBVP1:
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_BUBVP1,&channel[i].amount[0]);
            break;
        case SETCHANNEL_BUBVP2:
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_BUBVP2,&channel[i].amount[1]);
            break;
        case SETCHANNEL_BUBVP3:
            GetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_BUBVP3,&channel[i].amount[2]);
```

```
            break;
         case SETCHANNEL_METERTYPE:
            channel[i].max_volts = MASS_FLOWTMETER_VOLTAGE;
            GetCtrlVal(panel->hChannelSetup,SETCHANNEL_METERTYPE,
               &channel[i].flowmeter.index);
            switch (channel[i].flowmeter.index) {
               case HE:       /* Helium */
                  channel[i].flowmeter.gas_correct = 1;
                  break;
               case NITROGEN: /* Nitrogen */
                  channel[i].flowmeter.gas_correct = 1.42;
                  break;
               case ARGON:    /* Argon */
                  channel[i].flowmeter.gas_correct = 1;
                  break;
               default:
                  channel[i].flowmeter.gas_correct = 1;
                  break;
            }
            break;
         case SETCHANNEL_ASSOCBARO:
            GetCtrlVal(panel->hChannelSetup,SETCHANNEL_ASSOCBARO,&val);
            if(channel[val].type != BUBBLE_BARO) {
               MessagePopup("Warning! That channel is not presently set for a
barotron!");
            }
            channel[i].press_chan=val;
            break;
         case SETCHANNEL_FINISHED:
            flag = FALSE;
            break;
         case SETCHANNEL_EDITNAMES:
               EditNames(labels,panel);
               break;
      }
   }
   /* Correct flowmeter ranges for the gases used */
   for(i=1;i<=7;i++) {
      if((channel[i].type == BUBBLER)||(channel[i].type == BULB)) {
         channel[i].range = channel[i].flowmeter.range *
            channel[i].flowmeter.gas_correct / flowtube->gas_correct;
      }
      else channel[i].range = channel[i].barotron.range;
   }
   RemovePopup(0);         /* Removes current popup */
}

/*-------------------------------------------------------------------*/
/* This function sets all the controlto the values associated with the   */
/* channel number indicated on the channel control slide.             */
/*-------------------------------------------------------------------*/
void SetChanNum(struct gas_input *channel,struct handles *panel) {
   int i;
   GetCtrlVal(panel->hChannelSetup,SETCHANNEL_CHANUM,&i);
   SetCtrlVal(panel->hChannelSetup,SETCHANNEL_TYPE,channel[i].type);
   if((channel[i].type == TUBE_BARO)||(channel[i].type==BUBBLE_BARO)) {
      ActivateBarotron(panel);
      /* Set the proper values on the proper controls */
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_BARORANGE,channel[i].barotron.index);
   }
   else if(channel[i].type == BUBBLER) {
      ActivateBubblerFlow(panel);
      /* Set the proper values on the proper controls */
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_FLOWRANGE,channel[i].flowmeter.range);
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_METERTYPE,channel[i].flowmeter.index);
      SetCtrlVal(panel->hChannelSetup,SETCHANNEL_BUBVP1,channel[i].amount[0]);
      SetCtrlVal(panel->hChannelSetup,SETCHANNEL_BUBVP2,channel[i].amount[1]);
      SetCtrlVal(panel->hChannelSetup,SETCHANNEL_BUBVP3,channel[i].amount[2]);
      SetCtrlVal(panel->hChannelSetup,SETCHANNEL_ASSOCBARO,7-
channel[i].press_chan);
   }
   else if(channel[i].type == BULB) {
      ActivateBulbFlow(panel);
      /* Set the proper values on the proper controls */
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_FLOWRANGE,channel[i].flowmeter.range);
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_METERTYPE,channel[i].flowmeter.index);
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_MIXRATIO1,channel[i].amount[0]);
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_MIXRATIO2,channel[i].amount[1]);
      SetCtrlVal(panel-
>hChannelSetup,SETCHANNEL_MIXRATIO3,channel[i].amount[2]);
   }
}


/*==========================================================================*/
/*      Functions to aid the activation and deactivation of the appropriate
*/
/* controls in the MAIN_SETUP_CHAN panel.                                   */
/*==========================================================================*/

void ActivateBarotron(struct handles *panel) {
   DeactivateCtrls(panel);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BARORANGE,TRUE);
}

void ActivateBubblerFlow(struct handles *panel) {
   DeactivateCtrls(panel);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_FLOWRANGE,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_METERTYPE,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP1,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP2,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP3,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_ASSOCBARO,TRUE);
}

void ActivateBulbFlow(struct handles *panel) {
   DeactivateCtrls(panel);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BULBMIX,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_MIXRATIO1,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_MIXRATIO2,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_MIXRATIO3,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_FLOWRANGE,TRUE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_METERTYPE,TRUE);
}

void DeactivateCtrls(struct handles *panel) {
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BARORANGE,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_METERTYPE,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_FLOWRANGE,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BULBMIX,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_MIXRATIO1,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_MIXRATIO2,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_MIXRATIO3,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP1,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP2,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_BUBVP3,FALSE);
   SetInputMode(panel->hChannelSetup,SETCHANNEL_ASSOCBARO,FALSE);
}

/*=========================================================================*/
/*      Functions to Update Labels and Ranges on Controls
*/
/*=========================================================================*/

void UpdateChannelPanel(struct Titles *labels,struct handles *panel) {
   /* Update species names */
   SetCtrlAttribute(panel->hChannelSetup,SETCHANNEL_MIXRATIO1,LABEL,labels-
>species[0]);
   SetCtrlAttribute(panel->hChannelSetup,SETCHANNEL_MIXRATIO2,LABEL,labels-
>species[1]);
   SetCtrlAttribute(panel->hChannelSetup,SETCHANNEL_MIXRATIO3,LABEL,labels-
>species[2]);
   SetCtrlAttribute(panel->hChannelSetup,SETCHANNEL_BUBVP1,LABEL,labels-
>species[0]);
   SetCtrlAttribute(panel->hChannelSetup,SETCHANNEL_BUBVP2,LABEL,labels-
>species[1]);
   SetCtrlAttribute(panel->hChannelSetup,SETCHANNEL_BUBVP3,LABEL,labels-
>species[2]);
}

void UpdateMainCtrls(struct gas_input *channel, struct Titles *labels,
            struct handles *panel) {
   /* Update species names */
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_SPECIES1,LABEL,labels-
>species[0]);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_SPECIES2,LABEL,labels-
>species[1]);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_SPECIES3,LABEL,labels-
>species[2]);

   /* Update channel names */
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN1,LABEL,labels-
>channel[1]);
```

```
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN2,LABEL,labels-
>channel[2]);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN3,LABEL,labels-
>channel[3]);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN4,LABEL,labels-
>channel[4]);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN5,LABEL,labels-
>channel[5]);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN6,LABEL,labels-
>channel[6]);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN7,LABEL,labels-
>channel[7]);

   /* Set Channel Values to 0 to prevent overflow with new maximum */
   SetCtrlVal(panel->hFlowsPanel,FLOWSPANEL_CHAN1,0);
   SetCtrlVal(panel->hFlowsPanel,FLOWSPANEL_CHAN2,0);
   SetCtrlVal(panel->hFlowsPanel,FLOWSPANEL_CHAN3,0);
   SetCtrlVal(panel->hFlowsPanel,FLOWSPANEL_CHAN4,0);
   SetCtrlVal(panel->hFlowsPanel,FLOWSPANEL_CHAN5,0);
   SetCtrlVal(panel->hFlowsPanel,FLOWSPANEL_CHAN6,0);
   SetCtrlVal(panel->hFlowsPanel,FLOWSPANEL_CHAN7,0);

   /* Update channel maximums */
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN1,MAX,channel[1].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN2,MAX,channel[2].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN3,MAX,channel[3].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN4,MAX,channel[4].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN5,MAX,channel[5].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN6,MAX,channel[6].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN7,MAX,channel[7].range);

   /* Update channel minimums */

   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN1,MIN,-
.01*channel[1].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN2,MIN,-
.01*channel[2].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN3,MIN,-
.01*channel[3].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN4,MIN,-
.01*channel[4].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN5,MIN,-
.01*channel[5].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN6,MIN,-
.01*channel[6].range);
   SetCtrlAttribute(panel->hFlowsPanel,FLOWSPANEL_CHAN7,MIN,-
.01*channel[7].range);
}

/* -- Routines to simply edit names of labels, concentrations, or mixing
ratios --*/

void EditNames(struct Titles *labels,struct handles *panel) {
   int id,flag;
   flag=TRUE;

   InstallPopup(panel->hEditLabels);
   while (flag) {
      GetPopupEvent(WAIT,&id);
      switch (id) {
         case SPECNAME_FINISHED:
            flag = FALSE;
            break;
      }
   }
   /* Get labels */
   GetCtrlVal(panel->hEditLabels,SPECNAME_SPEC1,labels->species[0]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_SPEC2,labels->species[1]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_SPEC3,labels->species[2]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN1,labels->channel[1]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN2,labels->channel[2]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN3,labels->channel[3]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN4,labels->channel[4]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN5,labels->channel[5]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN6,labels->channel[6]);
   GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN7,labels->channel[7]);
   RemovePopup(0);
   UpdateChannelPanel(labels,panel);
}

void EditFlowtubeParamters(struct FlowTubeParameters *flowtube,
      struct handles *panel,struct gas_input *channel) {
   int id,flag,i;
   flag=TRUE;

   InstallPopup(panel->hFlowtubeParams);
   while (flag) {
      GetPopupEvent(WAIT,&id);
      switch (id) {
         case FLOWTPARAM_FINISHED:
            flag = FALSE;
            break;
      }
   }
   /* Read in the new parameters */
   GetCtrlVal(panel->hFlowtubeParams,FLOWTPARAM_FLOWTEMP,&flowtube->temp);
   GetCtrlVal(panel->hFlowtubeParams,FLOWTPARAM_RADIUS,&flowtube->radius);
   GetCtrlVal(panel->hFlowtubeParams,FLOWTPARAM_CARRIERGAS,&flowtube-
>gas_type);
   switch(flowtube->gas_type) {
      case HE:
         flowtube->gas_correct = 1;
         flowtube->gas_mw      = 4;
         break;
      case NITROGEN:
         flowtube->gas_correct = 1.42;
         flowtube->gas_mw      = 28;
         break;
      case ARGON:
         flowtube->gas_correct = 1;
         flowtube->gas_mw      = 40;
         break;
   }
   /* Correct flowmeter ranges for the gases used */
   for(i=1;i<=7;i++) {
      if((channel[i].type == BUBBLER)||(channel[i].type == BULB)) {
         channel[i].range = channel[i].flowmeter.range *
            channel[i].flowmeter.gas_correct / flowtube->gas_correct;
      }
   }
   RemovePopup(0);
}


void    ChangeXYRange(short hPanel,short graph,short change_x,short change_y,
                         short xauto,double x_min,double x_max,short yauto,
                         double y_min,double y_max) {
   int hPopup,id,flag;
   hPopup = LoadPanel (RESOURCE_FILE, XYRANGE);
   if (hPopup < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return;
   }
   flag = TRUE;
   InstallPopup(hPopup);
   if(change_x) {
      SetCtrlVal(hPopup,XYRANGE_XMIN,x_min);
      SetCtrlVal(hPopup,XYRANGE_XMAX,x_max);
   }
   if(!change_x) {
      SetInputMode(hPopup,XYRANGE_XMIN,FALSE);
      SetInputMode(hPopup,XYRANGE_XMAX,FALSE);
   }
   if(change_y) {
      SetCtrlVal(hPopup,XYRANGE_YMIN,y_min);
      SetCtrlVal(hPopup,XYRANGE_YMAX,y_max);
   }
   if(!change_y) {
      SetInputMode(hPopup,XYRANGE_YMIN,FALSE);
      SetInputMode(hPopup,XYRANGE_YMAX,FALSE);
   }



   flag = TRUE;
   while(flag) {
      GetPopupEvent(WAIT,&id);
      switch (id) {
         case XYRANGE_XMIN:
            GetCtrlVal(hPopup,XYRANGE_XMIN,&x_min);
            break;
         case XYRANGE_XMAX:
            GetCtrlVal(hPopup,XYRANGE_XMAX,&x_max);
            break;
         case XYRANGE_YMIN:
            GetCtrlVal(hPopup,XYRANGE_YMIN,&y_min);
            break;
         case XYRANGE_YMAX:
            GetCtrlVal(hPopup,XYRANGE_YMAX,&y_max);
            break;
         case XYRANGE_FINISHED:
            flag = FALSE;
            break;
         default:
            break;
      }
   }
   RemovePopup(0);         /* Removes current popup */
   UnloadPanel(hPopup);
   SetAxisRange(hPanel,graph,xauto,x_min,x_max,yauto,y_min,y_max);
}
```

```
                                  /* CHART.C */
#define BOARD_RES 4096
#define MAX_VOLTAGE 10.0
#define NUMPOINTS 300
extern short AnalogBoardType, CounterOrAnalog, AnalogOutput, Slot, QuadRange;
extern short SamplesPerMS;
#include "b:\modular\struct.h"



short chart(struct gas_input *channel, struct handles *panel,
                  struct FlowTubeParameters *flowtube, struct Titles *labels)
{
   float *mass,*buffer;
   int dwell_time,delay_time,hMenuBar,hPanel,ylogon=FALSE;
   int position=0,i,j,start=0,handle,id,status,numpoints;
   char dirname[80],filename[80],*p;
   int xauto, yauto;
   double x_min,x_max,y_min,y_max;
   double partial_pressure[3],total_flow,tube_pressure,velocity;
   FILE *fp;
   /*----------------------------------------------------------------------*/
   /*  Load the menu bar from the resource file.  Use the constant assigned */
   /*  in the editor to refer to the menu bar.  The handle returned by      */
   /*  LoadMenuBar must be used to reference the menu bar in all subsequent */
   /*  function calls. If load was successful, the menu bar will be drawn   */
   /*  at the top of the screen.                                            */
   /*----------------------------------------------------------------------*/

   hMenuBar = LoadMenuBar (RESOURCE_FILE, CHARTMENU);
   if (hMenuBar < 0) {
      MessagePopup("Unable to load the required menu bar from the resource
file.");
      return 0;
   }

   /*----------------------------------------------------------------------*/
   /*  Load the panel from the resource file.  Use the constant assigned    */
   /*  in the editor to refer to the panel.  The handle returned by         */
   /*  LoadPanel must be used to reference the panel in all subsequent      */
   /*  function calls. If the panel handle is negative, the load failed,    */
   /*  so print a message and exit the program. Otherwise, display the      */
   /*  the panel.                                                           */
   /*----------------------------------------------------------------------*/

   hPanel = LoadPanel (RESOURCE_FILE, STRIPCHART);
   if (hPanel < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   DisplayPanel (hPanel);

   /*----------------------------------------------------------------------*/
   /* Dynamically allocate memory for the masses and the corresponding      */
   /* signals.  This memory will be freed upon leaving this panel.  This    */
   /* prevents the program from exceeding the limited memory available.     */
/* One floating point value will be allocated for each of the possible   */
/* outputs on the DAC board.                                             */
/*-----------------------------------------------------------------------*/

if((mass = (float *) malloc(5 * sizeof(float))) == NULL) {
   /* Insert a popup message to tell user problem */
   MessagePopup("Not enough memory for masses.");
   return 0;
}
if((buffer = (float *) malloc(5 * NUMPOINTS * sizeof(float))) == NULL) {
   /* Insert a popup message to tell user problem */
   MessagePopup("Not enough memory for buffer.");
   return 0;
}

/*-----------------------------------------------------------------------*/
/* Look for the file that has the values from the last time his panel    */
/* was activated.  If it doesn't exist, load variables with defaults.    */
/* If it does exist, read in values and restore panel states.            */
/*-----------------------------------------------------------------------*/

if ((fp = fopen("chart.prm","rb")) == NULL ) { /* file doesn't exist */
   /* Read in default variables */
   GetCtrlVal(hPanel,STRIPCHART_DWELL,&dwell_time);
   GetCtrlVal(hPanel,STRIPCHART_CHAN_DELAY,&delay_time);
   position=0;
   for(i=0;i<5;i++) mass[i] = 0;
   for(i=0;i<NUMPOINTS;i++) buffer[i] = 0;
}
else { /* file was found */
   /* read in masses */
   fread(mass,sizeof(float),5,fp);
   /* read in buffer */
   fread(buffer,sizeof(float),5*NUMPOINTS,fp);
   /* read in position */
   fread(&position,sizeof(int),1,fp);
   /* read in dwell_time */
   fread(&dwell_time,sizeof(int),1,fp);
   /* read in delay_time */
   fread(&delay_time,sizeof(int),1,fp);
   /* see if log scale is being used */
   fread(&ylogon,sizeof(int),1,fp);
   /* close file */
   fclose(fp);
   /* recall panel state */
   RecallPanelState(hPanel,"chart.pnl");
   /* Take care of menu bar state */
   if (ylogon==TRUE) {
      SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGON,TRUE);
      SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGOFF,FALSE);
      SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGON,FALSE);
      SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGOFF,TRUE);
   }
   else {
      SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGON,FALSE);
      SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGOFF,TRUE);
      SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGON,TRUE);
      SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGOFF,FALSE);
```

```
        }
    }

    GetProgramDir(dirname);
    /*===================== MAIN LOOP ===================================*/

    while (TRUE) {
    /*------------------------------------------------------------------*/
    /*  Call GetUserEvent with the wait parameter set to FALSE.  This will */
    /*  cause the function not to wait for an event.  When an event occurs,*/
    /*  the handle variable will either match the menu bar handle or the   */
    /*  panel handle.  The id variable will match one of the menu bar or   */
    /*  control ID constants assigned in the editor.                       */
    /*------------------------------------------------------------------*/
       GetUserEvent (NO_WAIT, &handle, &id);
       /*------------------------------------------------------------------
*/
       /*  If the handle matches the menu bar handle, decode the id variable
*/
       /*  to figure out which menu item or immediate command was selected.
*/
       /*------------------------------------------------------------------
*/
       if (handle == hMenuBar) {
          switch (id) {
             case CHARTMENU_FILE_SAVEDATA : /* Save CHART */

                 status=FileSelectPopup(dirname,"*.dat","Select file to save
data.",NO_RESTRICT,NO_RESTRICT,TRUE,filename);
                 if(status) {
                    if(status==FILE_EXISTS) {
                       status=ConfirmPopup("That file already exits.
Overwrite?");
                    }
                    if(status) {
                       fp = fopen(filename,"wt");
                          /* WRITE DATA TO FILE */
                          fprintf(fp,"Delay = %d (msec)\t Dwell = %d
(msec)\n",

                             delay_time,dwell_time);
                          for(i=0;i<5;i++) {
                             fprintf(fp,"Mass %g\t",mass[i]);
                          }
                          fprintf(fp,"\n");
                          for(i=position;i<5*NUMPOINTS;i+=5) {
                             for(j=0;(j<5) && ((i+j) < 5*NUMPOINTS);j++) {
                                fprintf(fp,"%g\t",buffer[i+j]);
                             }
                             fprintf(fp,"\n");
                          }
                          for(i=0;i<position;i+=5) {
                             for(j=0;(j<5) && ((i+j) < position);j++) {
                                fprintf(fp,"%g\t",buffer[i+j]);
                             }
                             fprintf(fp,"\n");
                          }
                          fclose(fp);
                          /* save dir for future use */
```

```
                        p=strrchr(filename,'\\');
                        if(p!=NULL) {
                           *p=0;
                           strcpy(dirname,filename);
                        }
                    }
                 }
              break;
           case CHARTMENU_FILE_SAVESETUP :  /* Save flows setup */
              SaveSetupParameters(channel,flowtube,labels);
              break;
           case CHARTMENU_FILE_PRINT : /* Print out Chart */
              OutputGraph(0,panel->PrintSetup.OutputFile,
                                 panel-
>PrintSetup.ScaleFactor,hPanel,STRIPCHART_CHART);
              break;
           case CHARTMENU_FILE_PRINTSETUP :
              PrintSetup(panel);
              break;
           case CHARTMENU_SETUP_CHAN :
              SetupChannels(channel,labels,panel,flowtube);
              /* Update the screen */
              UpdateMainCtrls(channel,labels,panel);
              break;
           case CHARTMENU_SETUP_FLOWTUBPAR:
              EditFlowtubeParamters(flowtube,panel,channel);
              /* Update the screen */
              UpdateMainCtrls(channel,labels,panel);
              break;
           case CHARTMENU_SETUP_YLOGON:
              SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGON,TRUE);
              SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGOFF,FALSE);
              SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGON,FALSE);
              SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGOFF,TRUE);
              /* Rescale Y axis, 21 refers to y log axis variable     */
              SetGraphAttribute(hPanel,STRIPCHART_CHART,21,TRUE);
              ylogon = TRUE;
              break;
           case CHARTMENU_SETUP_YLOGOFF:
              SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGON,FALSE);
              SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGOFF,TRUE);
              SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGON,TRUE);
              SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGOFF,FALSE);
              /* Rescale Y axis, 21 refers to y log axis variable     */
              SetGraphAttribute(hPanel,STRIPCHART_CHART,21,FALSE);
              ylogon = FALSE;
              break;
           case CHARTMENU_SETUP_CHANGEXY:

   GetAxisRange(hPanel,STRIPCHART_CHART,&xauto,&x_min,&x_max,&yauto,&y_min,
                                    &y_max);

   ChangeXYRange(hPanel,STRIPCHART_CHART,0,1,xauto,x_min,x_max,yauto,y_min,
                                    y_max);
              break;
           case CHARTMENU_SWITCH_KINETICS:
           case CHARTMENU_SWITCH_MASS_SPEC:
           case CHARTMENU_FILE_QUIT :
```

```
         /* SAVE STATES */
         if ((fp = fopen("chart.prm","wb")) == NULL ) { /* error */
            /* Tell user there was an error. */
            MessagePopup("File error- can't save data.");
         }
         else { /* okay */
            /* save masses */
            fwrite(mass,sizeof(float),5,fp);
            /* save buffer */
            fwrite(buffer,sizeof(float),5*NUMPOINTS,fp);
            /* save position */
            fwrite(&position,sizeof(short),1,fp);
            /* write dwell_time */
            fwrite(&dwell_time,sizeof(short),1,fp);
            /* write delay_time */
            fwrite(&delay_time,sizeof(short),1,fp);
            /* record if log scale is being used */
            fwrite(&ylogon,sizeof(int),1,fp);
            /* close file */
            fclose(fp);
            /* save panel state */
            SavePanelState(hPanel,"chart.pnl");
         }
         /* free arrays */
         free(mass);
         free(buffer);
         /* Unload menubar and panel */
         UnloadMenuBar();
         UnloadPanel(hPanel);
         /* return proper value */
         if(id == CHARTMENU_SWITCH_KINETICS)      return KINETICS_PANEL;
         else if(id==CHARTMENU_SWITCH_MASS_SPEC) return MASSSPEC_PANEL;
         else if(id==CHARTMENU_FILE_QUIT)         return QUIT;
         break;
      default :
         break;
   }
}
else if(handle == hPanel) {
   switch (id) {
      case STRIPCHART_START :
         /* Set flag to start */
         start = 1;
         break;
      case STRIPCHART_STOP :
         /* Clear start flag */
         start = 0;
         break;
      case STRIPCHART_CLEAR :
         /* delete graph and clear buffer */
         position = 0;
         ClearStripChart(hPanel,STRIPCHART_CHART);
         break;
      case STRIPCHART_DWELL :
         /* get new dwell time */
         GetCtrlVal(hPanel,STRIPCHART_DWELL,&dwell_time);
         break;
      case STRIPCHART_CHAN_DELAY :
          /* Get the new delay between channels                    */
          GetCtrlVal(hPanel,STRIPCHART_CHAN_DELAY,&delay_time);
          break;
       case STRIPCHART_MASS1 :
          /* the new mass for mass[0] */
          GetCtrlVal(hPanel,STRIPCHART_MASS1,&mass[0]);
          break;
       case STRIPCHART_MASS2 :
          /* the new mass for mass[1] */
          GetCtrlVal(hPanel,STRIPCHART_MASS2,&mass[1]);
          break;
       case STRIPCHART_MASS3 :
          /* the new mass for mass[2] */
          GetCtrlVal(hPanel,STRIPCHART_MASS3,&mass[2]);
          break;
       case STRIPCHART_MASS4 :
          /* the new mass for mass[3] */
          GetCtrlVal(hPanel,STRIPCHART_MASS4,&mass[3]);
          break;
       case STRIPCHART_MASS5 :
          /* the new mass for mass[4] */
          GetCtrlVal(hPanel,STRIPCHART_MASS5,&mass[4]);
          break;
       default :
          break;
    }
 }


 ReadFlows(channel,panel,flowtube,partial_pressure,&total_flow,
                &tube_pressure,&velocity);

 if(start) { /* get points */
    if(position == 5*NUMPOINTS) position = 0;
    for(i=0;i<5;i++) {
       /* see that mass is non zero */
       if(mass[i]!=0.0) {
          /* Set mass spec to proper mass */
          WriteVoltage(mass[i]*10.0/(float)QuadRange);
          /* Delay appropriate amount between channels */
          delay(delay_time);
          /* Get data */
          buffer[position+i] = GetCounts(dwell_time);
       }
       else buffer[position + i] = 0;
    }
    /* plot points */
    PlotStripChart(hPanel,STRIPCHART_CHART,buffer,5,position,0,3);
    position += 5;
 }

}

}
```

```
                                /* KINETIC.C */
/*===========================================================================*/
/*      LabWindows replacement program for "Jayne-Ware" kinetics program.
                              by Darryl D. Spencer
                                 April 19, 1994
*/
/*===========================================================================*/

/*===========================================================================*/
/*    This program controlls the Extrell Mass Spectrometer/Flowtube setup
        used by Darryl Spencer and Roger Meads for the Alumina experiment.
        The computer used a National Instruments Lab-PC board in a 286 to read
        the voltage signals from the various mass flowmeters, baratrons, and
        the lock-in amplifier.  An output signal controlled the mass the Extrel
        stayed on.

        The screen had an XY Graph that showed the signal as a function of
        injector distance. Point could be taken or erased and a line fitted.
        A graph could also be printed.  Other data from the flowmeters and
        baratrons was continually updated on the screen.

        This Program seeks to do similar things in a similar way but different
        enough to take advantage of LabWindow's capabilities and my own
        preferences.                                                         */
/*===========================================================================*/

/*=== Includes ==============================================================*/

#include "b:\modular\struct.h"

/*=== Global Variables for Hardware ========================================*/

extern short AnalogBoardType, CounterOrAnalog, AnalogOutput, Slot, QuadRange;
extern short SamplesPerMS;

/*=== Prototypes for functions used only in this file =====================*/
void GetPoint(int numsamples,int dwell,struct sample *point);
void ChangeMass(double *mass);

/*=== Start of Kinetic Function ===========================================*/

short kinetic(struct gas_input *channel, struct handles *panel,
                  struct FlowTubeParameters *flowtube, struct Titles *labels)
{

   /* File access variables */
   char dirname[68],filename[80],*p;
   FILE *fp;

   /* Needed Handles */
   int hMenuBar,hPanel,handle;
   /* Control Variables */
   int id,control,val;
   int xauto, yauto;
   double x,y,x2,y2;
   double x_min,x_max,y_min,y_max;

   /* Data Acquisition Variables */
   struct PlotParameters plot;
   struct sample point;
   int x_pos,current_plot_hdl;
   double numerator[3],partial_pressure[3],flow,total_flow;
   double velocity,tube_pressure, pressure,time,sigma;
   double mass,signal,reynolds;
   double sum_x,sum_xsq,mean,variance;


   /* Error, Indexes, and Flag Variables */
   int status,i,j;
   struct time t1,t2;

   /* Reset Important Variables */
   plot.numpoints=0;
   plot.plotstyle=3;
   plot.plotcolor=15;
   current_plot_hdl=1;
   time = 0;
   velocity = 1000;
   x_pos=0;

   /*----------------------------------------------------------------------*/
   /*  Load the menu bar from the resource file.  Use the constant assigned */
   /*  in the editor to refer to the menu bar.  The handle returned by      */
   /*  LoadMenuBar must be used to reference the menu bar in all subsequent */
   /*  function calls. If load was successful, the menu bar will be drawn   */
   /*  at the top of the screen.                                            */
   /*----------------------------------------------------------------------*/

   hMenuBar = LoadMenuBar (RESOURCE_FILE, KINMENU);
   if (hMenuBar < 0) {
      MessagePopup("Unable to load the required menu bar from the resource
file.");
      return 0;
   }

   /*----------------------------------------------------------------------*/
   /*  Load the panel from the resource file.  Use the constant assigned    */
   /*  in the editor to refer to the panel.  The handle returned by         */
   /*  LoadPanel must be used to reference the panel in all subsequent      */
   /*  function calls. If the panel handle is negative, the load failed,    */
   /*  so print a message and exit the program. Otherwise, display the      */
   /*  the panel.                                                           */
   /*----------------------------------------------------------------------*/

   hPanel = LoadPanel (RESOURCE_FILE, KINETICS);
   if (hPanel < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   DisplayPanel (hPanel);
   /*----------------------------------------------------------------------*/
   /* Look for the file that has the values from the last time his panel    */
   /* was activated.  If it doesn't exist, load variables with defaults.    */
   /* If it does exist, read in values and restore panel states.            */
   /*-----------------------------------------------------------------------*/

   if ((fp = fopen("kinetic.prm","rb")) == NULL ) { /* file doesn't exist */
      /* Read in default variables */

   }
   else { /* file was found */
      /* read in plot info */
      fread(&plot,sizeof(plot),1,fp);
      fread(&x_pos,sizeof(int),1,fp);
      fread(&mass,sizeof(double),1,fp);
      fread(&current_plot_hdl,sizeof(int),1,fp);

      /* close file */
      fclose(fp);
      /* recall panel state */
      RecallPanelState(hPanel,"kinetic.pnl");
   }

   GetProgramDir(dirname);
   /*=================MAIN PROGRAM LOOP ===================================*/

   while (TRUE) {
   /*---------------------------------------------------------------------*/
   /*  Call GetUserEvent with the wait parameter set to FALSE.  This will */
   /*  cause the function not to wait for an event.  When an event occurs,*/
   /*  the handle variable will either match the menu bar handle or the   */
   /*  panel handle.  The id variable will match one of the menu bar or   */
   /*  control ID constants assigned in the editor.                       */
   /*---------------------------------------------------------------------*/
      GetUserEvent (NO_WAIT, &handle, &id);
      /*------------------------------------------------------------------
*/
      /*  If the handle matches the menu bar handle, decode the id variable
*/
      /*  to figure out which menu item or immediate command was selected.
*/
      /*------------------------------------------------------------------
*/
      if (handle == hMenuBar) {
         switch (id) {
            case KINMENU_FILE_SAVEDATA :

                  status=FileSelectPopup(dirname,"*.dat","Select file to save
data.",NO_RESTRICT,NO_RESTRICT,TRUE,filename);
                  if(status) {
                     if(status==FILE_EXISTS) {
                        status=ConfirmPopup("That file already exits.
Overwrite?");
                     }
                     if(status) {
                        fp = fopen(filename,"wt");
                        fprintf(fp,"Velocity\tTube Pressure
(torr)\tTemperature (K)\n");

   fprintf(fp,"%g\t%g\t%g\n",velocity,tube_pressure,flowtube->temp);
                        fprintf(fp,"Slope\tSlope Stdev\n");
                        fprintf(fp,"%g\t%g\n",plot.slope,plot.sigma_slope);
                        fprintf(fp,"Intercept\tIntercept Stdev\n");

   fprintf(fp,"%g\t%g\n",plot.intercept,plot.sigma_intercept);
                        fprintf(fp,"Flowmeter voltages");
                        for(i=1;i<=7;i++) {
                           fprintf(fp,"%g\t",channel[i].voltage);
                        }
                        fprintf(fp,"\n");
                        fprintf(fp,"Time\tSignal\tStdev.\n");
                        for(i=0;i<plot.numpoints;i++) {

   fprintf(fp,"%g\t%g\t%g\n",plot.x[i],plot.y[i],plot.y_sigma[i]);
                        }
                        /* print out raw information */
                        fclose(fp);
                        /* save dir for future use */
                        p=strrchr(filename,'\\');
                        if(p!=NULL) {
                           *p=0;
                           strcpy(dirname,filename);
                        }
                     }
                  }
               break;
            case KINMENU_FILE_SAVESETUP :
               SaveSetupParameters(channel,flowtube,labels);
               break;
            case KINMENU_FILE_PRINT : /* Print out Graph */
               OutputGraph(0,panel->PrintSetup.OutputFile,
                                    panel-
>PrintSetup.ScaleFactor,hPanel,STRIPCHART_CHART);
               break;
            case KINMENU_FILE_PRINTSETUP :
               PrintSetup(panel);
               break;
            case KINMENU_ACQUIRE_GETPOINT :
               /* Average 32 points with a .1 second dwell time */
               beep();
               x=0;
               sum_x=0;
               sum_xsq=0;

               for(i=0;i<32;i++) {
                  x=GetCounts(100);
                  sum_x   += x;
                  sum_xsq += pow(x,2);
               }
               mean = sum_x/(double)i;
               variance = (sum_xsq - (pow(sum_x,2)/i)) / (i-1);
               variance = fabs(variance);
               beep();
               plot.x[plot.numpoints] = time;
               plot.y[plot.numpoints]=mean;
               sigma = sqrt(variance);
               plot.y_sigma[plot.numpoints]=sigma;
               /* Plot point on the screen */
               current_plot_hdl = PlotPoint(hPanel,KINETICS_GRAPH,
   time,mean,plot.plotstyle,plot.plotcolor);
               plot.plot_hdl[plot.numpoints] = current_plot_hdl;
               /* Plot error bar on the screen */
               current_plot_hdl=PlotLine(hPanel,KINETICS_GRAPH,
                              time,mean + sigma,time,mean -
sigma,plot.plotcolor);
               plot.err_hdl[plot.numpoints] = current_plot_hdl;
               /* Increase point count */
               plot.numpoints++;
               break;
            case KINMENU_ACQUIRE_DELPOINT :
               if(plot.numpoints>0) {
                  /* Delete Point */
                  DeleteGraphPlot(hPanel,KINETICS_GRAPH,
                                    plot.plot_hdl[plot.numpoints-1],0);
                  /* Delete error bar */
                  DeleteGraphPlot(hPanel,KINETICS_GRAPH,
                                    plot.err_hdl[plot.numpoints-1],1);
                  /* Decrease Point Count */
                  plot.numpoints--;
                  beep();
               }
               break;
            case KINMENU_ACQUIRE_INCREASEX:
               x_pos++;
               SetCtrlVal(hPanel,KINETICS_INJECTOR_POSITION,x_pos);
               break;
            case KINMENU_ACQUIRE_DECREASEX:
               x_pos--;
               SetCtrlVal(hPanel,KINETICS_INJECTOR_POSITION,x_pos);
               break;
            case KINMENU_ACQUIRE_RESETX:
               x_pos = 0;
               SetCtrlVal(hPanel,KINETICS_INJECTOR_POSITION,x_pos);
               break;
            case KINMENU_ACQUIRE_LINFIT :
               /* Call least squares fit routine. */
               if(plot.numpoints < 3) {
                  MessagePopup("Need at least 3 points for a fit!");
                  break;
               }
               wlsf(&plot);
               x = plot.x[0];
               y = exp( x*plot.slope + plot.intercept );
               x2= plot.x[ (plot.numpoints) - 1 ];
               y2= exp( x2*plot.slope + plot.intercept );
               PlotLine(hPanel,KINETICS_GRAPH,x,y,x2,y2,15);
               beep();
               SetCtrlVal(hPanel,KINETICS_SLOPE,plot.slope);
               SetCtrlVal(hPanel,KINETICS_INTERCEPT,plot.intercept);
               SetCtrlVal(hPanel,KINETICS_SLOPE_ERR,sqrt(plot.sigma_slope));

    SetCtrlVal(hPanel,KINETICS_INTERCEPT_ERR,sqrt(plot.sigma_intercept));
               break;
            case KINMENU_ACQUIRE_DELGRAPH :
               if(!ConfirmPopup("Are you sure you want to delete this?"))
break;
```

```
            DeleteGraphPlot(hPanel,KINETICS_GRAPH,-1,1);
            plot.numpoints=0;
            break;
        case KINMENU_ACQUIRE_CHANGEMASS :
            /* Put in code for popup mass menu. */
            ChangeMass(&mass);
            break;
        case KINMENU_SETUP_CHAN :
            SetupChannels(channel,labels,panel,flowtube);
            /* Update the screen */
            UpdateMainCtrls(channel,labels,panel);
            break;
        case KINMENU_SETUP_FLOWTUBPAR:
            EditFlowtubeParamters(flowtube,panel,channel);
            /* Update the screen */
            UpdateMainCtrls(channel,labels,panel);
            break;
        case KINMENU_SETUP_YAUTON:
            /* turn y autoscale on */

GetAxisRange(hPanel,KINETICS_GRAPH,&xauto,&x_min,&x_max,&yauto,&y_min,
                                &y_max);

SetAxisRange(hPanel,KINETICS_GRAPH,xauto,x_min,x_max,1,y_min,y_max);
            /* Disable Y autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTON,6,0);
            /* Check Y autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTON,7,1);
            /* Enable Y autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTOFF,6,1);
            /* Uncheck Y autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTOFF,7,0);
            break;
        case KINMENU_SETUP_YAUTOFF:
            /* turn y autoscale off */

GetAxisRange(hPanel,KINETICS_GRAPH,&xauto,&x_min,&x_max,&yauto,&y_min,
                                &y_max);

SetAxisRange(hPanel,KINETICS_GRAPH,xauto,x_min,x_max,2,y_min,y_max);
            /* Enable Y autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTON,6,1);
            /* Uncheck Y autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTON,7,0);
            /* Disable Y autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTOFF,6,0);
            /* Check Y autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_YAUTOFF,7,1);
            break;
        case KINMENU_SETUP_XAUTON:
            /* turn x autoscale on */

GetAxisRange(hPanel,KINETICS_GRAPH,&xauto,&x_min,&x_max,&yauto,&y_min,
                                &y_max);

SetAxisRange(hPanel,KINETICS_GRAPH,1,x_min,x_max,yauto,y_min,y_max);
            /* Disable x autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTON,6,0);
            /* Check x autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTON,7,1);
            /* Enable x autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTOFF,6,1);
            /* Uncheck x autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTOFF,7,0);
            break;
         case KINMENU_SETUP_XAUTOFF:
            /* turn x autoscale off */

GetAxisRange(hPanel,KINETICS_GRAPH,&xauto,&x_min,&x_max,&yauto,&y_min,
                                &y_max);

SetAxisRange(hPanel,KINETICS_GRAPH,2,x_min,x_max,yauto,y_min,y_max);
            /* Enable x autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTON,6,1);
            /* Uncheck x autoscale on choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTON,7,0);
            /* Disable x autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTOFF,6,0);
            /* Check x autoscale off choice */
            SetMenuBarAttribute(KINMENU_SETUP_XAUTOFF,7,1);
            break;
         case KINMENU_SETUP_CHANGEXY:

GetAxisRange(hPanel,KINETICS_GRAPH,&xauto,&x_min,&x_max,&yauto,&y_min,
                                &y_max);

ChangeXYRange(hPanel,KINETICS_GRAPH,1,1,xauto,x_min,x_max,yauto,y_min,
                                y_max);
            break;
         case KINMENU_SWITCH_CHART:
         case KINMENU_SWITCH_MASS_SPEC:
         case KINMENU_FILE_QUIT :
            /* SAVE STATES */
            if ((fp = fopen("kinetic.prm","wb")) == NULL ) { /* error */
               /* Tell user there was an error. */
               MessagePopup("File error- can't save data.");
            }
            else { /* okay */
               /* read in plot info */
               fwrite(&plot,sizeof(plot),1,fp);
               fwrite(&x_pos,sizeof(int),1,fp);
               fwrite(&mass,sizeof(double),1,fp);
               fwrite(&current_plot_hdl,sizeof(int),1,fp);
               /* close file */
               fclose(fp);
               /* save panel state */
               SavePanelState(hPanel,"kinetic.pnl");
            }
            /* Unload menubar and panel */
            UnloadMenuBar();
            UnloadPanel(hPanel);
            /* return proper value */
            if(id == KINMENU_SWITCH_CHART)        return STRIPCHART_PANEL;
            else if(id==KINMENU_SWITCH_MASS_SPEC) return MASSSPEC_PANEL;
            else if(id==KINMENU_FILE_QUIT)        return QUIT;
            break;
            default :
               break;
         }
      }
      else if(handle == hPanel) {
         switch (id) {
            case KINETICS_INJECTOR_POSITION:
               GetCtrlVal(hPanel,KINETICS_INJECTOR_POSITION,&x_pos);
               break;
            default:
               break;
         }
      }

      /* ======== Calculate variables and put them on the screen ==== */
      ReadFlows(channel,panel,flowtube,partial_pressure,&total_flow,
                     &tube_pressure,&velocity);
      /* Calculate time in seconds */
      time = ((float)x_pos)/(velocity+TINY);
      SetCtrlVal(hPanel,KINETICS_TIME,time);

      /*-------------------- Show current plot postion on screen ------------
*/

      signal = GetCounts(1) + TINY;
      SetCtrlVal(hPanel,KINETICS_SIGNAL,signal);

      /* current_plot_hdl =
         PlotPoint(hPanel,KINETICS_GRAPH,time,signal,2,plot.plotcolor); */
      /* Alternative code to move a cursor to signal site. */
      SetGraphCursor(hPanel,KINETICS_GRAPH,1,time,signal);

      /*------------ Delete current plot position on screen ----------*/
      /* DeleteGraphPlot(hPanel,KINETICS_GRAPH,current_plot_hdl,1); */


   }

}


/*=========================================================================
==*/
/* Functions to Change or Edit Various Parameters.  Functions are provided to
*/
/* change basic experimental constants such as the mass selected. Others are
*/
/* used to reflect a change in hardware configuration in the flows.
/*=========================================================================
==*/

/*------------- Routine to set the mass spectrometer to a new mass ----------
--*/

void ChangeMass(double *mass) {
   int flag,control,hMassPanel;
   double signal;
   hMassPanel = LoadPanel (RESOURCE_FILE, SELECTMASS);
   if (hMassPanel < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return;
   }

   InstallPopup(hMassPanel);
   flag = TRUE;
   while(flag) {
      GetPopupEvent(NO_WAIT,&control);
      switch(control) {
         case SELECTMASS_MASSVALUE:
            /* Get new mass selected */
            GetCtrlVal(hMassPanel,SELECTMASS_MASSVALUE,mass);
            /* Insert code to Set Mass Spec to mass chosen */
            WriteVoltage((*mass)*10.0/(float)QuadRange);
            break;
         case SELECTMASS_INCRBY1:
            *mass += 1.0;
            SetCtrlVal(hMassPanel,SELECTMASS_MASSVALUE,*mass);
            /* Insert code to Set Mass Spec to mass chosen */
            WriteVoltage((*mass)*10.0/(float)QuadRange);
            break;
         case SELECTMASS_INCRBYP1:
            *mass +=  0.1;
            SetCtrlVal(hMassPanel,SELECTMASS_MASSVALUE,*mass);
            /* Insert code to Set Mass Spec to mass chosen */
            WriteVoltage((*mass)*10.0/(float)QuadRange);
            break;
         case SELECTMASS_DECRBY1:
            *mass -= 1.0;
            SetCtrlVal(hMassPanel,SELECTMASS_MASSVALUE,*mass);
            /* Insert code to Set Mass Spec to mass chosen */
            WriteVoltage((*mass)*10.0/(float)QuadRange);
            break;
         case SELECTMASS_DECRBYP1:
            *mass -= 0.1;
            SetCtrlVal(hMassPanel,SELECTMASS_MASSVALUE,*mass);
            /* Insert code to Set Mass Spec to mass chosen */
            WriteVoltage((*mass)*10.0/(float)QuadRange);
            break;
         case SELECTMASS_FINISHED:
            flag = FALSE;
            break;
      }
      /* Insert code to get signal from spectrometer */
      signal = GetCounts(100);
      /* Put it on screen */
      SetCtrlVal(hMassPanel,SELECTMASS_MASSIGNAL,signal);
   }
   RemovePopup(0);
}


/*==============================================================
==*/
```

```
/*                  Routines for acquiring and processing data
*/
/*=====================================================================
==*/

/*------ GetPoint acquires and averages points and calculates the variance ---
--*/
void GetPoint(int numsamples,int dwell,struct sample *point) {
   double sum_x,sum_xsq;
   float x;
   int i;
   x=0;
   sum_x=0;
   sum_xsq=0;

   for(i=0;i<numsamples;i++) {
      x=GetCounts(dwell);
      sum_x   += x;
      sum_xsq += (x*x);
      /* Insert Code to Update Screen Readout */
   }
   point->mean = sum_x/(double)i;
   point->variance = (sum_xsq - (sum_x*sum_x)/((double)i) ) / ((double)(i-1));
   point->variance = fabs(point->variance);
}

/*--------------- Weighted least squares fit for plotted data ----------------
-*/

void wlsf(struct PlotParameters *plot) {
   double sum_wx=0,sum_wxsq=0,sum_wy=0,sum_wysq=0,sum_wxy=0,sum_weight=0;
   double weight,x,y,D,slope,intercept,var_slope,var_intercept;
   double ssew=0,x2,y2;
   int i;

   for(i=0;i<plot->numpoints;i++) {
      x = plot->x[i];
      y = log(plot->y[i]);
      weight = (plot->y[i]*plot->y[i])/pow(plot->y_sigma[i],2);
      sum_wx += x*weight;
      sum_wy += y*weight;
      sum_wxsq += x*x*weight;
      sum_wysq += y*y*weight;
      sum_wxy += x*y*weight;
      sum_weight += weight;
   }

   D = (sum_weight*sum_wxsq - sum_wx*sum_wx);
   slope = ((sum_weight * sum_wxy) - (sum_wx * sum_wy)) / D;
   intercept = ((sum_wy*sum_wxsq)-(sum_wxy*sum_wx)) / D;

   /* Calculate errors */

   ssew = sum_wysq - slope*sum_wxy - intercept*sum_wy;

   plot->slope = slope;
   plot->intercept = intercept;
   var_slope = (ssew*sum_weight)/((plot->numpoints - 2) * fabs(D));
   var_intercept = (ssew*sum_wxsq)/((plot->numpoints - 2) * fabs(D));
   plot->sigma_slope = sqrt(var_slope);
   plot->sigma_intercept = sqrt(var_intercept);
}
```

```
                                        /* MASSSPEC.C */
#define  PLOT_COLOR  9
#define  MAX_ZOOM    5

#define  INITX1      0
#define  INITY1      0
#define  INITY2      12

#define  TRUE        1
#define  FALSE       0

#define BOARD_RES 4096
#define MAX_VOLTAGE 10.0

#include "b:\modular\struct.h"

extern short QuadRange;

short mass_spec(struct gas_input *channel, struct handles *panel,
                        struct FlowTubeParameters *flowtube, struct Titles
*labels) {
   float *masses,*signals;
   float start_mass,end_mass,sweep_rate,mass;
   double c1x,c1y,c2x,c2y,dx,dy,c3x,c3y,tmp;
   int i,j,start_index,end_index,pause,dwell,handle2,id2;
   int xauto, yauto,status,ylogon=FALSE;
   char filename[40],dirname[80],junk[80],*p;
   double x_min,x_max,y_min,y_max;
   double partial_pressure[3],total_flow,tube_pressure,velocity;
   FILE *fp;

   int hPanel,hMenuBar,handle,id,zoomlevel;

   float tab1x[MAX_ZOOM],tab1y[MAX_ZOOM],tab2x[MAX_ZOOM],tab2y[MAX_ZOOM];

   /*------------------------------------------------------------------------*/
   /*  Load the menu bar from the resource file.  Use the constant assigned */
   /*  in the editor to refer to the menu bar.  The handle returned by      */
   /*  LoadMenuBar must be used to reference the menu bar in all subsequent */
   /*  function calls. If load was successful, the menu bar will be drawn   */
   /*  at the top of the screen.                                            */
   /*------------------------------------------------------------------------*/

   hMenuBar = LoadMenuBar (RESOURCE_FILE, MASSMENU);
   if (hMenuBar < 0) {
      MessagePopup("Unable to load the required menu bar from the resource
file.");
      return 0;
   }

   /*------------------------------------------------------------------------*/
   /*  Load the panel from the resource file.  Use the constant assigned    */
   /*  in the editor to refer to the panel.  The handle returned by         */
   /*  LoadPanel must be used to reference the panel in all subsequent      */
   /*  function calls. If the panel handle is negative, the load failed,    */
   /*  so print a message and exit the program. Otherwise, display the      */
   /*  the panel.                                                           */
   /*------------------------------------------------------------------------*/
   hPanel = LoadPanel (RESOURCE_FILE, MASSSPEC);
   if (hPanel < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   DisplayPanel (hPanel);

   /*------------------------------------------------------------------------*/
   /* Dynamically allocate memory for the masses and the corresponding      */
   /* signals.  This memory will be freed upon leaving this panel.  This    */
   /* prevents the program from exceeding the limited memory available.     */
   /* One floating point value will be allocated for each of the possible   */
   /* outputs on the DAC board.                                              */
   /*------------------------------------------------------------------------*/

   if((masses = (float *) malloc(BOARD_RES * sizeof(float))) == NULL) {
      /* Insert a popup message to tell user problem */
      MessagePopup("Not enough memory for masses array.");
      exit(0);
   }
   if((signals = (float *) malloc(BOARD_RES * sizeof(float))) == NULL) {
      /* Insert a popup message to tell user problem */
      MessagePopup("Not enough memory for signals array.");
      exit(0);
   }

   /*------------------------------------------------------------------------*/
   /* Look for the file that has the values from the last time his panel    */
   /* was activated.  If it doesn't exist, load variables with defaults.    */
   /* If it does exist, read in values and restore panel states.            */
   /*------------------------------------------------------------------------*/

   if ((fp = fopen("massspec.prm","rb")) == NULL ) { /* file doesn't exist */
      /* Read in default variables */
      GetCtrlVal(hPanel,MASSSPEC_SWEEPRATE,&sweep_rate);
      GetCtrlVal(hPanel,MASSSPEC_STARTMASS,&start_mass);
      GetCtrlVal(hPanel,MASSSPEC_ENDMASS,&end_mass);
   }
   else { /* file was found */
      /* read in masses */
      fread(masses,sizeof(float),BOARD_RES,fp);
      /* read in signals */
      fread(signals,sizeof(float),BOARD_RES,fp);
      /* read in sweep_rate */
      fread(&sweep_rate,sizeof(float),1,fp);
      /* read in start_mass */
      fread(&start_mass,sizeof(float),1,fp);
      /* read in end_mass */
      fread(&end_mass,sizeof(float),1,fp);
      /* see if log scale is being used */
      fread(&ylogon,sizeof(int),1,fp);
      /* close file */
      fclose(fp);
      /* recall panel state */
      RecallPanelState(hPanel,"massspec.pnl");
      /* Take care of menu bar state */
      if (ylogon==TRUE) {
         SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGON,TRUE);
         SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGOFF,FALSE);
         SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGON,FALSE);
         SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGOFF,TRUE);
      }
      else {
         SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGON,FALSE);
         SetMenuItemCheckmark(CHARTMENU_SETUP_YLOGOFF,TRUE);
         SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGON,TRUE);
         SetInputMode(hMenuBar,CHARTMENU_SETUP_YLOGOFF,FALSE);
      }
   }

   GetProgramDir(dirname);

   /*===================== MAIN LOOP ===================================*/

   while (TRUE) {
   /*------------------------------------------------------------------*/
   /*  Call GetUserEvent with the wait parameter set to FALSE.  This will */
   /*  cause the function not to wait for an event.  When an event occurs,*/
   /*  the handle variable will either match the menu bar handle or the   */
   /*  panel handle.  The id variable will match one of the menu bar or   */
   /*  control ID constants assigned in the editor.                       */
   /*------------------------------------------------------------------*/
      GetUserEvent (NO_WAIT, &handle, &id);
      /*------------------------------------------------------------------
*/
      /*  If the handle matches the menu bar handle, decode the id variable
*/
      /*  to figure out which menu item or immediate command was selected.
*/
      /*------------------------------------------------------------------
*/
      if (handle == hMenuBar) {
         switch (id) {
            case MASSMENU_FILE_SAVEDATA : /* Save Mass Spectrum */

               status=FileSelectPopup(dirname,"*.dat","Select file to save
data.",NO_RESTRICT,NO_RESTRICT,TRUE,filename);
               if(status) {
                  if(status==FILE_EXISTS) {
                     status=ConfirmPopup("That file already exits.
Overwrite?");
                  }
                  if(status) {
                     fp = fopen(filename,"wt");
                     /* Input file wil now say */
                     /* "sweep rate\tstart mass\tend mass\tstart index\tend
index\n"*/
                     /* Now write that line */
                     fprintf(fp,"sweep rate\tstart mass\tend mass\tstart
index\tend index\n");
                     /* Now write important parameters */

   fprintf(fp,"%g\t%g\t%g\t%d\t%d\n",sweep_rate,start_mass,end_mass,
                                   start_index,end_index);
                        /* Input file will now say "Mass\tSignal\n" */
                        /* Now write in that line */
                        fprintf(fp,"Mass\tSignal\n");
                        /* Now write masses and signals */
                        for(i=start_index;i<=end_index;i++) {
                           fprintf(fp,"%g\t%g\n",masses[i],signals[i]);
                        }
                        fclose(fp);
                        /* save dir for future use */
                        p=strrchr(filename,'\\');
                        if(p!=NULL) {
                           *p=0;
                           strcpy(dirname,filename);
                        }
                     }
                  }
                  break;
               case MASSMENU_FILE_SAVESETUP :  /* Save flows setup */
                  SaveSetupParameters(channel,flowtube,labels);
                  break;
               case MASSMENU_FILE_GETOLDSPEC:
                  GetProgramDir(dirname);
                  status=FileSelectPopup(dirname,"*.dat","Select file to save
data.",NO_RESTRICT,NO_RESTRICT,TRUE,filename);
                  if(status) {
                     if(status==FILE_EXISTS) {
                        fp = fopen(filename,"rt");
                        /* Input file now says */
                        /* "sweep rate\tstart mass\tend mass\tstart index\tend
index"*/
                        /* Now read in that line */
                        fgets(junk, 80 , fp);
                        /* Now get important parameters */
                        fscanf(fp,"%f%f%f%d%d",&sweep_rate,&start_mass,&end_mass,
                                 &start_index,&end_index);
                        /* get carriage return */
                        fgets(junk,80,fp);
                        /* Input file now says "Mass\tSignal\n" */
                        /* Now read in that line */
                        fgets(junk,80,fp);
                        /* Now read in masses and signals */
                        for(i=start_index;i<=end_index;i++) {
                           fscanf(fp,"%f%f",&masses[i],&signals[i]);
                        }
                        fclose(fp);
                        /* Set controls to proper values */
                        SetCtrlVal(hPanel,MASSSPEC_SWEEPRATE,sweep_rate);
                        SetCtrlVal(hPanel,MASSSPEC_STARTMASS,start_mass);
                        SetCtrlVal(hPanel,MASSSPEC_ENDMASS,end_mass);
                        /* reset x axis on spectrum */

  GetAxisRange(hPanel,MASSSPEC_SPECTRUM,&xauto,&x_min,&x_max,
                                  &yauto,&y_min,&y_max);
                        x_min=start_mass;
                        x_max=end_mass;

  SetAxisRange(hPanel,MASSSPEC_SPECTRUM,xauto,x_min,x_max,yauto,y_min,y_max);
```

```
                    /* clear graph */
                    DeleteGraphPlot(hPanel,MASSSPEC_SPECTRUM,-1,1);
                    PlotXY(hPanel,MASSSPEC_SPECTRUM,masses+start_index,
                          signals+start_index,end_index-start_index,
                          3,3,0,11,1,15);
                    /* Set appropriate active controls */
                    SetCtrlAttribute(hPanel,MASSSPEC_DELETE,15,TRUE);
                    SetCtrlAttribute(hPanel,MASSSPEC_ZOOMIN,15,TRUE);
                    SetActiveCtrl(MASSSPEC_SPECTRUM);
                    /*-----------------------------------------------------
----------*/
                    /*  The initial axis configurations are preserved so that
it may   */
                    /*  always be possible to return to the original zooming
level.    */
                    /*-----------------------------------------------------
----------*/
                    zoomlevel = 0;
                    tab1x[zoomlevel] = start_mass;
                    tab1y[zoomlevel] = y_min;
                    tab2x[zoomlevel] = end_mass;
                    tab2y[zoomlevel] = y_max;
                 }
              }
              break;
           case MASSMENU_FILE_PRINT:  /* Print out mass spec */
              OutputGraph(0,panel->PrintSetup.OutputFile,
                                panel->PrintSetup.ScaleFactor,
                                hPanel,MASSSPEC_SPECTRUM);
              break;
           case MASSMENU_FILE_PRINTSETUP:
              PrintSetup(panel);
              break;
           case MASSMENU_SETUP_CHAN :
              SetupChannels(channel,labels,panel,flowtube);
              /* Update the screen */
              UpdateMainCtrls(channel,labels,panel);
              break;
           case MASSMENU_SETUP_FLOWTUBPAR:
              EditFlowtubeParamters(flowtube,panel,channel);
              /* Update the screen */
              UpdateMainCtrls(channel,labels,panel);
              break;
           case MASSMENU_SETUP_YLOGON:
              SetMenuItemCheckmark(MASSMENU_SETUP_YLOGON,TRUE);
              SetMenuItemCheckmark(MASSMENU_SETUP_YLOGOFF,FALSE);
              SetInputMode(hMenuBar,MASSMENU_SETUP_YLOGON,FALSE);
              SetInputMode(hMenuBar,MASSMENU_SETUP_YLOGOFF,TRUE);
              /* Rescale Y axis, 21 refers to y log axis variable      */
              SetGraphAttribute(hPanel,MASSSPEC_SPECTRUM,21,TRUE);
              ylogon = TRUE;
              break;
           case MASSMENU_SETUP_YLOGOFF:
              SetMenuItemCheckmark(MASSMENU_SETUP_YLOGON,FALSE);
              SetMenuItemCheckmark(MASSMENU_SETUP_YLOGOFF,TRUE);
              SetInputMode(hMenuBar,MASSMENU_SETUP_YLOGON,TRUE);
              SetInputMode(hMenuBar,MASSMENU_SETUP_YLOGOFF,FALSE);
              /* Rescale Y axis, 21 refers to y log axis variable      */
```

```
                SetGraphAttribute(hPanel,MASSSPEC_SPECTRUM,21,FALSE);
                ylogon = FALSE;
                break;
             case MASSMENU_SETUP_CHANGEXY: /* CHANGE RANGES ON AXIS */

   GetAxisRange(hPanel,MASSSPEC_SPECTRUM,&xauto,&x_min,&x_max,&yauto,&y_min,
                                       &y_max);

   ChangeXYRange(hPanel,MASSSPEC_SPECTRUM,1,1,xauto,x_min,x_max,yauto,y_min,y_
max);
                break;
             case MASSMENU_SWITCH_KINETICS:
             case MASSMENU_SWITCH_CHART:
             case MASSMENU_FILE_QUIT : /* END PROGRAM */
                /* SAVE STATES */
                if ((fp = fopen("massspec.prm","wb")) != NULL ) {
                   /* write masses */
                   fwrite(masses,sizeof(float),BOARD_RES,fp);
                   /* write signals */
                   fwrite(signals,sizeof(float),BOARD_RES,fp);
                   /* write sweep_rate */
                   fwrite(&sweep_rate,sizeof(float),1,fp);
                   /* write start_mass */
                   fwrite(&start_mass,sizeof(float),1,fp);
                   /* write end_mass */
                   fwrite(&end_mass,sizeof(float),1,fp);
                   /* record if log scale is being used */
                   fwrite(&ylogon,sizeof(int),1,fp);
                   /* close file */
                   fclose(fp);
                   /* save panel state */
                   SavePanelState(hPanel,"massspec.pnl");
                }
                /* free arrays */
                free(masses);
                free(signals);
                /* Unload menubar and panel */
                UnloadMenuBar();
                UnloadPanel(hPanel);
                /* RETURN PROPER VALUE */
                if(id == MASSMENU_SWITCH_KINETICS) return KINETICS_PANEL;
                else if(id==MASSMENU_SWITCH_CHART) return STRIPCHART_PANEL;
                else if(id==MASSMENU_FILE_QUIT)    return QUIT;
                break;
             default :
                break;
          }
       }
       else if(handle == hPanel) {
          switch (id) {
             case MASSSPEC_SCAN :
                /* disable copy data graph feature and make cursors */
                /* dissapear temporarily */
                SetGraphAttribute(hPanel,MASSSPEC_SPECTRUM,30,1);
                SetCursorAttribute(hPanel,MASSSPEC_SPECTRUM,1,2,3);
                SetCursorAttribute(hPanel,MASSSPEC_SPECTRUM,2,2,3);
                /* clear graph */
                DeleteGraphPlot(hPanel,MASSSPEC_SPECTRUM,-1,1);
```

```
            /* reset x axis on spectrum */

   GetAxisRange(hPanel,MASSSPEC_SPECTRUM,&xauto,&x_min,&x_max,&yauto,&y_min,
                              &y_max);
            x_min=start_mass;
            x_max=end_mass;

   SetAxisRange(hPanel,MASSSPEC_SPECTRUM,xauto,x_min,x_max,yauto,y_min,y_max);
            /* take a spectrum */
            /* calculate dwell time in milliseconds */
            dwell = (1000L*QuadRange)/(BOARD_RES * sweep_rate);
            /* make sure dwell is at least one millisecond */
            if(dwell==0) dwell=1;
            /* compute appropriate place to start in array for starting
mass */
            start_index = (int) ((start_mass * BOARD_RES) / QuadRange);
            /* compute appropriate place to stop in array for end mass */
            end_index = (int) ((end_mass * BOARD_RES) / QuadRange);
            /* Screen will redraw only once every other second.  Calculate
the number */
            /* of points that can be taken per second
*/
            pause = (int)(.1*(BOARD_RES * sweep_rate)/(float)QuadRange +
2);
            /* make sure pause isn't greater than all the points needed */
            if(pause>(end_index-start_index)) pause=end_index-start_index;
            /* Outer loop goes through all needed masses */
            for(i=start_index;i<end_index;i+=pause) {
               /* Inner loop lasts about one tenth of a second */
               for(j=0;(j<pause)&&((i+j)<end_index);j++) {
                  masses[i+j] =((float)QuadRange * (i+j)) /
(float)BOARD_RES;
                  /* set spectrometer to mass */
                  WriteVoltage(masses[i+j]*10.0/(float)QuadRange);
                  /* get signal */
                  signals[i+j] = GetCounts(dwell);
               }
               /* Plot Spectrum Segment*/
               if(i!=start_index) {
                  PlotXY(hPanel,MASSSPEC_SPECTRUM,masses+i-1,
                        signals+i-1,j+1,3,3,0,11,1,15);
               }
               else {
                  PlotXY(hPanel,MASSSPEC_SPECTRUM,masses+i,
                        signals+i,j,3,3,0,11,1,15);
               }
               /* Allow for user to abort scan */
               GetUserEvent (NO_WAIT, &handle2, &id2);
               if((handle2 == hPanel)&&(id2 == MASSSPEC_SCAN)) break;

            }
            /* Abort here if user pressed acquire during scan */
            if((handle2 == hPanel)&&(id2 == MASSSPEC_SCAN)) {
               DeleteGraphPlot(hPanel,MASSSPEC_SPECTRUM,-1,1);
               SetCtrlAttribute(hPanel,MASSSPEC_SCAN,15,TRUE);
               SetCtrlAttribute(hPanel,MASSSPEC_DELETE,15,FALSE);
               SetCtrlAttribute(hPanel,MASSSPEC_ZOOMIN,15,FALSE);
               SetCtrlAttribute(hPanel,MASSSPEC_ZOOMOUT,15,FALSE);
```

```
                     SetCtrlAttribute(hPanel,MASSSPEC_RESTORE,15,FALSE);
                     break;
                  }
                  /* Plot Spectrum */
                  DeleteGraphPlot(hPanel,MASSSPEC_SPECTRUM,-1,1);
                  SetGraphAttribute(hPanel,MASSSPEC_SPECTRUM,30,0);
                  SetCursorAttribute(hPanel,MASSSPEC_SPECTRUM,1,2,0);
                  SetCursorAttribute(hPanel,MASSSPEC_SPECTRUM,2,2,0);
                  PlotXY(hPanel,MASSSPEC_SPECTRUM,masses+start_index,
                           signals+start_index,end_index-
start_index,3,3,0,11,1,15);

                  /* Set appropriate active controls */
                  SetCtrlAttribute(hPanel,MASSSPEC_DELETE,15,TRUE);
                  SetCtrlAttribute(hPanel,MASSSPEC_ZOOMIN,15,TRUE);
                  SetActiveCtrl(MASSSPEC_SPECTRUM);
                  /*--------------------------------------------------------
----*/
                  /*  The initial axis configurations are preserved so that it
may   */
                  /*  always be possible to return to the original zooming level.
*/
                  /*--------------------------------------------------------
----*/
                  zoomlevel = 0;
                  tab1x[zoomlevel] = start_mass;
                  tab1y[zoomlevel] = y_min;
                  tab2x[zoomlevel] = end_mass;
                  tab2y[zoomlevel] = y_max;
                  break;
               case MASSSPEC_DELETE :
                  /*--------------------------------------------------------
----*/
                  /*  When a plot is deleted, the panel's controls are reset to
the  */
                  /*  status in which they were in the beginning.
*/
                  /*--------------------------------------------------------
----*/
                  DeleteGraphPlot(hPanel,MASSSPEC_SPECTRUM,-1,1);
                  SetCtrlAttribute(hPanel,MASSSPEC_SCAN,15,TRUE);
                  SetCtrlAttribute(hPanel,MASSSPEC_DELETE,15,FALSE);
                  SetCtrlAttribute(hPanel,MASSSPEC_ZOOMIN,15,FALSE);
                  SetCtrlAttribute(hPanel,MASSSPEC_ZOOMOUT,15,FALSE);
                  SetCtrlAttribute(hPanel,MASSSPEC_RESTORE,15,FALSE);
                  break;
               case MASSSPEC_ZOOMIN :
                  /*--------------------------------------------------------
----*/
                  /*  When the Zoom In button is pushed, the axes will be re-
scaled  */
                  /*  to the window defined by cursors 1 and 2. Because all
relative */
                  /*  positions of the 2 cursors are allowed, swaps may be
needed.   */
                  /*--------------------------------------------------------
----*/
                  GetGraphCursor(hPanel,MASSSPEC_SPECTRUM,1,&c1x,&c1y);
               GetGraphCursor(hPanel,MASSSPEC_SPECTRUM,2,&c2x,&c2y);
               if (c1x == c2x || c1y == c2y)
                  MessagePopup("Deltas must be non-zero...");
               else {
                  if (c1x > c2x) {
                     tmp = c1x;
                     c1x = c2x;
                     c2x = tmp;
                  }
                  if (c1y > c2y) {
                     tmp = c1y;
                     c1y = c2y;
                     c2y = tmp;
                  }
                  /*----------------------------------------------------------
-----*/
                  /*  Update zooming level. If zooming level is maximum,
disable   */
                  /*  further zooming. The Zoom Out and Restore buttons should
*/
                  /*  always be enabled after zooming in.
*/
                  /*----------------------------------------------------------
-----*/
                  SetCtrlAttribute(hPanel,MASSSPEC_RESTORE,15,TRUE);
                  SetCtrlAttribute(hPanel,MASSSPEC_ZOOMOUT,15,TRUE);
                  zoomlevel++;
                  if (zoomlevel == MAX_ZOOM)
                     SetCtrlAttribute(hPanel,MASSSPEC_ZOOMIN,15,FALSE);
                  else {
                     /*-------------------------------------------------------
------*/
                     /*  Store the current range of each axis in a table, so
that   */
                     /*  they may be restored at a later point.
*/
                     /*-------------------------------------------------------
------*/
                     tab1x[zoomlevel] = c1x;
                     tab1y[zoomlevel] = c1y;
                     tab2x[zoomlevel] = c2x;
                     tab2y[zoomlevel] = c2y;
                  }
                  SetAxisRange(hPanel,MASSSPEC_SPECTRUM,0,c1x,c2x,0,c1y,c2y);
               }
               break;
            case MASSSPEC_ZOOMOUT :
            case MASSSPEC_RESTORE :
               /*-------------------------------------------------------------
----*/
               /*  Update zooming level. If zooming level is minimum, disable
*/
               /*  further zooming. The Zoom In button should always be
enabled   */
               /*  after zooming out.
*/
               /*-------------------------------------------------------------
----*/
```

```
                SetCtrlAttribute(hPanel,MASSSPEC_ZOOMIN,15,TRUE);
                if (id == MASSSPEC_RESTORE)
                   zoomlevel = 0;
                else
                   zoomlevel--;
                if (zoomlevel == 0) {
                   SetCtrlAttribute(hPanel,MASSSPEC_RESTORE,15,FALSE);
                   SetCtrlAttribute(hPanel,MASSSPEC_ZOOMOUT,15,FALSE);
                }
                /*------------------------------------------------------------
----*/
                /*  Restore the scaling range of the previous zooming level.
*/
                /*------------------------------------------------------------
----*/
                c1x = tab1x[zoomlevel];
                c1y = tab1y[zoomlevel];
                c2x = tab2x[zoomlevel];
                c2y = tab2y[zoomlevel];
                SetAxisRange(hPanel,MASSSPEC_SPECTRUM,0,c1x,c2x,0,c1y,c2y);
                break;
             case MASSSPEC_SWEEPRATE :
                GetCtrlVal(hPanel,MASSSPEC_SWEEPRATE,&sweep_rate);
                break;
             case MASSSPEC_STARTMASS :
                GetCtrlVal(hPanel,MASSSPEC_STARTMASS,&start_mass);
                break;
             case MASSSPEC_ENDMASS :
                GetCtrlVal(hPanel,MASSSPEC_ENDMASS,&end_mass);
                break;
          }
       }

       /*---------------------------------------------------------------------
*/
       /*  Read the current position of each cursor.
*/
       /*---------------------------------------------------------------------
*/
       GetGraphCursor(hPanel,MASSSPEC_SPECTRUM,1,&c1x,&c1y);
       GetGraphCursor(hPanel,MASSSPEC_SPECTRUM,2,&c2x,&c2y);
       GetGraphCursor(hPanel,MASSSPEC_SPECTRUM,3,&c3x,&c3y);
       /*---------------------------------------------------------------------
*/
       /*  Update cursor readouts.
*/
       /*---------------------------------------------------------------------
*/
       SetCtrlVal(hPanel,MASSSPEC_C1_X,c1x);
       SetCtrlVal(hPanel,MASSSPEC_C1_Y,c1y);
       SetCtrlVal(hPanel,MASSSPEC_C2_X,c2x);
       SetCtrlVal(hPanel,MASSSPEC_C2_Y,c2y);
       SetCtrlVal(hPanel,MASSSPEC_DELTA_X,c2x-c1x);
       SetCtrlVal(hPanel,MASSSPEC_DELTA_Y,c2y-c1y);
       SetCtrlVal(hPanel,MASSSPEC_X_POS,c3x);
       SetCtrlVal(hPanel,MASSSPEC_Y_POS,c3y);

       ReadFlows(channel,panel,flowtube,partial_pressure,&total_flow,
                              &tube_pressure,&velocity);

      }
   }
```

```
                              /* READFLOW.C */
#define BOARD_RES 4096
#define MAX_VOLTAGE 10.0
#include "b:\modular\struct.h"

void ReadFlows(struct gas_input *channel,struct handles *panel,
                        struct FlowTubeParameters *flowtube,
                        double *partial_pressure,double *total_flow,
                        double *tube_pressure,double *velocity)
{
      int i,j,hPanel;
      char hours_min_sec[40];
      /* Data Acquisition Variables */
      struct PlotParameters plot;
      struct sample point;
      int current_plot_hdl;
      double numerator[3],flow;
      double pressure;
      double signal,volume,reynolds;

      hPanel = panel->hFlowsPanel;
      /* -----Read Voltages from all baratrons and flowmeters ----- */
      for(i=1;i<=7;i++) {
         channel[i].voltage = ReadVoltage(i);
         channel[i].reading = channel[i].voltage*channel[i].range /
            (channel[i].max_volts+TINY);
      }
      /* Code to update species partial pressures in flowtube
*/
      /* Mixing ratio of species = (ratio1*Flow1+ratio2*Flow2+ratio3*Flow3) /
*/
      /*                                              total_flow
*/
      /* Partial Pressure = total_pressure*mixing_ratio
*/

      for(i=0;i<3;i++) {
         partial_pressure[i]=0;         /* Clear Variables */
         numerator[i]=0;
      }
      *total_flow=0;
      for(i=1;i<=7;i++) {
         if(channel[i].type == BUBBLER) {
            flow = channel[i].reading;
            *total_flow += flow;
            pressure = channel[channel[i].press_chan].reading;
            for(j=0;j<3;j++)
               numerator[j] += ( channel[i].amount[j] * flow / (pressure +
TINY));
         }
         else if(channel[i].type == BULB) {
            flow = channel[i].reading;
            *total_flow += flow;
            for(j=0;j<3;j++) numerator[j] += flow*channel[i].amount[j];
         }
         else if(channel[i].type == TUBE_BARO) {
            *tube_pressure = channel[i].reading;
         }
      }
      /* Calculate Velocity, time, and Reynold's number
*/
      /*  velocity = flow(corrected for new pressure and temperature)/ Area
*/
      *velocity = (*total_flow)*flowtube->temp*760/
         (273*(*tube_pressure)*PI*flowtube->radius*flowtube->radius+TINY);
      *velocity /= 60; /* convert from minutes to seconds */

      /* Calculate Reynold's number. */
      switch (flowtube->gas_type) {
         case HE: /* Helium */
            flowtube->eata =  -.0007246 * flowtube->temp * flowtube->temp
                        +.8026 * flowtube->temp + 22.74;
            break;
         case NITROGEN: /* Nitrogen */
            flowtube->eata = 182.7;
            break;
         case ARGON: /* Argon */
            flowtube->eata =  -4.921e-5 * flowtube->temp * flowtube->temp
                        +.09364 * flowtube->temp + 199.3;
            break;
      }

      reynolds = flowtube->gas_mw * (16.03 / flowtube->temp * *tube_pressure)
                        * flowtube->radius * *velocity / flowtube->eata;

      for(i=0;i<3;i++)
         partial_pressure[i] = numerator[i]*(*tube_pressure)/(*total_flow
+TINY);


      /*-------------------- Display values on screen  ----------------------
*/
      SetCtrlVal(hPanel,FLOWSPANEL_CHAN1,channel[1].reading);
      SetCtrlVal(hPanel,FLOWSPANEL_CHAN2,channel[2].reading);
      SetCtrlVal(hPanel,FLOWSPANEL_CHAN3,channel[3].reading);
      SetCtrlVal(hPanel,FLOWSPANEL_CHAN4,channel[4].reading);
      SetCtrlVal(hPanel,FLOWSPANEL_CHAN5,channel[5].reading);
      SetCtrlVal(hPanel,FLOWSPANEL_CHAN6,channel[6].reading);
      SetCtrlVal(hPanel,FLOWSPANEL_CHAN7,channel[7].reading);
      SetCtrlVal(hPanel,FLOWSPANEL_SPECIES1,partial_pressure[0]);
      SetCtrlVal(hPanel,FLOWSPANEL_SPECIES2,partial_pressure[1]);
      SetCtrlVal(hPanel,FLOWSPANEL_SPECIES3,partial_pressure[2]);
      SetCtrlVal(hPanel,FLOWSPANEL_VEL,*velocity);
      SetCtrlVal(hPanel,FLOWSPANEL_RN,reynolds);
      SetCtrlVal(hPanel,FLOWSPANEL_TEMP,flowtube->temp);

      /* ------------------- Update the time -------------------------*/
      strcpy(hours_min_sec,TimeStr());
      SetCtrlVal(hPanel,FLOWSPANEL_TIME,hours_min_sec);
}
```

```
                                /* SETUP.C */
#define PC_LPM_16 13

#include "b:\modular\struct.h"
/*==========================================================================*/
/*                          Setup For Program                               */
/*==========================================================================*/

extern short AnalogBoardType, CounterOrAnalog, AnalogOutput, Slot, QuadRange;
extern float SamplesPerMS;
short LoadProgramPanels(struct handles *panel) {

   /*------------------------------------------------------------------------*/
   /*  Load the panel from the resource file.  Use the constant assigned     */
   /*  in the editor to refer to the panel.  The handle returned by          */
   /*  LoadPanel must be used to reference the panel in all subsequent       */
   /*  function calls. If the panel handle is negative, the load failed,     */
   /*  so print a message and exit the program. Otherwise, display the       */
   /*  the panel.                                                            */
   /*------------------------------------------------------------------------*/

   panel->hFlowsPanel = LoadPanel (RESOURCE_FILE, FLOWSPANEL);
   if (panel->hFlowsPanel < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   panel->hChannelSetup = LoadPanel(RESOURCE_FILE, SETCHANNEL);
   if (panel->hChannelSetup < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   panel->hEditLabels = LoadPanel (RESOURCE_FILE, SPECNAME);
   if (panel->hEditLabels < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   panel->hFlowtubeParams = LoadPanel(RESOURCE_FILE, FLOWTPARAM);
   if (panel->hFlowtubeParams < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   panel->PrintSetup.hPrintSetup = LoadPanel(RESOURCE_FILE, PRINTSETUP);
   if (panel->PrintSetup.hPrintSetup < 0) {
      MessagePopup("Unable to load the required panel from the resource
file.");
      return 0;
   }

   return 1;
}
```

```
short SetupParameters(struct gas_input *channel,struct handles *panel,
      struct FlowTubeParameters *flowtube,struct Titles *labels) {
   /*------------------------------------------------------------------------*/
   /*  Load up critical values from the previous time the program was run.  */
   /*  This file should be in the same directory as the program and be      */
   /*  called "dkin.prm".  If this file cannot be found, load into the      */
   /*  variables the default values.  These are saved when the user uses    */
   /*  the Save Setup command from the File menu. Values are a binary file. */
   /*  The format the variables are saved in are the following order:       */
   /*      struct gas_input channel[8];                                     */
   /*      struct FlowTubeParameters flowtube;                              */
   /*      struct Titles labels;                                            */
   /*                                                                       */
   /*------------------------------------------------------------------------*/
   int flag = FALSE;
   int status,i,j;
   FILE *fp;

   fp=fopen("dkin.prm","rb");
   if(fp!=NULL) { /* Success, found the file, now read in parameters. */
      status=fread(channel,sizeof(channel[0]),8,fp);
      if(status!=8) flag = (flag || 1);
      status=fread(flowtube,sizeof(*flowtube),1,fp);
      if(status!=1) flag = (flag || 1);
      status = fread(labels,sizeof(*labels),1,fp);
      if(status!=1) flag = (flag || 1);
      fclose(fp);
   }
   else flag = 1;
   if(flag) {  /* Error, establish default values.                   */
      /* Initial state is that all readings come from a flowtube barotron */
      /* in the resource file.  Therefore, set every channel to that      */
      /* setting.                                                         */
      for(i=1;i<=7;i++) {
         channel[i].type = TUBE_BARO;
         channel[i].barotron.index = TEN;
         channel[i].barotron.range = 10.0;
         channel[i].flowmeter.index = HE;
         channel[i].flowmeter.range = 10.0;
         channel[i].flowmeter.gas_correct = 1.0;
         channel[i].range = 10.0;
         for(j=0;j<3;j++) channel[i].amount[j] = 0;
         channel[i].press_chan=2;
         channel[i].max_volts = BAROTRON_VOLTAGE;
         channel[i].voltage = .1;
         channel[i].reading = .1;
      }
      /* Make one channel a flow so we have some velocity */
      channel[3].type = BULB;

      /* Setup flowtube settings */

      flowtube->gas_type = HE;
      flowtube->gas_correct = 1.0;
      flowtube->gas_mw = 4.0;
      flowtube->temp = 200;
```

```
        flowtube->radius = 1.4;
        flowtube->eata = 1.0;

        /* Get labels */
        GetCtrlVal(panel->hEditLabels,SPECNAME_SPEC1,labels->species[0]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_SPEC2,labels->species[1]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_SPEC3,labels->species[2]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN1,labels->channel[1]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN1,labels->channel[1]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN2,labels->channel[2]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN3,labels->channel[3]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN4,labels->channel[4]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN5,labels->channel[5]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN6,labels->channel[6]);
        GetCtrlVal(panel->hEditLabels,SPECNAME_CHAN7,labels->channel[7]);


    }
    /* Put values into places from file */
    /* Put the Labels in */
    SetCtrlVal(panel->hEditLabels,SPECNAME_SPEC1,labels->species[0]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_SPEC2,labels->species[1]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_SPEC3,labels->species[2]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN1,labels->channel[1]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN1,labels->channel[1]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN2,labels->channel[2]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN3,labels->channel[3]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN4,labels->channel[4]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN5,labels->channel[5]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN6,labels->channel[6]);
    SetCtrlVal(panel->hEditLabels,SPECNAME_CHAN7,labels->channel[7]);

    /* Set the channel # indicator to the top of the slide */
    SetCtrlVal(panel->hChannelSetup,SETCHANNEL_CHANUM,0);
    SetChanNum(channel,panel);
    UpdateMainCtrls(channel,labels,panel);
    UpdateChannelPanel(labels,panel);

    /* Set Flowtube Variables */
    SetCtrlVal(panel->hFlowtubeParams,FLOWTPARAM_FLOWTEMP,flowtube->temp);
    SetCtrlVal(panel->hFlowtubeParams,FLOWTPARAM_CARRIERGAS,flowtube-
>gas_type);
    SetCtrlVal(panel->hFlowtubeParams,FLOWTPARAM_RADIUS,flowtube->radius);


    return 1;
}

/*-------------------------------------------------------------------------*/
/*    Save critical values.  See SetupParameters(...) above.              */
/*  This file should be in the same directory as the program and be       */
/*  called "dkin.prm".  Values are a binary file.                          */
/*  The format the variables are saved in are the following order:         */
/*      struct gas_input channel[8];                                       */
/*      struct FlowTubeParameters flowtube;                                */
/*      struct Titles labels;                                              */
/*-------------------------------------------------------------------------*/
```

```
short SaveSetupParameters(struct gas_input *channel,
      struct FlowTubeParameters *flowtube,struct Titles *labels) {

   int flag = FALSE;
   int status;
   FILE *fp;
   fp=fopen("dkin.prm","wb");
   if(fp) { /* Success, found the file, now write in parameters. */
      status=fwrite(channel,sizeof(channel[0]),8,fp);
      if(status!=8) flag = (flag || 1);
      status=fwrite(flowtube,sizeof(*flowtube),1,fp);
      if(status!=1) flag = (flag || 1);
      status = fwrite(labels,sizeof(*labels),1,fp);
      if(status!=1) flag = (flag || 1);
   }
   fclose(fp);
   return 1;

}

/* Setup for boards */

short InitializeBoards(void) {

   int i,boardCode,status;
   double voltage=0,total_voltage=0,interval;
   float rate =0;
   struct time t1,t2;
   char result[50];
   /* See if its an National Instruments Board */
   if(AnalogBoardType == 0) { /* NI Board */
      /* See if we need to auto detect the board number */
      if(Slot == 0) {
         for(i=1;i<=4;i++) {
            status=Init_DA_Brds(i,&boardCode);
            if(status < 0) continue;
            else {
               Slot = i;
               break;
            }
         }
         if(i==5) {
            MessagePopup("Board couldn't be found.");
            return 0;
         }
      }
      else {
         status=Init_DA_Brds(Slot,&boardCode);
         if(status < 0) {
            MessagePopup("Board couldn't be found.");
            return 0;
         }
      }
      /* Now Congifure board */
      /* look for PCLPM 16 Board */
      if(boardCode == PC_LPM_16) AI_Configure(Slot,-1,1,5,1,0);
      else AI_Configure(Slot,-1,1,10,1,0);
      AI_Clear;
      AO_Config(Slot,AnalogOutput,1,10,0);
      /* Now see how much time it takes to do 5000 scans */
      MessagePopup("Calibrating A/D timing.");
      gettime(&t1);
      for(i=0;i<5000;i++) {
         AI_VRead(Slot,0,1,&voltage);
         total_voltage += voltage;
      }
      gettime(&t2);
   }
   else { /* DT2801 board */
      /* Now see how much time it takes to do 5000 scans */
      MessagePopup("Calibrating A/D timing.");
      InitDT2801();
      gettime(&t1);
      for(i=0;i<5000;i++) {
         rate += DTVIn(0);
      }
      gettime(&t2);
   }
   /* Calculate time interval */
   interval=(float)t2.ti_hund/100.0+(float)t2.ti_sec+(float)t2.ti_min*60.0
         -
((float)t1.ti_hund/100.0+(float)t1.ti_sec+(float)t1.ti_min*60.0);
   SamplesPerMS = 5000.0/(interval*1000);
   if (SamplesPerMS < 1.0) SamplesPerMS=1.0;
   sprintf(result,"Sampling rate is %f per millisecond.", SamplesPerMS);
   MessagePopup(result);
   return 1;
}

void PrintSetup(struct handles *panel) {
   int flag,control,hPanel,status;
   char dirname[80],filename[80];

   hPanel = panel->PrintSetup.hPrintSetup;

   InstallPopup(hPanel);
   /* Setup Values to present ones */
   SetCtrlVal(hPanel,PRINTSETUP_PRINT_TO,panel->PrintSetup.PrintTo);
   SetCtrlVal(hPanel,PRINTSETUP_OUTPUTFILE,
                           panel->PrintSetup.OutputFile);
   SetCtrlVal(hPanel,PRINTSETUP_PAGE_ORIENTATION,
                              panel->PrintSetup.Orientation);
   SetCtrlVal(hPanel,PRINTSETUP_SCALE_FACTOR,
                           panel->PrintSetup.ScaleFactor);
   SetCtrlVal(hPanel,PRINTSETUP_PAGE_EJECT,
                           panel->PrintSetup.PageEject);

   flag = TRUE;
   while(flag)
   {
      GetPopupEvent(WAIT,&control);
      switch (control)
      {
         case PRINTSETUP_PRINT_TO:
            GetCtrlVal(hPanel,PRINTSETUP_PRINT_TO,&panel->PrintSetup.PrintTo);
            if(panel->PrintSetup.PrintTo)
         { /* print to lpt1 */
            /* make output filename a null string */
            panel->PrintSetup.OutputFile[0]=0;
         }
         else
         { /* need to get a filename to print to file */
            GetProgramDir(dirname);
            status=FileSelectPopup(dirname,"*.prn","Select file to print
to.",NO_RESTRICT,NO_RESTRICT,TRUE,filename);
            if(status)
            {
               if(status==FILE_EXISTS)
               {
                  status=ConfirmPopup("That file already exits.
Overwrite?");
               }
               if(status) strcpy(panel->PrintSetup.OutputFile,filename);
               else {
                  panel->PrintSetup.OutputFile[0]=0;
                  /* Put setttings back to print to LPT1 */
                  panel->PrintSetup.PrintTo = 1;
                  SetCtrlVal(hPanel,PRINTSETUP_PRINT_TO,1);
               }
            }
         }
         /* Write filename to panel */
         SetCtrlVal(hPanel,PRINTSETUP_OUTPUTFILE,
                        panel->PrintSetup.OutputFile);
         break;
      case PRINTSETUP_PAGE_ORIENTATION:
         /* 1 = Landscape, 0 = Portrait */
         GetCtrlVal(hPanel,PRINTSETUP_PAGE_ORIENTATION,
                          &panel->PrintSetup.Orientation);
         break;
      case PRINTSETUP_SCALE_FACTOR:
         /* 1 = full size, 0 = screen size */
         GetCtrlVal(hPanel,PRINTSETUP_SCALE_FACTOR,
                        &panel->PrintSetup.ScaleFactor);
         break;
      case PRINTSETUP_PAGE_EJECT:
         /* 1 = eject , 0 = don't eject */
         GetCtrlVal(hPanel,PRINTSETUP_PAGE_EJECT,
                        &panel->PrintSetup.PageEject);
         break;
      case PRINTSETUP_FINISHED:
         flag = FALSE;
         break;
   }
}
if(panel->PrintSetup.OutputFile[0]==0)
{
   ConfigurePrinter("LPT1",panel->PrintSetup.Orientation,
                        0,0,panel->PrintSetup.PageEject);
}
else
{
   ConfigurePrinter(panel->PrintSetup.OutputFile,
                          panel->PrintSetup.Orientation,
                                    0,0,panel->PrintSetup.PageEject);
   }
   RemovePopup(0);
}
```

```
                              /* DRIVERS.C */
extern short AnalogBoardType, CounterOrAnalog, AnalogOutput, Slot, QuadRange;
extern float SamplesPerMS;
#include "b:\modular\struct.h"
float GetCounts(short milliseconds) {
   float rate = 0;
   double voltage, total_voltage=0;
   long i,samples;

   if(CounterOrAnalog) return CounterRate(milliseconds);
   else {
      samples = (long)(milliseconds*SamplesPerMS + .5);
      if(AnalogBoardType == 0) { /* NI Board */
         for(i=0;i<samples;i++) {
            AI_VRead(Slot,0,1,&voltage);
            total_voltage += voltage;
         }

      }
      else { /* DT2801 Board */
         for(i=0;i<samples;i++) {
            total_voltage += DTVIn(0);
         }
      }
      rate = total_voltage / samples;
      return rate;
   }
}

float ReadVoltage(short channel) {
   double voltage;
   /* NI Board */
   if(AnalogBoardType == 0) {
      AI_VRead(Slot,channel,1,&voltage);
      return (float) voltage;
   }
   /* DT2801 */
   else return DTVIn(channel);
}

void WriteVoltage(float voltage) {
   /* NI Board */
   if(AnalogBoardType == 0) {
      AO_VWrite(Slot,AnalogOutput,voltage);
      return;
   }
   /* DT2801 */
   else {
      DTVOut(AnalogOutput,voltage);
      return;
   }
}
```

```
                                        /* DT2801.C */
#define Base  0x2EC
#define Command (Base + 1)
#define Status Command
#define Data Base
#define Command_Wait 0x04
#define Write_Wait  0x02
#define Read_Wait 0x05
#define Data_In_Full  0x02
#define Ready 0x04
#define Data_Out_Ready 0x01
#include <conio.h>

/* #include "b:\modular\struct.h" */
void InitDT2801(void);
float DTVIn(unsigned char channel);
void DTVOut(unsigned char channel,float voltage);

void InitDT2801(void) {
   unsigned char byte;

   /* Stop board from whatever it's doing */
   outp(Command,0x0F);
   /* Read Data Register to Clear Data Out Flag */
   byte = inp(Data);
   /* Wait for Data in Full flag to clear and Ready flag to set. */
   /* Then write RESET command */

   /* Wait for Data in Full flag to clear and Ready flag to set. */
   /* Then write Clear command */
   do {
      byte = inp(Status);
   } while((byte & Data_In_Full) | (!(byte & Ready)));
   outp(Command,0x01); /* Clear Command*/
}

void DTVOut(unsigned char channel, float voltage) {
   unsigned char byte,low,high;

   low = ((short)(voltage/10.0*4096)) % 0x100;
   high = ((short)(voltage/10.0*4096)) / 0x100;


   /* Wait for Data in Full flag to clear and Ready flag to set. */
   /* Then write WRITE DAC IMMEDIATE command */
   do {
      byte = inp(Status);
   } while((byte & Data_In_Full) | (!(byte & Ready)));
   outp(Command,0x08); /* WRITE DAC Immediate */

   /* Wait for Data in Full flag to clear. */
   /* Then write DAC SELECT BYTE */
   do {
      byte = inp(Status);
   } while(byte & Data_In_Full);
   outp(Data,channel); /* 0 means output channel 0 */

   /* Wait for Data in Full flag to clear. */
```

```
    /* Then write low byte */
    do {
       byte = inp(Status);
    } while(byte & Data_In_Full);
    outp(Data,low);

    /* Wait for Data in Full flag to clear. */
    /* Then write high byte */
    do {
       byte = inp(Status);
    } while(byte & Data_In_Full);
    outp(Data,high);
 }

 float DTVIn(unsigned char channel) {
    unsigned char byte,low,high;
    float voltage;

    /* Wait for Data in Full flag to clear and Ready flag to set. */
    /* Then write READ A/D IMMEDIATE command */
    do {
       byte = inp(Status);
    } while((byte & Data_In_Full) | (!(byte & Ready)));
    outp(Command,0x0C); /* READ A/D Immediate */

    /* Wait for Data in Full flag to clear. */
    /* Then write Gain of 1 (0x00) */
    do {
       byte = inp(Status);
    } while(byte & Data_In_Full);
    outp(Data,0x00); /* 0 means Gain = 1 */
    /* Wait for Data in Full flag to clear. */
    /* Then write channel number */
    do {
       byte = inp(Status);
    } while(byte & Data_In_Full);
    outp(Data,channel);

    /* Wait for Data Out flag to be set and Ready flag too */
    do {
       byte = inp(Status);
    } while(!(byte & (Data_Out_Ready | Ready)));
    low = inp(Data);
    /* Wait for Data Out flag to be set and Ready flag too */
    do {
       byte = inp(Status);
    } while(!(byte & (Data_Out_Ready | Ready)));
    high = inp(Data);
    voltage = (0x100 * high + low)*10.0/4096;
    return voltage;
 }
```

```
                                   /* DT2819.C */
#define BASE 0x230
#define CONTROL_STATUS_REG (BASE + 4)
#define COMMAND_REG (BASE + 1)
#define DATA_PORT (BASE + 0)

#define BASE 0x230
#define CONTROL_STATUS_REG (BASE + 4)
#define COMMAND_REG (BASE + 1)
#define DATA_PORT (BASE + 0)
#include <conio.h>
/* #include "b:\modular\struct.h" */
#include <dos.h>
void InitDT2819(void);
unsigned char bin(char *string);
void Board2819Ready(void);
float CounterRate(short milliseconds);

void InitDT2819(void) {
   /* Initialize DT2819 */
   Board2819Ready();
   outp(CONTROL_STATUS_REG,0x00);
   Board2819Ready();
   outp(COMMAND_REG,0XFF);
   Board2819Ready();
   outp(COMMAND_REG,0x5F);
   /* Select 1 MHz clock */
   Board2819Ready();
   outp(CONTROL_STATUS_REG,0x40 /*bin("01000000")*/);
   /* Go to Master Mode Register */
   Board2819Ready();
   outp(COMMAND_REG,0x17 /*bin("00010111")*/);
   /* low byte */
   /* FOUT=F1, no comparator, ToD disabled */
   Board2819Ready();
   outp(DATA_PORT,0x00 /*bin("00000000")*/);
   /* high byte */
   /* BCD div, no auto-inc, 8-bit bus, FOUT enab. */
   /* FOUT divide by 1 */
   Board2819Ready();
   outp(DATA_PORT,0xC1 /*bin("11000001")*/);
   /* Set up Mass Spec counters */
   /* Timer 1 */
   /* point to counter 1 mode register */
   Board2819Ready();
   outp(COMMAND_REG,0x01);
   /* no hardware trig, reload from load, count rep.*/
   /* bin count, count up, TC active high pulse     */
   Board2819Ready();
   outp(DATA_PORT,0x29 /*bin("00101001")*/);
   /* active high TCN-1 gate, count rising edge,    */
   /* count source 1, (the mass spec)               */
   Board2819Ready();
   outp(DATA_PORT,0x21 /*bin("00100001")*/);
   /* point to counter 1 load register */
   Board2819Ready();
   outp(COMMAND_REG,0x09);
   /* Fill with 0 */
```

```
    Board2819Ready();
    outp(DATA_PORT,0x00);
    Board2819Ready();
    outp(DATA_PORT,0x00);
    /* Timer 2 */
    /* point to counter 2 mode register */
    Board2819Ready();
    outp(COMMAND_REG,0x02);
    /* no hardware trig, reload from load, count once*/
    /* bin count, count up, TC pulse inactive     */
    Board2819Ready();
    outp(DATA_PORT,0x08 /*bin("00001000")*/);
    /* no gating, count rising edge, source = TCN-1 */
    Board2819Ready();
    outp(DATA_PORT,0x00 /*bin("00000000")*/);
    /* point to counter 2 load register */
    Board2819Ready();
    outp(COMMAND_REG,0x0A);
    /* Fill with 0 */
    Board2819Ready();
    outp(DATA_PORT,0x00);
    Board2819Ready();
    outp(DATA_PORT,0x00);
    Board2819Ready();

    /* Set up timers */
    /* Timer 4 */
    /* point to counter 4 mode register */
    Board2819Ready();
    outp(COMMAND_REG,0x04);
    /* no hardware trig, reload from Load & Hold */
    /* count rep, bin count, count down, TC pulse*/
    Board2819Ready();
    outp(DATA_PORT,0x61 /*bin("01100001")*/);
    /* no gate, count rise, source=F3 (10 KHz) */
    Board2819Ready();
    outp(DATA_PORT,0x0D /*bin("00001101")*/);


    /* Timer 5 */
    /* point to counter 5 mode register */
    Board2819Ready();
    outp(COMMAND_REG,0x05);
    /* no hardware trig, reload from Load, count once*/
    /* bin count, count down, TC toggle */
    Board2819Ready();
    outp(DATA_PORT,bin("00000010"));
    /* no gate, count rise, source = TCN-1 */
    Board2819Ready();
    outp(DATA_PORT,0x00 /*bin("00000000")*/);
    /* point to counter 5 load register, fill with 2 */
    Board2819Ready();
    outp(COMMAND_REG,0x0D);
    Board2819Ready();
    outp(DATA_PORT,0x02);
    Board2819Ready();
    outp(DATA_PORT,0x00);
}
```

```
unsigned char bin(char *string) {
   unsigned char byte = 0;
   unsigned char *bit = string;
   short i;
   for(i=0;i<8;i++,bit++) if(*bit-'0') byte=(byte|(0x80>>i));
   return byte;
}
void Board2819Ready(void) {
   char byte;
   /* Wait until bit 7 of the Status Register clears to 0.     */
   /* Board requires 1.5 microseconds between reads and writes */
   do {
      byte = inportb(CONTROL_STATUS_REG);
   } while(128 & byte);
}

float CounterRate(short milliseconds) {
   unsigned char low,high,byte;
   unsigned short counter1,counter2;
   long total;
   float rate;

   if(milliseconds==0) milliseconds = 1;
   /* point to counter 4 load register*/
   Board2819Ready();
   outp(COMMAND_REG,0x0C);
   /* Fill with 1 millisecond delay */
   Board2819Ready();
   outp(DATA_PORT, (10*milliseconds) % 0x100 );
   Board2819Ready();
   outp(DATA_PORT, (10*milliseconds) / 0x100 );
   /* point to counter 4 Hold register */
   Board2819Ready();
   outp(COMMAND_REG,0x14);
   /* Fill with gate time in milliseconds */
   Board2819Ready();
   outp(DATA_PORT, 1 );
   Board2819Ready();
   outp(DATA_PORT, 0 );

   /* set output of counter 5 low */
   Board2819Ready();
   outp(COMMAND_REG,bin("11100101"));

   /* load and arm counters 1,2,4,5 */
   Board2819Ready();
   outp(COMMAND_REG,0x7B /*bin("01111011")*/);
   Board2819Ready();
   /*Watch for TC 5 going high */
   do {
      byte = inp(COMMAND_REG);
   } while(!(byte & 32));
   /* Output went high */

   /* Disarm and save 1,2,4,5 */
   Board2819Ready();
   outp(COMMAND_REG,0x9B /*bin("10011011")*/);
```

```
    /* Get counts */
    /* point to counter 1 hold register */
    Board2819Ready();
    outp(COMMAND_REG,0x11);
    Board2819Ready();
    low = inp(DATA_PORT);
    Board2819Ready();
    high = inp(DATA_PORT);
    counter1 = 256*high + low;
    Board2819Ready();
    /* point to counter 2 hold register */
    outp(COMMAND_REG,0x12);
    Board2819Ready();
    low = inp(DATA_PORT);
    Board2819Ready();
    high = inp(DATA_PORT);
    counter2 = 256*high + low;
    total = 65356*counter2 + counter1;
    rate = ((float) total) * 1000.0/ milliseconds;
    return rate;
}
```